1.INTRODUCTION

In recent years, reversible computing has emerged as one of the most efficient and well-known low-power circuit design approaches.

The Reversible Logic has received significant attention in recent years due to its ability to reduce power dissipation. Reversible logic gates are also finding deep and promising applications in growing growth industries such as DSP, nanotechnology, and so on, because of their one-to-one correspondence of inputs to outputs.

Reversible logic, also known as x-input and y-output logic, is defined as logic in which inputs and outputs are equal in number.

i.e., the p-input, q-output function F (X1, X2, X3…, Xn), also referred to as the (p, q) function, is reversible if and only if the following conditions are met:

(i) p equals q and,

(ii) Each pattern in the input corresponds to a different output pattern.

In addition, the gate must be able to compute in both forward and backward directions. In other words, the outputs can be used to retrieve inputs. The second Law of Thermodynamics ensures that the device does not dissipate any heat when these two conditions are met.

Irreversible logic gates are used as design units in traditional digital circuit design approaches. These gates, however, result in information loss after each operation owing to bit loss. The most prevalent source of heat generation in digital systems is bit loss from digital circuits.

1.1 CONCEPT OF REVERSIBILITY

The reversible logic gate is an n-input, n-output device with One-to-One Mapping, which allows the outputs to be determined from the inputs and vice versa. In addition, extra outputs can be added to make the output count equal to the input.

A reversible logic gate is a one-to-one logic device with n inputs and n outputs. Reversible circuits have a one-to-one mapping between vectors of inputs and outputs; therefore, the input state vector may always be reconstructed from the output state vector. The number of reversible gates in a circuit should be kept to a minimum. In reversible logic circuits, fan-out and loops are not permitted. On the other hand, additional gates can achieve fan-out and feedback.

Reversibility is a process in which direction can be reversed during computation. Data can be retrieved from backward computation when there is any data loss. Logic reversibility is the ability to retrieve an earlier calculation stage
Is made by computing the results in either backward or forward direction. The recovery of a lost bit during a calculation is a distinct aspect of reversible logic computing. The recovery of the lost bit requires unique input-output mapping approaches, as the traditional
logic computation has failed to recover the lost bit. If we used physical reversibility before logical reversibility, the benefit of logical reversibility can be increased. Physical reversibility refers to a process in which no energy dissipation results in heat. Unfortunately, in the computational system, physical reversibility is not possible.

When voltage levels shift from High to Low in computing systems, energy dissipation occurs in the form of heat. This voltage level shifting method uses substantially less energy, and the majority of the energy is consumed as heat. The reversible circuit element would transport The charge movement from one node to the next node in a gradual manner. As a result, each transfer loses a little quantity of energy. The principle of reversibility has a significant impact on today's digital system. The concept of reversibility recovers the state of the inputs from the outputs which has implications for assembly programming languages and instruction sets. It's because, in order to be as efficient as possible, these must also be reversible.

In synthesizing reversible circuits, direct fan-Out is not allowed since one–to–many concepts are irreversible. In reversible circuits, however, fan-out is accomplished by adding more gates. Therefore, the smallest number of reversible gates should be used in a reversible circuit.

Reversible logic has the potential to be a viable computing paradigm that eliminates the requirement for cooling in machines. Reversible computing arose from applying quantum physics concepts to the event of a universal computer. The principles of reversible computing have validated the link between entropy, heat transfer between molecules in a system, the possibility of a quantum particle keeping a fixed state at any given time, and the quantum field theory between electrons once they are insufficiently closely placed.

Reversible computing is based on the premise that a bijective device with an equal range of input-output lines may provide a computing environment in which electrodynamics can predict all future circumstances using all previous conditions. A reversible computer circuit is a one-to-one mapping between the input and output of an associate degree N-input N-output logic device. It allows us to retrieve the outputs from the inputs. It does, however, allow us to unambiguously reconstruct the inputs from the outputs.

The idealogy of reversibility and how it differs from irreversibility has been shown in Figures 1.1&Figure 1.2. It can be noted that if we add something to the irreversible, the design can be transformed into a reversible one.

