GATES IN INSTANTANEOUS NOISE BASED LOGIC: CREATING THE XOR/XNOR GATE

A Thesis

by

MOHAMMAD BASIM A. KHREISHAH

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Laszlo B. Kish
Committee Members,	Mi Lu
	Jun Zou
	Ivan Ivanov
Head of Department,	Costas N. Georghiades

August 2023

Major Subject: Electrical Engineering

ABSTRACT

In this research, we propose a new method of applying the XOR and XNOR gates on exponentially large superpositions in Instantaneous Noise-Based Logic. These new gates are repeatable, and they can achieve an exponential speed up in computation with a polynomial requirement in hardware complexity. Applying these basic operations is pivotal for using the Instantaneous Noise Based Logic in different applications. We will be using mathematical proofs to show how these gates work and their generality when applied on any superposition. We will show that every XOR/XNOR gate requires 4 multiplications to do an O(2^N) number of parallel logic operations. Finally, we will show examples about how to cascade and to apply different logic gates on any superposition.

DEDICATION

This work is dedicated to my mother, father and three siblings, for their great support and encouragement throughout all my academic journey.

ACKNOWLEDGMENTS

First of all, I thank all mighty Allah for the endless blessings bestowed upon me, including very supportive family, advisors and friends.

I would like to thank Prof. Laszlo Kish for his impeccable academic insight and continuous support, as he supervised and reviewed this work. His teachings has opened my mind and influenced me greatly to try my best in conducting research in a creative and professional way.

Also, I would like to thank Dr. Walter Daugherity for the valuable discussions that we had during this research, and for sharing his valuable expertise. I would also like to thank him for his advice and revision while conducting this research.

I thank my committee members, Prof. Jun Zou, Prof. Ivan Ivanov and Prof. Mi Lu for taking the time to review this work.

I thank my good friends Hasan Ibrahim and Mohammad Nasim for providing valuable advice and support in academia and on a personal level.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by Prof. Laszlo Kish of the department of Electrical Engineering. All work for the dissertation was completed by the student.

Funding Sources

All work was completed independently without financial support.

NOMENCLATURE

NBL	Noise Based Logic
INBL	Instantaneous Noise Based Logic
RTW	Random Telegraph Waves
RNS	Reference Noise System

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGMENTS iv

CONTRIBUTORS AND FUNDING SOURCES v

NOMENCLATURE vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

1. INTRODUCTION AND LITERATURE REVIEW 1

    1.1 Classic Logic VS Noise-Based Logic 1
    1.2 Instantaneous Noise-based Logic 3
        1.2.1 Random Telegraph Waves 3
        1.2.2 Classical set up of Instantaneous Noise-based Logic 3
            1.2.2.1 The Probabilistic definitions of Random Telegraph Waves 4
        1.2.3 Strings and the Representation of Numbers 5
        1.2.4 Superpositions 6
        1.2.5 Circuit Realizations of Random Telegraph Waves 7
    1.3 NOT and CNOT gates in Instantaneous Noise Based Logic 8
        1.3.1 The NOT Gate 8
        1.3.2 The CNOT Gate 9

2. CREATING THE XOR AND THE XNOR GATES IN INBL 15

    2.1 Problem definition 15
    2.2 Solving the Dependency of the output bit 16
    2.3 Applying the XOR gate 17
    2.4 Applying the XNOR gate 18
    2.5 Cascading XOR gates 20
    2.6 Cascading an XOR gate with a CNOT gate 22

3. CONCLUSIONS AND FUTURE WORK 25

    3.1 Conclusions 25

3.2 Future work 25

REFERENCES 26

LIST OF FIGURES

FIGURE Page

1.1 This is a classic logic scheme. The voltage axis is normalized such that the supply voltage equals 1. The lower level of power dissipation is scaled with the voltage supply. The time axis is measured in clock cycles. 1

1.2 Part of a NBL processor. 2

1.3 RTW example. The dashed line represents that beginning of a new clock cycle. At the beginning of a clock cycle, the RTW has a 50% chance to change its value, and 50% chance to stay the same. 4

1.4 Implementation of a Universe using RTWs 7

1.5 A typical system in INBL that uses reference wires 8

1.6 Implementation of a NOT Gate 9

1.7 General System before applying a CNOT Gate 10

1.8 CNOT Gate 11

1.9 Cascaded CNOTs: applying $CNOT_{fh}$ then $CNOT_{if}$ 13

1.10 Cascaded CNOTs: applying $CNOT_{if}$ then $CNOT_{fh}$ 14

2.1 A general system before applying XOR Gate 15

2.2 Producing an independent output bit in a superposition 17

2.3 The XOR gate 18

2.4 The XNOR gate using the NOT operation on the XOR gate 19

2.5 An alternate implementation of the XNOR gate 20

2.6 Cascaded XOR gates circuit. The inputs and outputs can be of any value. This is just an illustration to show the intended operation and does not mean that inputs or outputs have a singular value. 21

2.7 Implementation of cascaded XOR gates 22

2.8 Cascading CNOT gate with an XOR gate 22

2.9 Applying $CNOT_{di}$ 23

2.10 Final implementation of cascaded CNOT and XOR gates 24

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Classic Logic VS Noise-Based Logic

Recently, the increasing use of electronics in high-performance applications has led to a growing demand for energy-efficient devices that can handle large amounts of computation. As the electronic devices continue to reach their minimum possible size, the computation demand has been increasing at a rapid pace. To address this issue, researchers have been developing new materials, designs and technologies to create devices with a low power cost and a high computation capability. In this thesis, Noise Based Logic (NBL) is introduced as an alternate method of computation. The main motivation behind NBL is to surpass the limitations of modern computation devices and to tackle challenging computation problems in an efficient manner.

In Classic logic schemes, binary values are assigned based on voltage levels. Two voltage thresholds are used to decide the binary value; a high voltage value (U_{high}) and a low voltage value (U_{low}). Figure 1.1 illustrates this idea. If the voltage level is above U_{high}, then that indicates logic 1, and if the voltage is less than U_{low}, then that indicates logic 0.

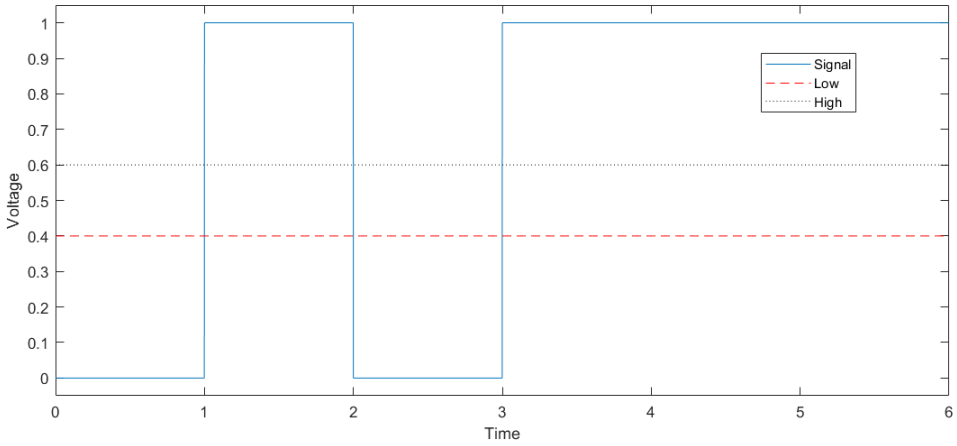

Figure 1.1: This is a classic logic scheme. The voltage axis is normalized such that the supply voltage equals 1. The lower level of power dissipation is scaled with the voltage supply. The time axis is measured in clock cycles.

NBL [1] was inspired by the stochastic nature of neuron signals, since the brain is incredibly efficient at processing large amounts of data. Therefore, bits in Noise-Based Logic (NBL) are stochastic functions of time and can be processed simultaneously. In NBL, each bit requires two independent orthogonal noise sources, a noise source that represents the high value of a bit, and another noise source that represents the low value of a bit. A reference system generates the independent orthogonal noises required for operations. Figure 1.2 shows part of a typical NBL processor. The reference system needs to be supplied to all gates.

