**Multi-Valued Logic Gates**

**using Verilog HDL**

Zhanibek Kalkayev

201434867

**Table Of Contents**

**Acknowledgement**

I would like to express my sincere gratitude to my professor and the coordinator of Summer 2020 internships Vipin Kizheppatt, School of Engineering and Digital Sciences of Nazarbayev Univeristy and the Electrical and Computing Engineering faculty for the support and possibility to work on university projects during the global COVID-19 pandemic. Moreover, I would like to express lots of appreciation and thanks to my professor Mohammad Hashmi, who has showed high level of patience, support and guidance despite anything.

Additionally, one time more many thanks to our professor Vipin Kizheppatt for awesome three years with us and I sincerely wish you all the best in your career and personal life.

**1. Introduction**

**Motivation**

Digital circuits are designed and implemented using the binary logics, which is shown as a bivalent logic which must be either “absolute true” or “absolute false”, because of its simplicity and convenience. For example, for the statement “the earth is round” the binary logicwill give a value of “true”. Moreover, the voltage mode binary logic allows for optimal noise margins that come from using only the two voltage references for the logic values, thus making it simpler to analyze between a logic “1” and logic “0”. However, there are some circumstances where the binary logic breaks down. For instance, traffic lights have 3 states (green, yellow,red), which is reasonably problematic to implement using bivalent logic. Hereby, the multi-valued logics comes.

**Objectives**

Thus, the main objective of the current internship work is to get acknowledged with Multi-Valued Logic and enhance the understanding in both Multi-Valued Logics and Hardware Description Language.

**Contribution**

In science, a non-classic many-valued logic is represented as a propositional calculus in

which no restrictions about the number of truth values are present: it allows larger sets of truth

degrees. In modern era, the main MVL systems are represented by Łukasiewicz logics, Gödel

logics, t-Norm based systems, three-valued systems, Dunn/Belnap’s 4-valued system, and

product systems. In electronic devices’ structure, an interconnection cost and delay of circuits is

a big challenge. Thus, high-radix systems could be beneficial towards solving the problem by

providing a better relationship between circuits. The same phenomenon is present in other

relevant areas such as signal processing, data structures and discrete system algorithms.

For example, considering decimal number “322” and its representations in other

radixes, it is seen that the number of digits is inversely proportional to the base of the number

itself: as radix increases amount of digits decreases.

322 10 = 11002 4 = 101000010 2

As it can be seen from the table, the amount of functions that can be represented by

quaternary is much larger than binary number. Accordingly, MVL not only reduces the total

number of connections, but also causes reduction of total number of gates compared to its

binary implementation, as several binary gates potentially can be replaced by a single MVL

gate.

Though the potential of Multi-Valued Logic gates looks very promising because of its several

major advantages, the feasibility of the systems depends on these factors:

* Availability of reliable gateway implementations.
* Adequate synthesis tools and techniques.

**Structure of the report**

The report is organized as follows: Section 2 provides the overall background information about the Multi-Valued Logic with its application, advantages and disadvantages. Section 3 provides details of the internship work including all the steps of MVL development: implementation of basic logic gates in MVL and developing of the quaternary Arithmetical Logical Unit (ALU) including all the truth tables and graphs. Section 4 concludes the report and provides the details on future efforts. The last Section 5 includes Verilog codes of basic logic gates and ALU in MVL.

2. Background

To the date, researchers and developers has proposed MVL for devices such as

memories, combinational circuits (adders, multipliers) and programmable devices. If the first

discoverer of MVL application on memory was Intel, releasing their products which could store

two bits of information per cell with four voltage levels, Samsung launched their NAND-based

flash based memory in face of 840 Evo Series SSDs that could store 3 bits of information per

cell. Such technologies turned out to be beneficial because of its possibility to pack larger

amount of information into small areas. The technology was referred as Multi-Level Cell, which

allowed to store two or more bits of information per cell instead of one by using intermediate

voltage levels. Consequently, it had positive impact on cost-capacity relationship.

While talking about standard gates, despite the multiple researches and application

attempts, only few of them could be classified as real alternatives to a regular basic CMOS

(Complementary Metal-Oxide-Semiconductor) gates. Such implementations of Multi-Valued

Logic circuits are categorized as voltage-mode and current-mode operations.

The work principle of voltage-mode implementation is based on having several stable

intermediate voltage levels between fully charged and fully discharged, which sometimes

requires nonstandard technologies. The main advantage of such technique is little amount of

power consumption, which happens only during logic level switching.