FIGURE 1.1: IRREVERSIBLE LOGIC

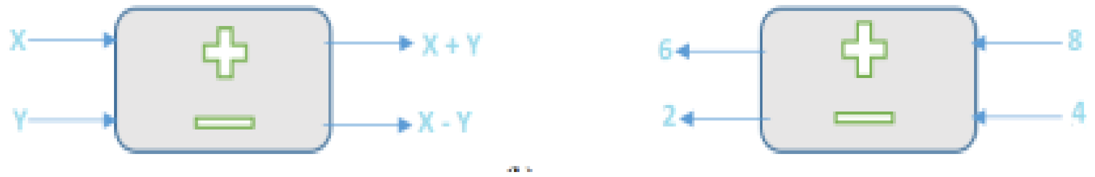


FIGURE 1.2: REVERSIBLE LOGIC

1.2 TERMINOLOGY

The terminology pertaining to the Reversible Logic Gates is explained in the terms given below:

1.2.1 Reversible Logic: It's a k-input, k-output logic function with one-to-one mapping between inputs and outputs. The output vector may be uniquely derived from the input vector using this One-to-One Mapping Technique.

1.2.2 Quantum Cost: The quantum cost is the cost of the circuit expressed in terms of primitive gates. Quantum cost is computed by having knowledge about the number of primitive Reversible Logic Gates required for realizing the Arithmetic and Logic Circuits.The quantum cost is the number of 2x2 unitary gates necessary to represent a circuit while maintaining the output unchanged. Any 1x1 gate, as well as any 2x2 gate, has a QC of one.

1.2.3 Delay: The Delay of a Reversible Logic Gate is the maximum number of gates along a circuit from any input line to its corresponding output line. The following two assumptions underpin the notion of delay:

i) Each gate completes the calculation in a single second.

ii) Before the computation begins, all of the circuit's inputs are available.

1.2.4 Hardware Complexity: Hardware complexity refers to the total number of logic processes in a circuit. It refers to the total number of AND, OR, and Ex-OR operations performed by a circuit.

1.2.5 Ancillary Inputs or Constant Inputs: The Ancillary Inputs are the Reversible Logic Gates' inputs that must remain constant at "0" or "1".

1.2.6 Garbage Outputs: Garbage outputs are unutilized outputs in reversible logic circuits that retain reversibility but do not conduct any practical operations. Garbage outputs, for example, are the number of outputs added to make an p-input q-output function reversible. Constant inputs are current value inputs that are used to make a (p:q) function reversible.

input + constant input = output + garbage outputs.

The following aspects must be considered in the design of reversible logic circuits in order to generate an optimum circuit. They are as follows:

- It is not authorised to fan out.
- Garbage outputs must be kept to a bare minimum.
- Less delay.
- Quantum costs as low as possible.
- Constant inputs should be avoided.

The number of gates, garbage outputs, constant inputs, and quantum cost must be less. Therefore, these are the key obstacles in developing reversible circuits.

Garbage outputs are the output lines that are never utilized again. Reducing these waste outputs is one of the most challenging undertakings. Only the reversible functions are realised by any reversible logic gate. Many Boolean functions, however, are not reversible. We must first convert irreversible functions into reversible functions before implementing these Boolean functions. On the circuit's input side, every transformation technique that turns an irreversible function to a reversible one introduces input lines set to zero. Constant inputs are what they're called. As a result, any effective reversible logic design should reduce the number of garbages and constant inputs.

REVERSIBLE GATES	**SIZE**	**QUANTUM COST**
NOT	**1x1**	**1**
FEYNMAN	**2x2**	**1**
FREDKIN	**3x3**	**5**
TOFFOLI	**3x3**	**5**
PERES	**3x3**	**4**
DOUBLE FEYNMAN	**3x3**	**2**
NG	**3x3**	**11**

TABLE 1.1: LIST OF REVERSIBLE GATES SIZE AND QUANTUM COST

1.3 MOTIVE BEHIND REVERSIBLE LOGIC GATES

Reversible Logic Gates are now finding tremendous and enticing applications in emerging growing paradigms such as Quantum Computing, Quantum Dot Cellular Automata, Optical Computing, Digital Electronics, Low Power CMOS Design, Nanoelectronics, and so on, due to its exceptional technique of One-to-One Mapping between the inputs and the corresponding results. As an outcome of its ability to lower power dissipation, a key concern in digital design, the Reversible Logic has attracted considerable attention in recent years.