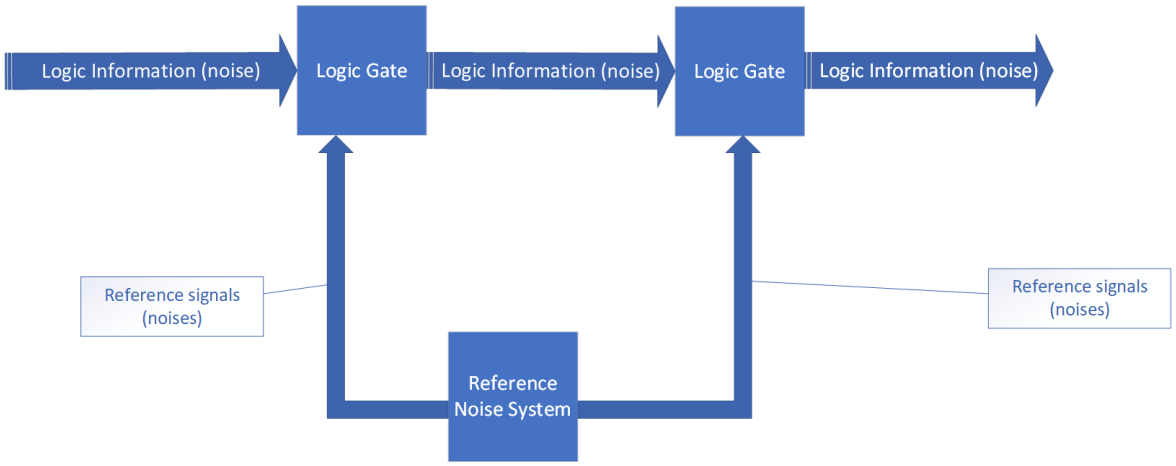

Figure 1.2: Part of a NBL processor.

Power dissipation comparison: In conventional logic, clock distribution is one of the most significant sources of dynamic power loss, because it is supplied everywhere in chips and it requires the highest switching frequency. It also requires relatively high voltage so that it can switch fast enough, as there is a trade off between switching frequency and voltage amplitude. In NBL, the reference wires need to be supplied throughout the whole system, but it requires significantly less voltage amplitude. Low voltage requirement makes NBL more efficient because the power dissipation is proportional to the squared value of the voltage. Leakage currents are also reduced in NBL because the voltage amplitude is less.

Computation capability comparison: NBL inherently posses the capability to apply operations

in a parallel fashion (this shall be discussed later in detail), unlike conventional logic, where the bits need to be processed separately.

1.2 Instantaneous Noise-based Logic

Since NBL utilizes independent orthogonal stochastic processes, it requires time averaging to produce the logic information which is inefficient. To eliminate the need for time averaging, Instantaneous Noise-based Logic (INBL) [2–9] was proposed. INBL is a class of NBL, where computations only require basic mathematical operations to produce the output. INBL has a higher computation speed and less hardware complexity than NBL.

Types of INBL include Random Telegraph Waves (RTW) and Random spike train based logic. This thesis will focus on RTW as it is relevant to the main work. The usual set up for INBL will be explained using RTWs. In the following sections, all relevant literature will be reviewed in detail.

1.2.1 Random Telegraph Waves

Random Telegraph Waves (RTW) are driven by a periodic clock, such that their values only change when a new clock cycle begins. Each RTW has only two values, positive and negative. These two values are also equal in magnitude. At the beginning of each clock cycle, RTWs change their values randomly with a probability of 0.5. Figure 1.3 illustrates the simplest type of RTWs where the wave oscillates between 1 and -1 randomly and with equal probabilities. These values for the RTW will used throughout the rest of this thesis.

1.2.2 Classical set up of Instantaneous Noise-based Logic

Since RTWs are relevant to this work, they will be used as an example to illustrate a typical INBL scheme. Each bit has two values (high and low), thus, each bit requires two independent RTWs to represent each of these values. The independent RTWs are generated by a Reference Noise System (RNS), so that they can be used throughout the whole system and not just in the generation phase. If a system has a resolution of *N* noise bits, then 2*N* independent RTWs are needed to represent such a system. Each RTW reference is denoted by $R_{i,j}(t)$, where $[i]$ is the bit significance number and the $[j]$ is the bit value ($j \in \{1, 0\}$).

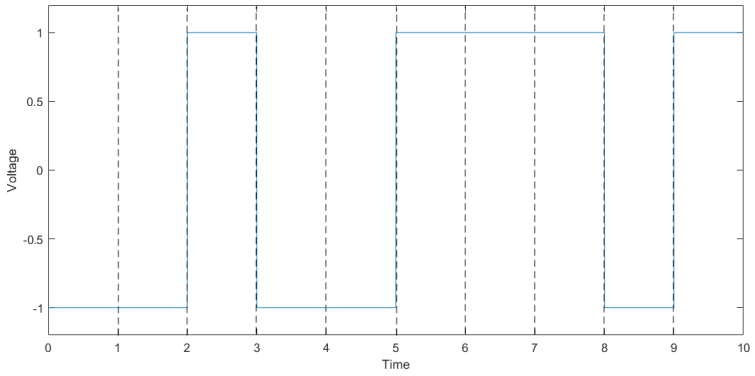

Figure 1.3: RTW example. The dashed line represents that beginning of a new clock cycle. At the beginning of a clock cycle, the RTW has a 50% chance to change its value, and 50% chance to stay the same.

For example, to represent a 2-bit system, Four independent RTWs need to be supplied by the RNS. The following RTWs are needed for such a system:

$$R_{1,0}(t) \rightarrow \text{Bit significance 1 with binary value 0,}$$
$$R_{1,1}(t) \rightarrow \text{Bit significance 1 with binary value 1,}$$
$$R_{2,0}(t) \rightarrow \text{Bit significance 2 with binary value 0,}$$
$$R_{2,1}(t) \rightarrow \text{Bit significance 2 with binary value 1}$$

1.2.2.1 The Probabilistic definitions of Random Telegraph Waves

Each RTW must have an average of zero:

$$\langle R_{i,j}(t) \rangle = 0 \tag{1.1}$$

Where the $\langle\rangle$ operator is used to indicate the average of a time varying signal.

The amplitude of all RTWs are assumed to be one, and by extension the multiplication of a RTW with itself will give a one:

$$R_{i,j}^2(t) = 1 \tag{1.2}$$

All the RTWs must be independent and they must be orthogonal to each other, meaning that the cross correlation between the RTWs must be zero:

$$\langle R_{i,j}(t)R_{n,m}(t)\rangle = 0 \tag{1.3}$$

such that $i \neq n$ or $j \neq m$.

The product of any number of RTWs is a new RTW that is orthogonal to all other RTWs, including the RTWs that contribute to the product. The following equations prove and illustrate this idea:

$$R_{i,j}(t)R_{n,m}(t) = R_{k,l}(t) \tag{1.4}$$

such that such that $i \neq n$ or $j \neq m$.
$R_{k,l}(t)$ is orthogonal to $R_{i,j}(t)$ and $R_{n,m}(t)$:

$$\langle R_{i,j}(t)R_{k,l}(t)\rangle = \langle R^2_{i,j}(t)R_{n,m}(t)\rangle = \langle R_{n,m}(t)\rangle = 0 \tag{1.5}$$

$$\langle R_{n,m}(t)R_{k,l}(t)\rangle = \langle R^2_{n,m}(t)R_{i,j}(t)\rangle = \langle R_{i,j}(t)\rangle = 0 \tag{1.6}$$

The product $R_{k,l}(t)$ is also orthogonal to any other RTW $W_{a,b}(t)$, where $a \notin \{i, n\}$:

$$\langle R_{a,b}(t)R_{k,l}(t)\rangle = \langle R_{a,b}(t)R_{i,j}(t)R_{n,m}(t)\rangle = 0 \tag{1.7}$$

1.2.3 Strings and the Representation of Numbers

Suppose that the a RNS produces 2*N* of RTWs, then any binary number {*B*} in the range of [0,2^N] can be represented. To represent a binary number, the RTWs corresponding to the binary value need to be multiplied together:

$$\prod_{i=1}^{N} R_{i,j(i,B)}(t) \tag{1.8}$$

Where *i* is the bit significance, and *j* is the bit value and it is a function of *B* and *i*.

For example to represent the number $(1010)_2$, then 4 RTWs need to be supplied by the RNS and their product will be:

$$B_{(1010)_2} \rightarrow R_{1,0}(t)R_{2,1}(t)R_{3,0}(t)R_{4,1}(t) \tag{1.9}$$

The product in equation 1.8 shall be called a string and denoted by $S_B(t)$. It is important to note that each string must be the product of *N* number of independent RTWs, where *N* is the number of the bits.