On the other side, the work principle of current-mode implementation is based on

Kirchhoff’s current law, where logic levels are represented by different currents that can be

involved on addition and subtraction operations in order to obtain the desired output. The

major advantage of such technique is a simplicity of arithmetic operations, while a higher

power consumption is tend to be a disadvantage, as the current constantly flows through the

circuit.

3. Details of the internship work

1. Introduction

Right after the getting acknowledged with Multi-Valued Logic theory and brief background,

the next step was to design basic and foundation gates such as NOT, AND, NAND, OR,

NOR, XOR and XNOR gates. As a base for logic the quaternary logic with four logic levels

(0, 1, 2, 3) was chosen. Observing theoretical background of quaternary algebra, the truth

table of basic gates were derived using default functional quaternary operators such as MIN,

MAX and Sum Of:

1.1 NOT Gate

As well as in binary logic, NOT Gate has one input and produces one output in quaternary

algebra. Truth table is given below:

**table 1**

1.2 AND and NAND Gates

AND Gate is described as a basic digital logic gate which implements logical conjuction.

In binary logic, the high signal appears only in case of all inputs are high. As in quater-

nary algebra there are 4 levels of truth, the output is determined by comparing both binary

values of inputs. For example, for inputs 1 and 2 we obtain binary values of 01 and 10,

consequently using classic binary AND gate for 0 and 1, 1 and 0, we get the final value of 00,

thus the output is 0. NAND Gate is the opposite of AND gate. Both AND and NAND Gates

have two inputs and produce one output in quaternary algebra. Truth tables are given below:

**table 2**

**1.3** OR and NOR Gates

OR Gate is described as a basic digital logic gate which implements logical disjunction. In

binary logic, the high signal appears in any case while one of the inputs is high. As it was

mentioned before, the output in quaternary algebra is determined by comparing both binary

values of inputs. For inputs 1 and 2 we obtain binary values of 01 and 10, consequently using

classic binary OR gate for 0 and 1, 1 and 0, we get the final value of 11, thus the output is

3. NOR Gate is the opposite of OR gate. Both OR and NOR Gates have two inputs and

produce one output in quaternary algebra. Truth tables are given below:

**table 3**

**1.4 XOR and XNOR Gates**

XOR (EOR, EXOR, Exclusive OR) is characterized as a digital logic gate which gives a high

signal output when the number of true inputs is odd. To find the output of XOR Gate in

quaternary logic, function Sum Of is used. For inputs 1 and 2 (01 and 10), we add first

digits and second digits to each other: 0 + 1, 1 + 0, thus obtaining 11, which is decimal

3. XNOR Gate is the opposite of XOR. Both XOR and XNOR Gates have two inputs and

produce one output in quaternary algebra. Truth tables are given below:

**table 4**

1. Arithmetic Logical Unit

The final step of the whole process was to implement and design a simple arithmetical logical unit (ALU) with functionality such as addition, subtraction and switching for quaternary logic system. The ALU reads input operands A and B, and the operation to perform is selected using **ALU\_Sel** input, which acts as an Opcode. The final results is to be shown on **ALU\_Out** for final output, and **CarryOut** flag for identify if carry was found during addition. The ALU comprises following functions:

* Addition
* Subtraction
* Shift right/left
* Comparison

As there are no quaternary logic system in Verilog HDL and wave analyzer software such as GTKWave or ModelSim, the main algorithm for addition and subtraction was designed as implementation of addition and subtraction for 4-radix numbers:

1. Find if there are extra tens by adding first digits of A and B - **A % 10 + B % 10 > 3;**

2. Use **temp** register to add tens of A and B to identify if carry exists.

3. If carry exists, use modulo operation for **tens** to obtain proper result, and equalize **hun** (hundreds) to 1.

4. The final result is obtained by simple arithmetic equation **ALU\_Result = ten \* 10 + (A % 10 + B % 10) % 4;** The same algorithm was used for subtraction reusing **ten**s as **borrows**.

4. Conclusion

The following internship work demonstrates how Multi-Valued Logic Systems can be implemented using Verilog Hardware Description Language. The MVL proposes wide range of possibilities and advantages over binary logic such as increased data density, computational ability which gives plenty of potential to the system. Despite the fact that MVL should be preferable over binary system, some obstacles in the form of a lack of materials were faced. Nevertheless, the system provides range of possibilities in future developments of hardware engineering field. It server both as to solve binary logic systems more efficiently and to design electronic circuits that possess more than two discrete levels of signals, which includes mentioned above many-valued memories, arithmetic circuits and FPGAs.