In the course of their operation, standard combinational logic circuits give off heat in proportion to the amount of information that is lost. Because of this, it is completely impossible to retrieve a piece of information after it has been lost. However, if the same circuit is built using reversible logic gates, it is feasible to recover, and the amount of heat that is lost during the construction process is decreased.

In the 1960s, R. Landauer demonstrated that even high-tech systems with irreversible hardware are susceptible to significant energy dissipation and efficiency loss. He made his discovery. He demonstrated that the amount of energy that is lost during the exchange of one bit of information is equal to K.ln2 Joules,
where K is the Boltzmann Constant and T the Absolute Temperature at which the oper ation is performed. Later in 1973, Bennett demonstrated that this quantity of energy loss could be prevented by designing the circuit using the Reversible Logic approach. This was done to reduce the amount of energy that was lost.

According to Moore's Law, the number of components on a chip would double every 18 months; thus, irreversible technologies will produce a great deal of heat and limit the life of circuits. At this time, Reversible Logic kicks in, not only recovering the data that was lost but also reducing the amount of heat that is produced.

Because high-performing CPUs generate enormous quantities of waste heat, there is a realistic upper limit to how much further the system's performance can be improved. Soon, reversible circuits that save information by world-organizing computing bits rather than wasting them might be the only physically possible means of maintaining increased performance. This would be accomplished through world-organizing computing bits.

An increase in energy efficiency is going to be brought about through reversible computing. The amount of energy available may have an effect on the speed of electronic circuits such as nano circuits as well as the speed of the majority of computer applications. In addition, reversible computing is required in order to increase the portability of electronic gadgets. Circuit component sizes will be able to be reduced to atomic size limitations as a result of "It can make it will," and as a result, gadgets will become more flexible. In the current age of computing, facility value and performance are more important than logic hardware value. This is despite the fact that the costs of hardware styles that will be incurred in the very near future will also be rather expensive. Because of this, the significance of having reversible computing cannot be overlooked.

When the information contained in the input vectors cannot be reconstructed from the vectors produced by the circuit, information is lost. Reversible logic eliminates the need for heating since it is possible to get the input vectors of reversible circuits in a unique manner from the output vectors of the reversible circuits that they belong to. Bennett demonstrated that the gating network must be made up of reversible gates in order for there to be a possibility of zero energy being lost.

It's possible that, in the near future, reversible circuits, which conserve information by uncomputing bits rather than throwing them away, will be the only physically practical method of increasing processing speed. The efficiency with which energy is used will also improve because to reversible computing. Energy efficiency will have an effect on the speed at which circuits such as Nano circuits, and therefore the pace at which the majority of computer applications, operate.

In order to improve the mobility of devices once again, reversible computing is required. In addition to this, it will make it possible to shrink the sizes of the individual circuit elements to atomic levels, which will result in portable electronics. In spite of the fact that the expenses of developing the hardware may soon become significant, the need for reversible computing is something that simply cannot be ignored in the modern age of computing, when power and performance are prioritised above the costs of developing the logic hardware.

Figure 1.3 is a useful tool for gaining a better understanding of the influence that reversibility and its breadth have had and will continue to have on the present computer industry as well as the VLSI industry in the future. It was obvious that the researchers had come very close to reaching their limit in terms of the amount of energy they used. The key to understanding this is to keep in mind the idea of reversibility. A further development of the design is possible if we search for new models that are more environmentally friendly.