1.2.4 Superpositions

A superposition $Y(t)$ is the summation of strings:

$$Y(t) = \sum_{u=1}^{U} S_{B_u}(t) \tag{1.10}$$

Where B_u is the binary number corresponding to the index *u*, and *U* is the number of strings that contribute to the overall superposition. The following observations should be noted about superpositions:

- $1 \leq U \leq 2^N$
- $1 \leq B_u \leq 2^N$
- The number of all possible sub-spaces of a superposition S_{Total}:

$$S_{Total} = \sum_{k=1}^{2^N} \binom{2^N}{k} = 2^{2^N} - 1 \tag{1.11}$$

This means that a superposition can represent more than one string at the same time. This property is what gives NBL its potential in parallel computing; i.e. exponential parallel operations can be potentially realized. For example, the superposition that represents the set of binary numbers

$\{(010)_2, (110)_2, (111)_2\}$ will be:

$$Y(t) = R_{1,0}(t)R_{2,1}(t)R_{3,0}(t) + R_{1,0}(t)R_{2,1}(t)R_{3,1}(t) + R_{1,1}(t)R_{2,1}(t)R_{3,1}(t) \tag{1.12}$$

Note: From here on, the time notation will be omitted for convenience, as all the signals included should be functions of time.

Another example is the Achilles Heel algorithm which is an efficient algorithm that can be used to realize a superposition that includes all possible states, such a superposition is called a Universe and shall be denoted by U :

$$U = (R_{10} + R_{11})(R_{20} + R_{21})(R_{30} + R_{31})....(R_{N0} + R_{N1}) \tag{1.13}$$

1.2.5 Circuit Realizations of Random Telegraph Waves

As a consequence of the previous discussion, four simple circuit elements can be used to realize the hardware synthesis of RTWs: Addition, Subtraction, Multiplication and Division (Subtraction and Division will not be used in this thesis). This means that the implementation of RTWs is fairly simple. Figure 1.4 shows the implementation of a Universe of a three bit system.

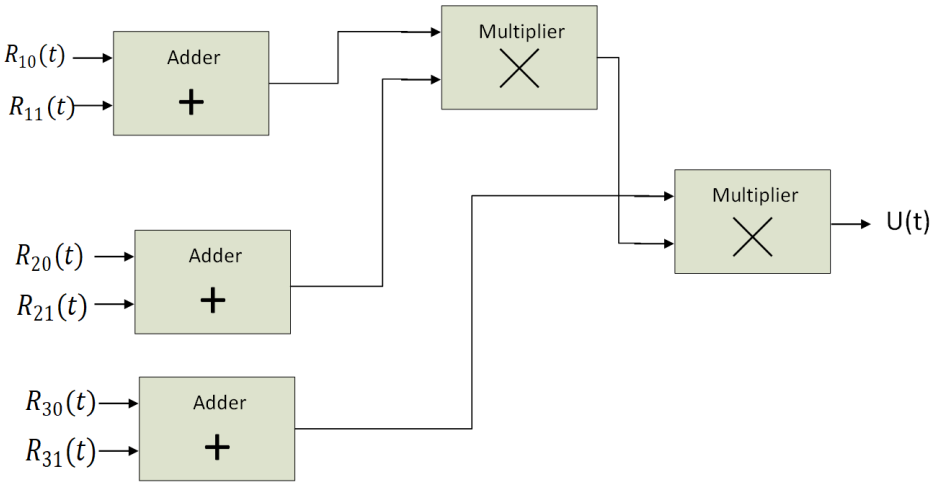

Figure 1.4: Implementation of a Universe using RTWs

1.3 NOT and CNOT gates in Instantaneous Noise Based Logic

Previously, the NOT and the CNOT gates were created by applying operations on the reference wires. Before discussing these methods, it will be beneficial to discuss what a general system should look like. Figure 1.5 illustrates a typical system, where the "Hyperspace Synthesizer" unit produces the output superposition *Y*.

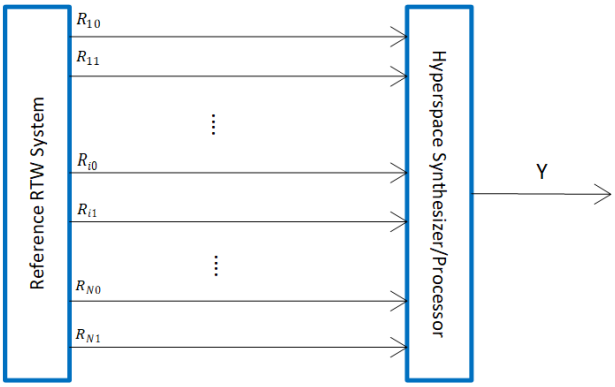

Figure 1.5: A typical system in INBL that uses reference wires

Let us suppose that we have the following arbitrary superposition:

$$Y = Y_0 R_{i0} + Y_1 R_{i1} \tag{1.14}$$

Where Y_0 and Y_1 are arbitrary superpositions that do not contain R_{i0} or R_{i1}. Furthermore, Y_0 and Y_1 can not be zero at the same time. Equation 1.14 shows that any arbitrary superposition can be represented in terms of any bit *i*. This general superposition can represent a Universe or any subset of it, and it will used to prove the generality of the gates.

1.3.1 The NOT Gate

To conduct a NOT operation [10] in INBL for a certain bit *i*, the high value (R_{i1}) and low value (R_{i0}) reference wires of that bit need to be multiplied by the multiplication of the high and low values ($R_{i1}R_{i0}$) of that bit. Figure 1.6 shows the implementation of the NOT gate.

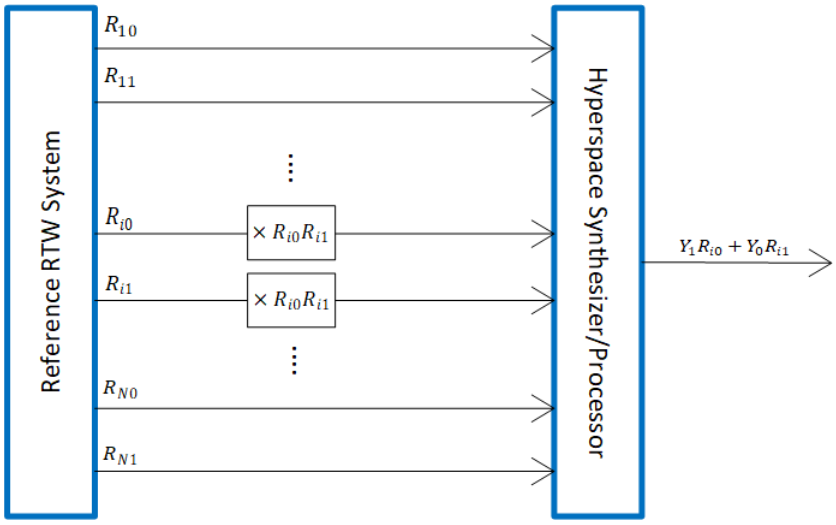

Figure 1.6: Implementation of a NOT Gate

If the NOT operation is applied on the superposition in equation 1.14, then the superposition becomes:

$$Y = Y_0 R_{i0} R_{i0} R_{i1} + Y_1 R_{i1} R_{i0} R_{i1} \tag{1.15}$$

By using equation 1.2, equation 1.15 becomes:

$$Y = Y_0 R_{i1} + Y_1 R_{i0} \tag{1.16}$$

By comparing equation 1.14 and equation 1.16, it is noticed that the NOT gate have been applied to all strings in the superposition.

1.3.2 The CNOT Gate

The Controlled NOT operation [10] requires a control bit and a target bit. When the control bit is high value, a NOT operation will be applied to the target bit, and when the control bit is low value, no operation will be applied to the target bit. Suppose that a system yields the following superposition:

$$Y = Y_0 R_{i0} R_{f0} + Y_1 R_{i0} R_{f1} + Y_2 R_{i1} R_{f0} + Y_3 R_{i1} R_{f1} \tag{1.17}$$

Where Y_0, Y_1, Y_2 and Y_3 are arbitrary superposition that do not contain bits *i* or *f*. This operation shall be denoted by $CNOT_{if}$. Figure 1.7 shows such a system.