Generally, the following internship has provided lots of opportunity to enhance the knowledge about how things are arranged in hardware engineering. I have practiced a lot on coding on Verilog HDL, read lots of materials and different researches on the topic, watched hours of valuable tutorials, learned how to properly test my code and how to manage it in an effective way. I personally believe I made a step ahead to become an electrical and electronics engineer. However, there are plenty of work to do. The current topic could be revealed more, if there was no time limit. For example, application of the designed quaternary logic gates, or developing the quaternary multiplexer as it seem more attractive over basic logic gates because of its better reliability and reduced complexity.

5. Weekly Activities

|  |  |  |  |
| --- | --- | --- | --- |
| Week Number | Start Date: End Date | Brief description of activity | Number of hours spent on each activity |
| 1 | June 2 : June 8 | To get acknowledged with Multi-Valued Logic Gates + Report | ~40 |
| 2 | June 13 : June 25 | Implementing basic logic circuits in MVL (Verilog) + Report and Presentation | ~80 |
| 3 | July 3 : July 20 | Implementing Quaternary Arithmetic and Logic Unit with some functionality such as switching/addition/subtraction + Report | ~80 |

6. Appendix

module **AND**(

input [1:0] A,

input [1:0] B,

output reg [1:0] Y

);

always @\*

begin

Y[0] = A[0] & B[0];

Y[1] = A[1] & B[1];

end

endmodule

module **NAND**(

input [1:0] A, B,

output reg [1:0] Y

);

always @\*

begin

Y[0] = ~(A[0] & B[0]);

Y[1] = ~(A[1] & B[1]);

end

endmodule

module **NOR**(

input [1:0 ]A, B,

output reg [1:0] Y

);

always @\*

begin

Y[0] = ~(A[0] || B[0]);

Y[1] = ~(A[1] || B[1]);

end

endmodule

module **NOT**(

input [1:0] A,

output reg [1:0] Y

);

always @\*

begin

Y[1] = ~A[1];

end

endmodule

module **OR**(

input [1:0] A, B,

output reg [1:0] Y

);

always @\*

begin

Y[0] = A[0] || B[0];

Y[1] = A[1] || B[1];

end

endmodule

module **XNOR**(

input [1:0] A, B,

output reg [1:0] Y

);

always @\*

begin

Y[0] = ~(A[0] + B[0]);

Y[1] = ~(A[1] + B[1]);

end

endmodule

module **XOR**(

input [1:0] A, B,

output reg [1:0] Y

);

always @\*

begin

Y[0] = A[0] + B[0];

Y[1] = A[1] + B[1];

end

endmodule

module **Qalu**(

input [6:0] A,B, // ALU 8-bit Inputs

input [3:0] ALU\_Sel,// ALU Selection

output [6:0] ALU\_Out, // ALU 8-bit Output

output CarryOut // Carry Out Flag

);

reg [6:0] ALU\_Result;

reg [6:0] temp;

reg [6:0] ten;

reg [6:0] hun;

assign ALU\_Out = ALU\_Result; // ALU out

assign CarryOut = hun; // Carryout flag

initial

begin

ALU\_Result = 0;

ten = 0;

temp = 0;

end

always @(\*)

begin

case(ALU\_Sel)

4'b0000:

begin

if (A%10 + B%10 > 3)

ten = 1;

temp = ten + (A - A%10)/10 + (B-B%10)/10;

ten = (temp)%4;

if (temp > 3)

hun = 1;

else

hun = 0;

ALU\_Result = ten \* 10 + (A%10 + B%10)%4;

end

4'b0001:

begin

if(A%10 < B%10)

ten = 1;

else

ten = 0;

ten = (A - A%10)/10 - (B-B%10)/10 - ten;

if(A%10 < B%10)

ALU\_Result = ten\*10 + ((A%10 - B%10) + 4);

else

ALU\_Result = ten\*10 + (A%10 - B%10);

end

4'b0010:

ALU\_Result = A%10\*10; // shift left

4'b0011:

ALU\_Result = (A - A%10)/10; // shift right

4'b0100:

ALU\_Result = (A>B)? 1 : 0;

endcase

end

endmodule