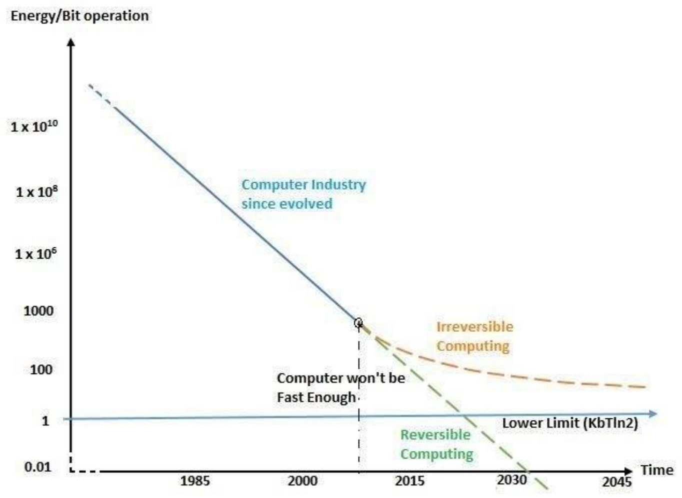


FIGURE 1.3: SCOPE OF REVERSIBLE COMPUTING

1.4 TOOLS USED:

1. Xilinx: Vivado Design Suite is a software suite developed by Xilinx to synthesise and analyse hardware description language (HDL) designs. It replaces Xilinx ISE and adds capabilities for system on chip development and high-level synthesis.

   Verilog is a programming language for describing hardware (HDL).It is a programming language used to describe digital systems such as network switches, microprocessors, memory, and flip-flops. We can describe any digital hardware at any level using an HDL. HDL-described designs are technology-independent and simple to create and debug. They are frequently more valuable than diagrams, especially when dealing with big circuits.

2. LTspice: "Simulation Program with Integrated Circuit Emphasis" (SPICE) is an abbreviation that stands for "Simulation Program with Integrated Circuit Emphasis. It's a free, open-source simulation program for modeling analog circuits in realistic situations. Devices are also simulated using SPICE.

1.4 AIM

This project aims to design and simulate reversible logic circuits.

1.5 OBJECTIVES

1. To design and simulate reversible logic circuits using Verilog.
2. To prove the concept of reversible computing in reversible logic gates.
3. To portray the advantages of reversible logic gates over irreversible logic gates.

1.6 ORGANIZATION OF THESIS

The project report presented is categorized into seven chapters. The first chapter consists of the introduction to the project. Then, we state the aim and objective of our project and briefly discuss the concept of reversibility and the motive behind using these reversible logic circuits. Following this introductory chapter is chapter 2, which is the literature survey. We collected numerous research papers for the survey. Finally, we presented them in a report describing the reversible logic circuits, their properties, and why they dissipate less power than conventional digital circuit designing methods where irreversible logic circuits are used.

Chapter 3 contains the basic reversible logic circuits designed and simulated; and circuits that are realized using these gates. Chapter 4 provides implementation, i.e., the code and testbench for above mentioned logical circuits.

Then, in chapter 5, the simulation results for the reversible logic circuits are provided. In chapter 6, we conclude our project and provide future scope. Finally, the last chapter, i.e., chapter 7, presents the references that helped us design and simulate reversible logic circuits.

2. LITERATURE SURVEY

- Bits of information are erased during logic operations in conventional circuits, resulting in the dissipation of insignificant amounts of energy. As a result, if circuits are designed to preserve information bits, power consumption can be reduced. The information bits are not lost in the case of reversible logic computation. We can use reversible logic technology to reduce power consumption, heat dissipation, and increase speed, among other things. This paper[8] describes various reversible logic gates such as Toffoli, Peres, Fredkin, Feynman, and others. A comparison of classical and quantum logic gates on various parameters is also provided, as are the limitations of conventional computing.

- The gate (or circuit) is said to be reversible if the inputs and outputs have a one-to-one correspondence, the outputs can be uniquely determined from the inputs, and the inputs can be retrieved from the outputs. Thus, a gate is reversible if and only if each input assignment generates its own output. It is commonly assumed that a reversible gate has the same number of inputs and outputs. The truth table rows of a reversible gate are obtained by permuting the input rows (circuit). For exactly half of its input assignments, each output function equals 1. (These are referred to as balanced functions.)