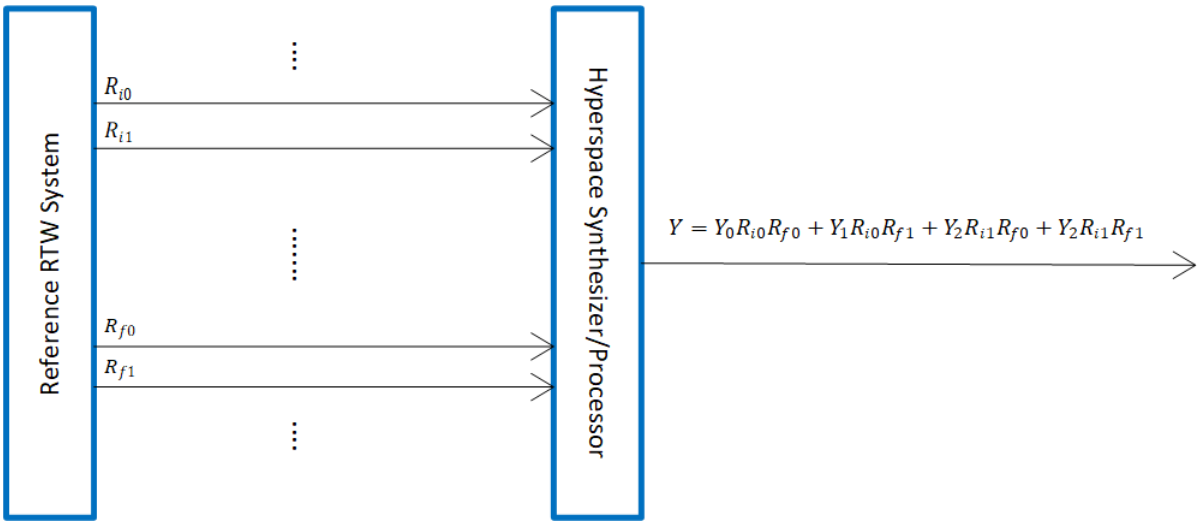

Figure 1.7: General System before applying a CNOT Gate

Suppose that in the previous superposition it is required to apply the CNOT gate such that bit *i* is the control bit and bit *f* is the target bit. This operation shall be denoted by $CNOT_{if}$. To apply the gate, all that is needed is to multiply the reference wire of the high value of the control bit (R_{i1}) by the multiplication of the high value and the low value of the target bit ($R_{f1}R_{f0}$). By substituting this operation in equation 1.17, the following superposition is produced:

$$Y = Y_0 R_{i0} R_{f0} + Y_1 R_{i0} R_{f1} + Y_2 R_{i1} (R_{f0} R_{f1}) R_{f0} + Y_3 R_{i1} (R_{f0} R_{f1}) R_{f1} \tag{1.18}$$

Using equation 1.2, equation 1.18 becomes:

$$Y = Y_0 R_{i0} R_{f0} + Y_1 R_{i0} R_{f1} + Y_2 R_{i1} R_{f1} + Y_3 R_{i1} R_{f0} \tag{1.19}$$

Figure 1.9 illustrates $CNOT_{if}$ and its output.

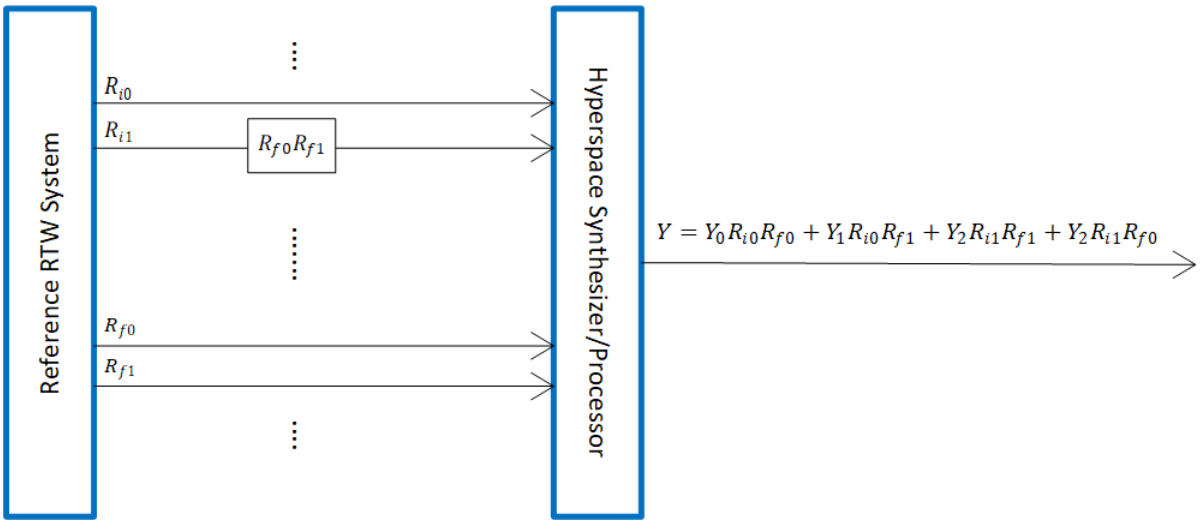

Figure 1.8: CNOT Gate

Cascaded CNOT operations shall be investigated next. Suppose that two CNOT gates need to be applied one after the other, then the sequence of operation must be considered and must be accounted for. There are two situations that arise from cascading CNOT gates:

- Situation1: The CNOT gates are non-interacting, and that means that the CNOT can be applied normally just as a single CNOT operation.Non-interacting CNOT gates will occur when the target bit of the CNOT operation that is being applied first, is not the control bit of the second CNOT operation.

- Situation2: The CNOT gates are interacting, and that means that the original method needs modification since the first CNOT will interfere with the second CNOT. Interacting CNOT gates will occur when the target bit of the CNOT operation that is being applied first is the control bit of the second CNOT operation.

It is best to explain such situations with examples.Assuming that a system generates the following superposition:

$$Y = Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} R_{h0} + Y_2 R_{i1} R_{f0} R_{h0} + Y_3 R_{i1} R_{f1} R_{h0} + Y_4 R_{i0} R_{f0} R_{h1} + Y_5 R_{i0} R_{f1} R_{h1} + Y_6 R_{i1} R_{f0} R_{h1} + Y_7 R_{i1} R_{f1} R_{h1} \tag{1.20}$$

Example of Situation1: Let us assume that $CNOT_{fh}$ will be applied first then $CNOT_{if}$ will be applied next. In this case, the CNOT gates are non-interacting and the gates can applied directly with no modification, because the control bit *i* of $CNOT_{if}$ has not changed. To apply $CNOT_{fh}$, the reference wire of the high value of the control bit (R_{f1}) is multiplied by the multiplication of the high value and the low value of the target bit ($R_{h1}R_{h0}$). To apply $CNOT_{if}$, the reference wire of the high value of the control bit (R_{i1}) is multiplied by the multiplication of the high value and the low value of the target bit ($R_{f1}R_{f0}$). According to these operations, equation 1.20 becomes:

$$\begin{aligned} Y = {} & Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} (R_{h1} R_{h0}) R_{h0} + Y_2 R_{i1} (R_{f1} R_{f0}) R_{f0} R_{h0} \\ & + Y_3 R_{i1} (R_{f1} R_{f0}) R_{f1} (R_{h1} R_{h0}) R_{h0} + Y_4 R_{i0} R_{f0} R_{h1} + Y_5 R_{i0} R_{f1} (R_{h1} R_{h0}) R_{h1} \\ & + Y_6 R_{i1} (R_{f1} R_{f0}) R_{f0} R_{h1} + Y_7 R_{i1} (R_{f1} R_{f0}) R_{f1} (R_{h1} R_{h0}) R_{h1} \end{aligned} \tag{1.21}$$

Using equation 1.2, equation 1.21 becomes:

$$Y = Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} R_{h1} Y_2 R_{i1} R_{f1} R_{h0} + Y_3 R_{i1} R_{f0} R_{h1} + Y_4 R_{i0} R_{f0} R_{h1} + Y_5 R_{i0} R_{f1} R_{h0} + Y_6 R_{i1} R_{f1} R_{h1} + Y_7 R_{i1} R_{f0} R_{h0} \tag{1.22}$$

Figure 1.9 illustrates the non-interacting CNOTs.

Example of Situation2: Let us assume that $CNOT_{if}$ will be applied first then $CNOT_{fh}$ will be applied next. In this case, the CNOT gates are interacting because the target bit *f* in the first CNOT is the control bit in the second CNOT. A modification is needed to get the correct results. In this setup, there is an indirect relationship between between bits *i* and *h*. Three special interactions appear here:

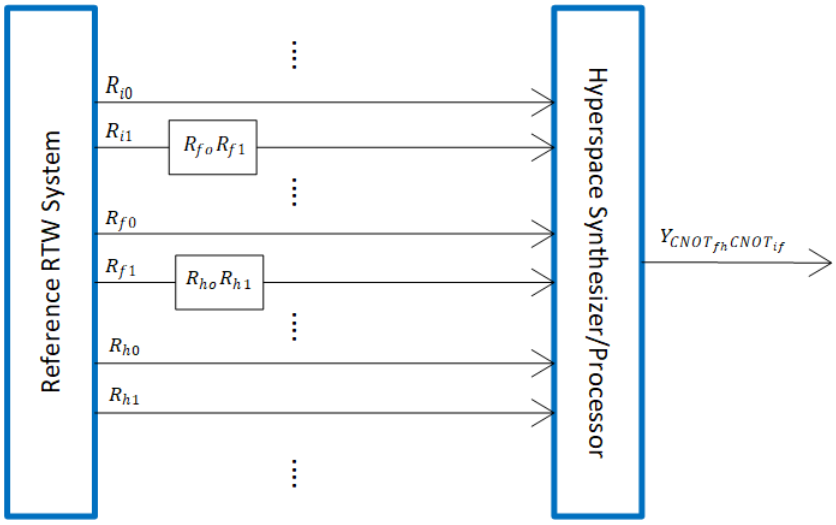

Figure 1.9: Cascaded CNOTs: applying $CNOT_{fh}$ then $CNOT_{if}$

- When *i* is high and *f* is low, bit *h* gets complemented.
- When *i* is high and *f* is high, bit *h* doesn't get complemented.
- When *i* is low and *f* is high, bit *h* gets complemented.

In order to apply the gates successfully, the same operations will be applied as if they were non-interacting, but to account for the three special cases, the high value of *i* (R_{i1}) needs to be multiplied by the low and high value of *h* ($R_{h1}R_{h0}$). Equation 1.20 becomes:

$$\begin{aligned} Y = {} & Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} (R_{h1} R_{h0}) R_{h0} + Y_2 R_{i1} (R_{f1} R_{f0}) (R_{h1} R_{h0}) R_{f0} R_{h0} \\ & + Y_3 R_{i1} (R_{f1} R_{f0}) (R_{h1} R_{h0}) R_{f1} (R_{h1} R_{h0}) R_{h0} + Y_4 R_{i0} R_{f0} R_{h1} + Y_5 R_{i0} R_{f1} (R_{h1} R_{h0}) R_{h1} \\ & + Y_6 R_{i1} (R_{f1} R_{f0}) (R_{h1} R_{h0}) R_{f0} R_{h1} + Y_7 R_{i1} (R_{f1} R_{f0}) (R_{h1} R_{h0}) R_{f1} (R_{h1} R_{h0}) R_{h1} \end{aligned} \tag{1.23}$$

Using equation 1.2, equation 1.23 becomes:

$$Y = Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} R_{h1} + Y_2 R_{i1} R_{f1} R_{h1} + Y_3 R_{i1} R_{f0} R_{h0} + Y_4 R_{i0} R_{f0} R_{h1} + Y_5 R_{i0} R_{f1} R_{h0} + Y_6 R_{i1} R_{f1} R_{h0} + Y_7 R_{i1} R_{f0} R_{h1} \tag{1.24}$$

Figure 1.10 illustrates such a case.

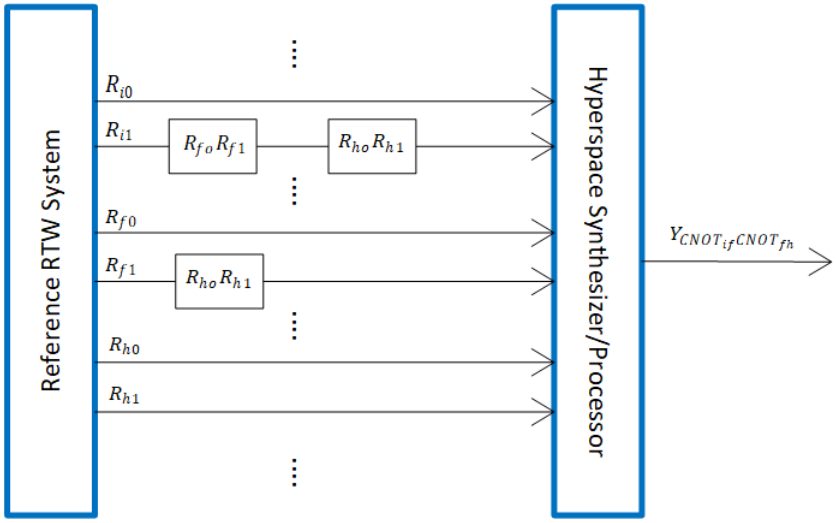

Figure 1.10: Cascaded CNOTs: applying $CNOT_{if}$ then $CNOT_{fh}$

2. CREATING THE XOR AND THE XNOR GATES IN INBL

2.1 Problem definition

Suppose that an INBL system generates the following superpostion:

$$Y_{initial} = Y_0 R_{i0} R_{f0} R_{hx_0} + Y_1 R_{i0} R_{f1} R_{hx_1} + Y_2 R_{i1} R_{f0} R_{hx_2} + Y_3 R_{i1} R_{f1} R_{hx_3} \tag{2.1}$$

Where R_{i0} represents the zero value of the *i*th bit,R_{f0} represents the zero value of the fth bit, R_{i1} represents the one value of the *i*th bit, R_{f1} represents the one value of the fth bit. { $R_{hx_0}, R_{hx_1}, R_{hx_2}, R_{hx_3}$ } represent the *h*th bit that have arbitrary initial values of { x_0, x_1, x_2, x_3 }. { Y_0,Y_1,Y_2,Y_3 } are arbitrary superpositions that do not contain RTWs of bits *i,f* or *h*. Figure 2.1 illustrates such a system.

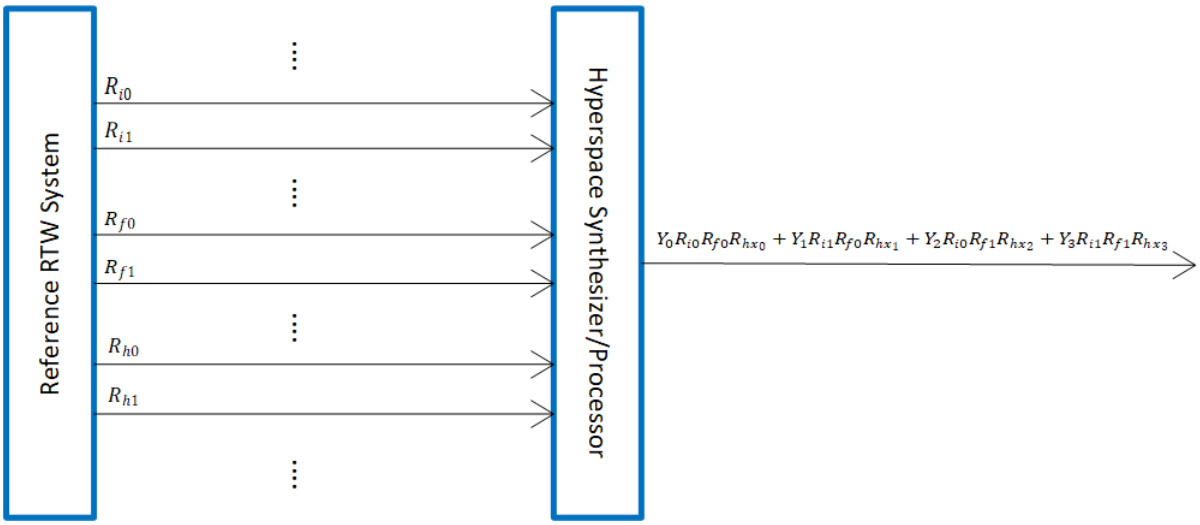

Figure 2.1: A general system before applying XOR Gate

Assume that bits { *i,f* } are the inputs to the XOR gate and the output appears on bit { *h* }. All strings in a superposition must be considered when constructing a gate in INBL; this means that all the combinations of input bits in each string should give the correct result on the output bit.

Since there are 3 bits involved in this operation, one would expect that 8 possibilities would arise in equation 2.1, but this does not happen. Since the XOR gate is only applied on inputs, the output bit value is not part of the computation and its previous value should not affect the result or the method. This causes the possibilities to collapse to only 4.

Given the previous discussion, it is concluded that all strings in a superposition need to be independent of the output bit to successfully apply the XOR gate. This means that the value of the output bit should not depend on the string in the superposition. Another challenge that arises in INBL is that there is a way to invert a bit, but there is no way to set it to a certain value with a single operation. In the following subsections, we will explain how to solve these problems together and get our desired output.

2.2 Solving the Dependency of the output bit

In order to solve the output dependency on the strings in a superposition, the strings need to be manipulated such that all output bits in any string will have the same output value. For example, all the values { x_0, x_1, x_2, x_3 } in equation 2.1 need to be the same value. Zero is chosen arbitrarily, but one would work as well. This can be done by multiplying the reference wire of R_{h1} by $R_{h0}R_{h1}$. If a string contains a R_{h1}, it will be inverted to a R_{h0} and if it contains a R_{h0}, it will stay the same, thus, all the strings will contain R_{h0}. This operation transforms the superposition in equation 2.1 to the following:

$$Y_{Independent} = Y_0R_{i0}R_{f0}R_{h0} + Y_1R_{i0}R_{f1}R_{h0} + Y_2R_{i1}R_{f0}R_{h0} + Y_3R_{i1}R_{f1}R_{h0} \qquad (2.2)$$

Figure 2.2 illustrates this operation.