- Traditional approaches to combinational logic design are vastly different from those used in reversible logic design. The number of inputs and outputs on the reversible logic circuit must be equal. As a result, each output is only used once, resulting in an acyclic circuit. The most difficult task is reducing non-utilized waste outputs.

- By comparing thermodynamics and information theory, (Landauer, 1961) predict The birth of reversible technology and declared heat loss with each lost bit of i nformation. Bennett (1973) declared the birth of reversible technology by stating that there would be no heat dissipation if we avoided bit loss by mapping numb er of output lines for n number of input lines. Following that, reversible logic-based Feynman, Toffoli, and Fredkin gates replaced classical irreversible logic ba sed gates such as two input NAND, NOR, XOR, and XNOR. In irreversible log ic, NAND and NOR are considered universal logic gates, and any Boolean expre ssion in the sum of product or product of sum can be implemented using these gates. Because of their similarities, Toffoli and Fredkin gates are also considered universal.

- Digital circuits are used in almost every aspect of modern life. Low-power systems are critical in today's world. The primary outcome of this necessity is the reversible logic approach. Reversible technology is widely used in nanotechnology, low-power CMOS design, optical computing, and other fields. Using reversible design units, the reversible approach aims to redesign any digital circuit. In this paper, we propose a reversible approach to designing an N-bit adder/subtractor.

- Reversible computing has emerged as one of the most efficient and prominent techniques in low power circuit design in recent years. This paper uses therevers ible Arithmetic and Logic Unit (ALU) to demonstrate its major implications on t he Central Processing Unit (CPU). This paper proposes a reversible ALU design. The proposed design includes eight arithmetic operations and four logical operat ions The proposed design employs the Peres Full Adder Gate (PFAG).

- Reversible computing is a fascinating topic to study. According to Landauer's research, each irreversible bit operation consumes at least kT In2 joules of energy in thermodynamic entropy, where k is Boltzmann's constant and T is the ambient temperature. Bennet asserted that if the network was built up of reversible gates, there would be no energy dissipation, implying that reversible computing will become more important in circuit design in the next years.

- In recent years, reversible logic has emerged as a promising technology, with applications in low power CMOS, quantum computing, nanotechnology, and optic al computing. Reversible logic circuits use less power and assign different output s to different inputs. Traditional gates like XOR and XNOR are irreversible. The goal of this paper is to find a reversible counterpart for all irreversible basic lo gic gates and to create a complete custom layout with reversibility of all these g ates using 0.25µm technology to synthesise and stimulate them to ensure correct ness.

3. REVERSIBLE LOGIC CIRCUITS

In this project, basic reversible logic gates have been implemented and using these diffe rent combinational circuits and sequential circuits have been implemented:

3.1 Fredkin Gate:

Fig 3.1 shows a 3x3 Fredkin gate. The input vector is given as I (A, B, C) and the o utput vector is given as O (P, Q, R). The output is defined by P=A, Q=A′B+AC and R=A′C+AB.

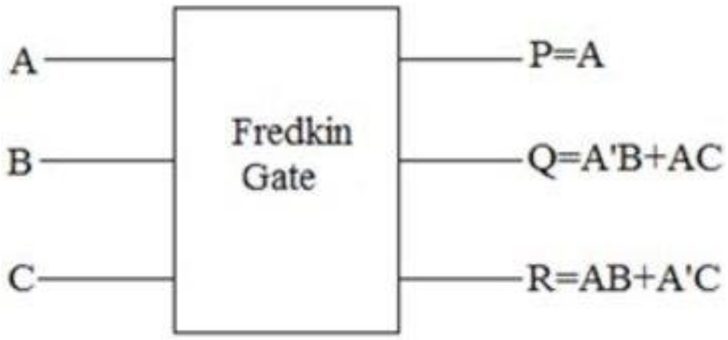


Figure 3.1: Fredkin Gate

3.2 Feynman Gate:

Fig 3.2 shows a 2x2 Feynman gate. The input vector is given as I (P, Q) and the out put vector is given as O (X, Y). The output is defined by X=P, Y=P^Q.