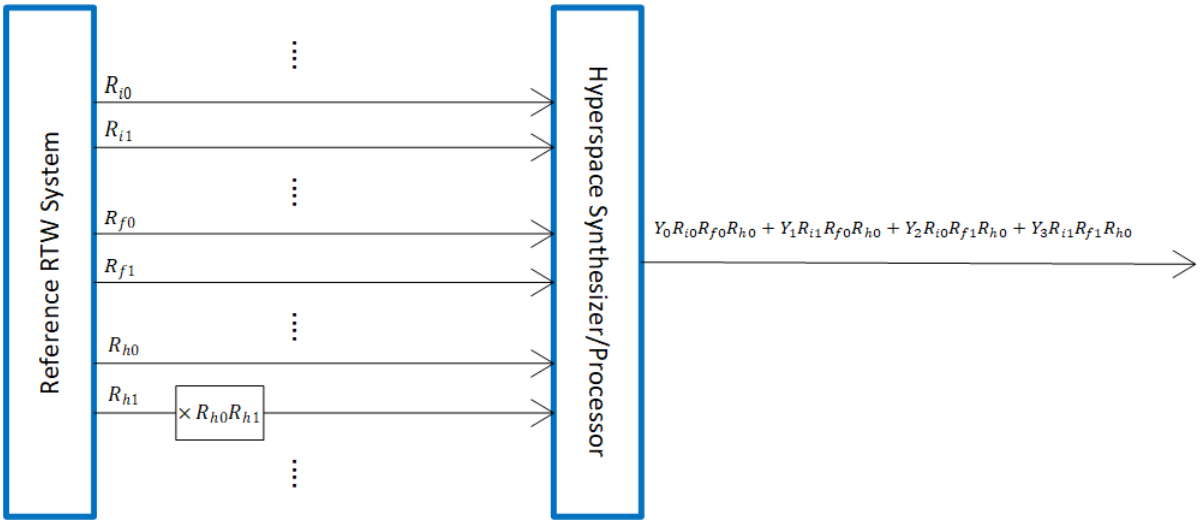

Figure 2.2: Producing an independent output bit in a superposition

2.3 Applying the XOR gate

An XOR operation yields one when the inputs are the same and yields a zero otherwise. In equation 2.2, the output bit starts at zero, this means that applying the XOR gate is equivalent to inverting the output bit when the inputs are different. As discussed previously, inverting R_{h0} means multiplying it by $R_{h0}R_{h1}$ and inverting a bit twice yields the original value. In the previous statement lies the key to the XOR gate, if the high value of the inputs (R_{i1} and R_{f1}) are multiplied by the inverting operator ($R_{h0}R_{h1}$), the following cases occur:

- Bit *i* is high and bit *f* is high: the output is inverted twice and it retains its zero value.
- Bit *i* is low and bit *f* is high: the output is inverted to one.
- Bit *i* is high and bit *f* is low: the output is inverted to one.
- Bit *i* is low and bit *f* is low: no operation happens because the low reference wires of the inputs have not been multiplied by anything. The output bit retains its zero value.

Equation 2.2 becomes:

$$\begin{aligned} Y_{XOR} &= Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} (R_{h0} R_{h1}) R_{h0} \\ &+ Y_2 R_{i1} (R_{h0} R_{h1}) R_{f0} R_{h0} + Y_3 R_{i1} (R_{h0} R_{h1}) R_{f1} (R_{h0} R_{h1}) R_{h0} \\ &= Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} NOT(R_{h0}) \\ &+ Y_2 R_{i1} R_{f0} NOT(R_{h0}) + Y_3 R_{i1} R_{f1} NOT(NOT(R_{h0})) \\ &= Y_0 R_{i0} R_{f0} R_{h0} + Y_1 R_{i0} R_{f1} R_{h1} + Y_2 R_{i1} R_{f0} R_{h1} + Y_3 R_{i1} R_{f1} R_{h0} \end{aligned} \tag{2.3}$$

The XOR gate is illustrated in Figure 2.3. It is worth noting that there were multiple ways to apply the XOR, but they all have the same principle. For example, the low value reference wires R_{i0} and R_{f0} could have been multiplied by $R_{h0}R_{h1}$ to get the XOR gate.

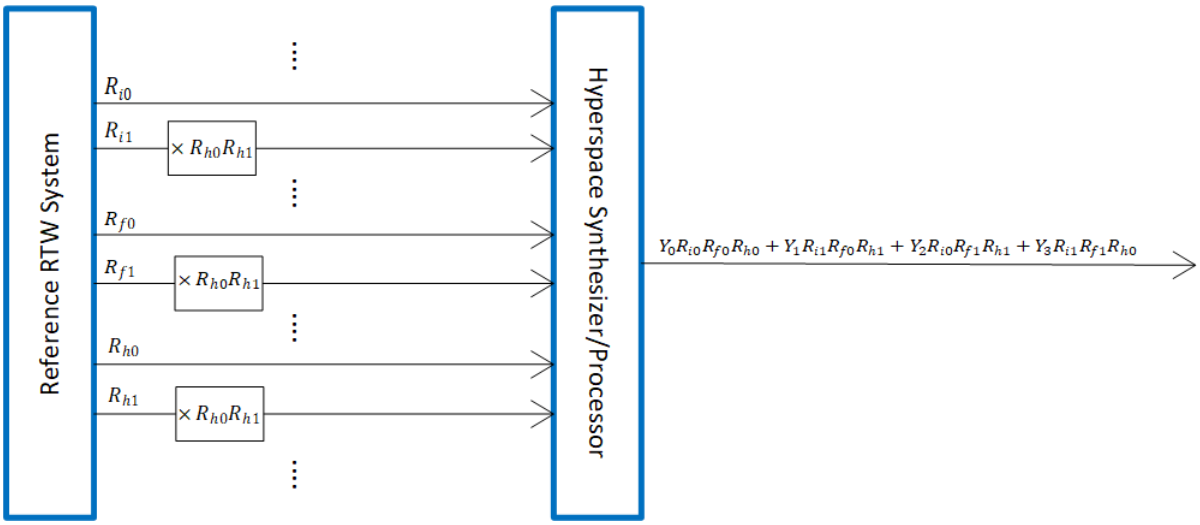

Figure 2.3: The XOR gate

2.4 Applying the XNOR gate

Since we implemented the XOR gate, we can simply use the result from the XOR gate and the NOT gate to get the XNOR gate. Figure 2.4 illustrates such an idea. If the NOT gate is applied to bit *h* in equation 2.3, then we get the XNOR gate:

$$Y_{XNOR} = Y_0 R_{i0} R_{f0} R_{h1} + Y_1 R_{i0} R_{f1} R_{h0} + Y_2 R_{i1} R_{f0} R_{h0} + Y_3 R_{i1} R_{f1} R_{h1} \tag{2.4}$$

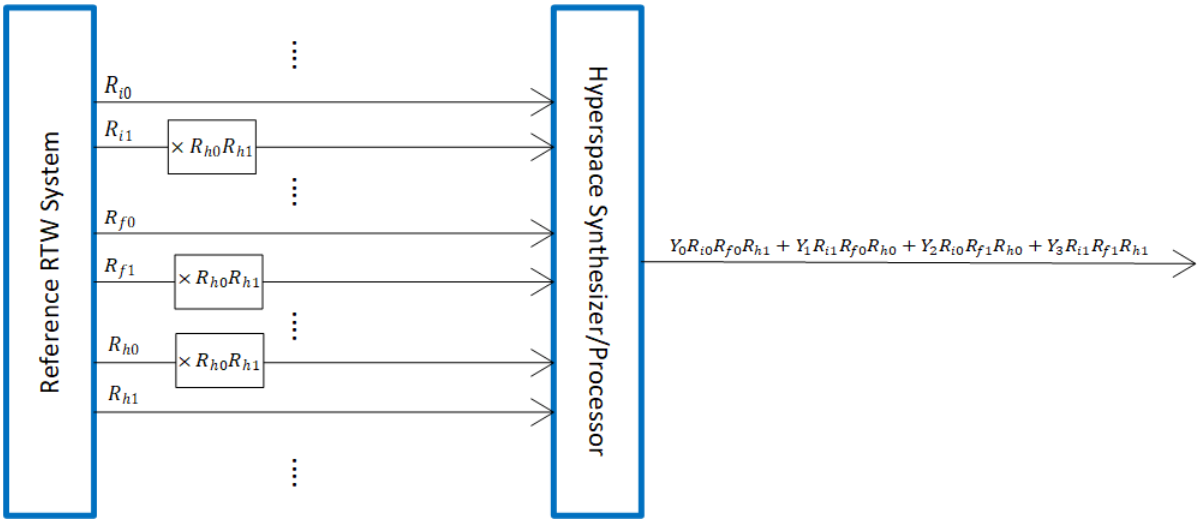

Figure 2.4: The XNOR gate using the NOT operation on the XOR gate

Alternatively, the XNOR gate can be derived using the same method as the XOR gate. Starting from equation 2.2, the high value of an input R_{i1} and the low value of the other input R_{f0} need to be multiplied by $R_{h0}R_{h1}$ to get equation 2.4. This is illustrated by Figure 2.5.

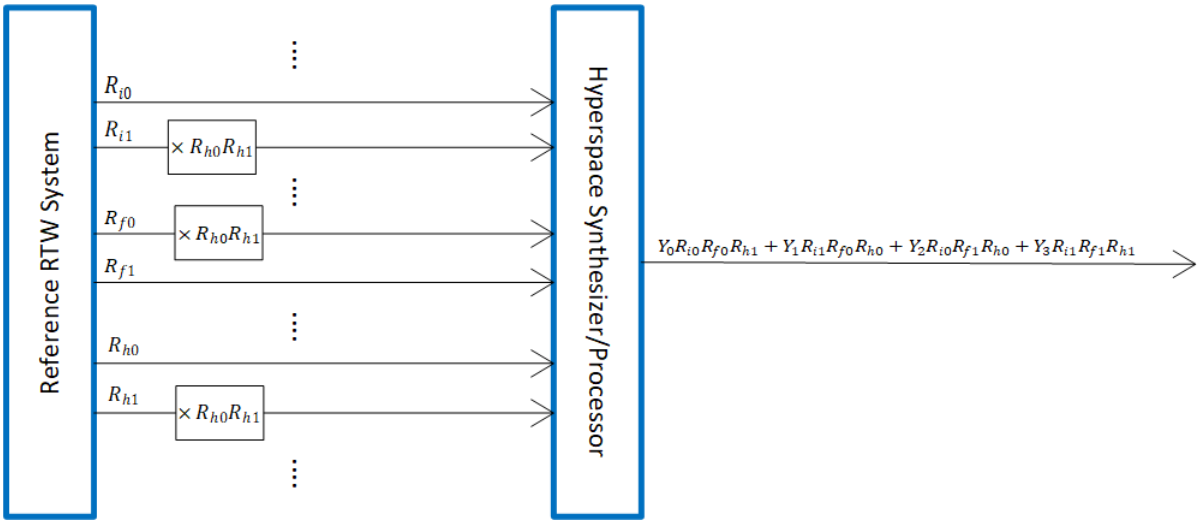

Figure 2.5: An alternate implementation of the XNOR gate

2.5 Cascading XOR gates

In this section, the cost of cascading XOR gates will be discussed. Suppose that it is needed to cascade two XOR gates as in Figure 2.6. The following superposition represents such a situation:

$$\begin{aligned} Y_{initial} =& Y_0R_{i0}R_{f0}R_{d0}R_{hx_0} + Y_1R_{i1}R_{f0}R_{d0}R_{hx_1} + Y_2R_{i0}R_{f1}R_{d0}R_{hx_2} + Y_3R_{i1}R_{f1}R_{d0}R_{hx_3} + \\ & Y_4R_{i0}R_{f0}R_{d1}R_{hx_4} + Y_5R_{i1}R_{f0}R_{d1}R_{hx_4} + Y_6R_{i0}R_{f1}R_{d1}R_{hx_6} + Y_7R_{i1}R_{f1}R_{d1}R_{hx_7} \end{aligned} \tag{2.5}$$

Where superpositions $\{ Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7\}$ are arbitrary. Bits *i*,*f* and *d* are the inputs and the output is bit *h*. Only one output bit is needed for such an operation since the gates will be applied in a single step rather than multiple steps.

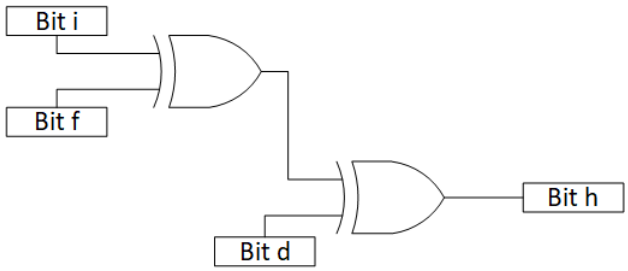

Figure 2.6: Cascaded XOR gates circuit. The inputs and outputs can be of any value. This is just an illustration to show the intended operation and does not mean that inputs or outputs have a singular value.

To apply the cascaded XOR gates, the output needs to be independent and all the high values of the input reference wires need to be multiplied by $R_{h0}R_{h1}$. Figure 2.7 illustrates this idea. The following superposition will be generated:

$$\begin{aligned} Y_{CascadedXOR} =& Y_0 R_{i0} R_{f0} R_{d0} R_{h0} + Y_1 R_{i1} R_{f0} R_{d0} R_{h1} + Y_2 R_{i0} R_{f1} R_{d0} R_{h1} + Y_3 R_{i1} R_{f1} R_{d0} R_{h0} + \\ & Y_4 R_{i0} R_{f0} R_{d1} R_{h1} + Y_5 R_{i1} R_{f0} R_{d1} R_{h0} + Y_6 R_{i0} R_{f1} R_{d1} R_{h0} + Y_7 R_{i1} R_{f1} R_{d1} R_{h1} \end{aligned} \tag{2.6}$$

Cascading two XOR gates only requires an extra multiplication operation and requires no extra bits. This can be extended to any XOR operations that are acting on *N* number of bits. After the first XOR gate, any extra XOR gate will require only one multiplication operation.

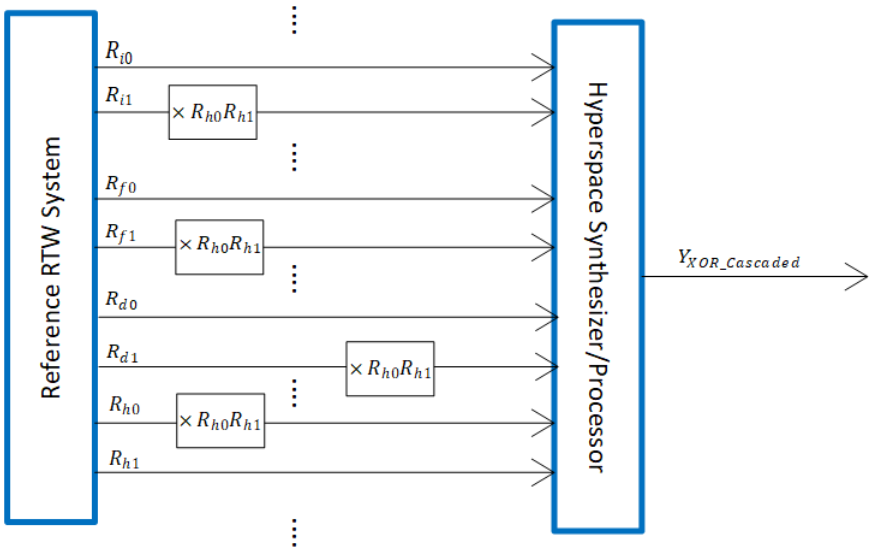

Figure 2.7: Implementation of cascaded XOR gates

2.6 Cascading an XOR gate with a CNOT gate

Assume that the circuit in Figure 2.8 needs to be implemented in INBL. The initial superposition will be the same as the one in equation 2.5.

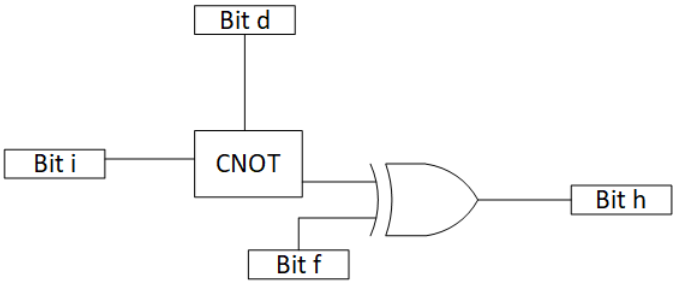

Figure 2.8: Cascading CNOT gate with an XOR gate

First, the CNOT gate will be applied, where *d* is the control bit and *i* is the target bit. The following superposition will be generated after the CNOT gate:

$$Y_{CNOT_{di}} = Y_0 R_{i0} R_{f0} R_{d0} R_{hx_0} + Y_1 R_{i1} R_{f0} R_{d0} R_{hx_1} + Y_2 R_{i0} R_{f1} R_{d0} R_{hx_2} + Y_3 R_{i1} R_{f1} R_{d0} R_{hx_3} + Y_4 R_{i1} R_{f0} R_{d1} R_{hx_4} + Y_5 R_{i0} R_{f0} R_{d1} R_{hx_4} + Y_6 R_{i1} R_{f1} R_{d1} R_{hx_6} + Y_7 R_{i0} R_{f1} R_{d1} R_{hx_7} \tag{2.7}$$

Figure 2.9 illustrates this operation.

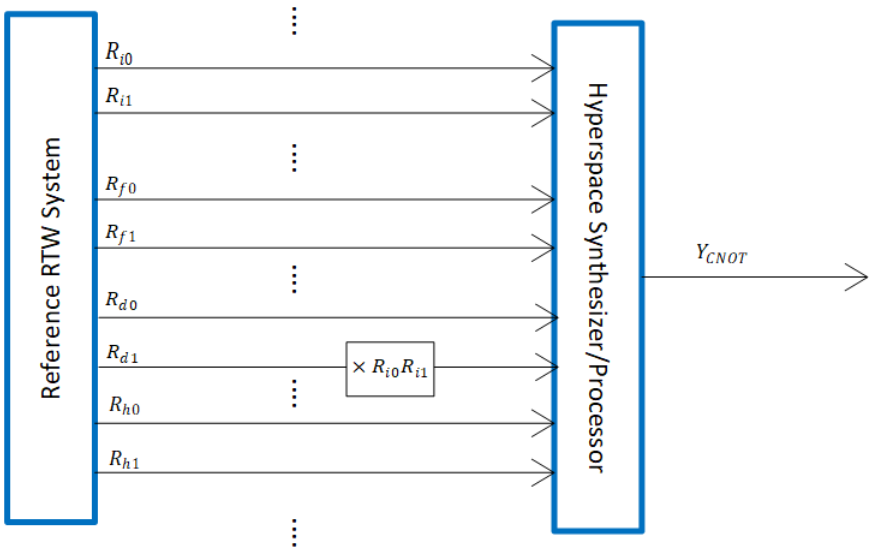

Figure 2.9: Applying $CNOT_{di}$

The tricky part is applying the XOR gate. Two cases arise here:

- Bit *d* is zero: XOR gate is applied normally as bit *i* did not change.
- Bit *d* is one: *i* gets inverted and the XOR gate will be inverted .

These two cases can be realized by multiplying R_{d1} by $R_{h0}R_{h1}$, then the regular XOR gate will be applied. Figure 2.10 demonstrates this.

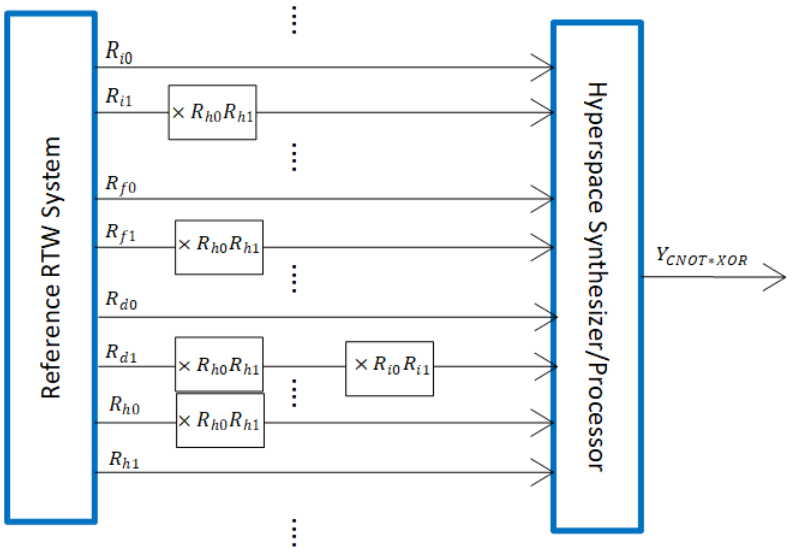

Figure 2.10: Final implementation of cascaded CNOT and XOR gates

The resulting superposition will be:

$$Y_{CascadedXOR} = Y_0 R_{i0} R_{f0} R_{d0} R_{h0} + Y_1 R_{i1} R_{f0} R_{d0} R_{h1} + Y_2 R_{i0} R_{f1} R_{d0} R_{h1} + Y_3 R_{i1} R_{f1} R_{d0} R_{h0} + Y_4 R_{i1} R_{f0} R_{d1} R_{h1} + Y_5 R_{i0} R_{f0} R_{d1} R_{h0} + Y_6 R_{i1} R_{f1} R_{d1} R_{h0} + Y_7 R_{i0} R_{f1} R_{d1} R_{h1} \tag{2.8}$$

3. CONCLUSIONS AND FUTURE WORK

3.1 Conclusions

The XOR and XNOR gates [11] have been successfully implemented in INBL. It was proven that only 4 multiplications were needed in the implementation:

- A single multiplication for the inverting operator.
- Three multiplications for the input and output bits.

Since the gates in INBL act on all the strings in a superposition, the yield of a single XOR operation will be exponentially large. It was also shown that the XOR operation is commutative as the XOR operation was repeatable with low cost. It can also be integrated efficiently with previous gates. These gates have potential applications in challenging the supremacy of quantum computing schemes, as they perform the same functions with simpler implementation and higher accuracy.

3.2 Future work

Our future work will focus on implementing other gates in INBL, so that we can eventually implement quantum algorithms. The commutativity of the cascaded CNOT gate with the XOR gate shall be explored later. We are also investigating new ways to implement a search algorithm that is both efficient and repeatable. We are also exploring new NBL schemes that might have better potential than the current system. Finally, we are investigating schemes that can be applied directly on the superposition and do not act on the reference wires.

REFERENCES

[1] L. B. Kish, "Noise-based logic: Binary, multi-valued, or fuzzy, with optional superposition of logic states," *Physics Letters A*, vol. 373, no. 10, pp. 911–918, 2009.

[2] L. B. Kish, S. Khatri, and S. Sethuraman, "Noise-based logic hyperspace with the superposition of 2n states in a single wire," *Physics Letters A*, vol. 373, no. 22, pp. 1928–1934, 2009.

[3] L. B. Kish, S. Khatri, and F. Peper, "Instantaneous noise-based logic," *Fluctuation and Noise Letters*, vol. 09, no. 04, pp. 323–330, 2010. doi: 10.1142/S0219477510000253.

[4] F. Peper and L. B. Kish, "Instantaneous, non-squeezed, noise-based logic," *Fluctuation and Noise Letters*, vol. 10, no. 02, pp. 231–237, 2011. doi: 10.1142/S0219477511000521.

[5] H. E. Wen, L. B. Kish, A. Klappenecker, and F. Peper, "New noise-based logic representations to avoid some problems with time complexity," *Fluctuation and Noise Letters*, vol. 11, no. 02, p. 1250003, 2012. doi: 10.1142/S0219477512500034.

[6] H. E. Wen, L. B. Kish, and A. Klappenecker, "Complex noise-bits and large-scale instantaneous parallel operations with low complexity," *Fluctuation and Noise Letters*, vol. 12, no. 01, p. 1350002, 2013. doi: 10.1142/S0219477513500028.

[7] L. B. Kish, C. G. Granqvist, T. Horvath, A. Klappenecker, H. Wen, and S. M. Bezrukov, "Bird's-eye view on noise-based logic," *International Journal of Modern Physics: Conference Series*, vol. 33, p. 1460363, 2014. doi: 10.1142/S2010194514603639.

[8] B. Zhang, L. B. Kish, and C.-G. Granqvist, "Drawing from hats by noise-based logic," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 32, no. 3, pp. 244–251, 2017. doi: 10.1080/17445760.2016.1140168.

[9] L. B. Kish, "quantum supremacy revisited: low-complexity, deterministic solutions of the original deutschjozsa problem in classical physical systems," *Royal Society Open Science*,

vol. 10, no. 3, p. 221327, 2023. doi: 10.1098/rsos.221327.

[10] L. B. Kish and W. C. Daugherity, “Noise-based logic gates by operations on the reference system,” *Fluctuation and Noise Letters*, vol. 17, no. 04, p. 1850033, 2018.

[11] M. B. Khreishah, W. C. Daugherity, and L. B. Kish, “XOR and XNOR gates in instantaneous noise based logic,” 2023.