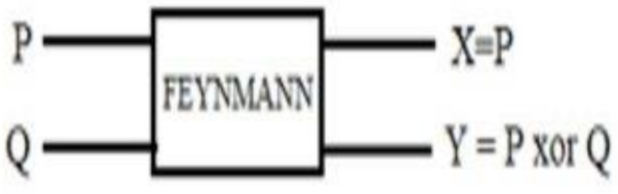


Figure 3.2 Feynman Gate

3.3 Toffoli Gate:

Fig 3.3 shows a 3x3 Toffoli gate. The input vector is given as I (A, B, C) and the ou tput vector is given as O (P, Q, R). The output is defined by P=A, Q=B and R=AB^C .

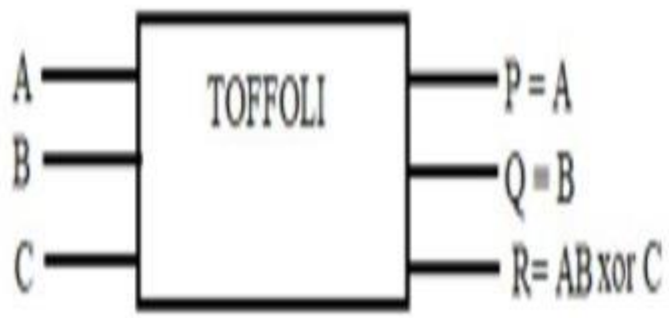


Figure 3.3: Toffoli Gate

3.4 Peres Gate:

Fig 3.4 shows a 3x3 Peres gate. The input vector is given as I (A, B, C) and the output vector is given as O (P, Q, R). The output is defined by P=A, Q=A^B and R=AB^C.

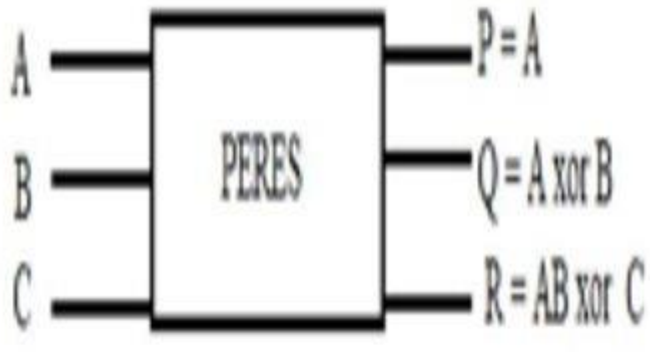


Figure 3.4: Peres Gate

3.5 HNG Gate:

Fig 3.5 shows a 4x4 HNG gate. The input vector is given as I (A, B, C, D) and the output vector is given as O (P, Q, R, S). The output is defined by P=A, Q=B, R=A^B ^C and S= (A^B). C^AB^D.

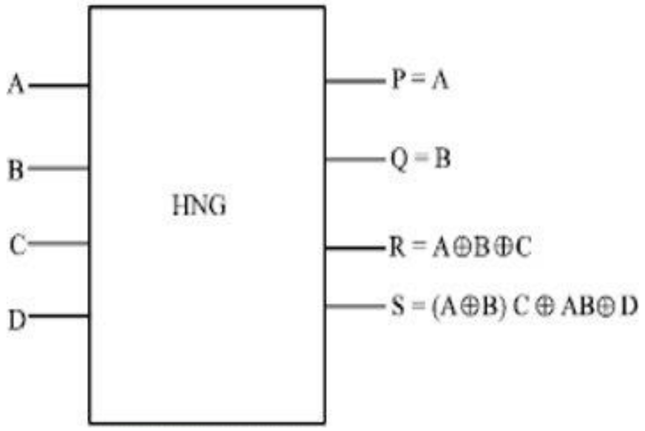


Figure 3.5: H.N.G. Gate

3.6 TSG Gate:

Fig 3.6 shows a 4x4 TSG gate. The input vector is given as I (A, B, C, D) and the o utput vector is given as O (P, Q, R, S). The output is defined by P=A, Q=A’C’^B’, R =(A’C’^B’)^D, S= (A’C’^B’)D^(AB^C).

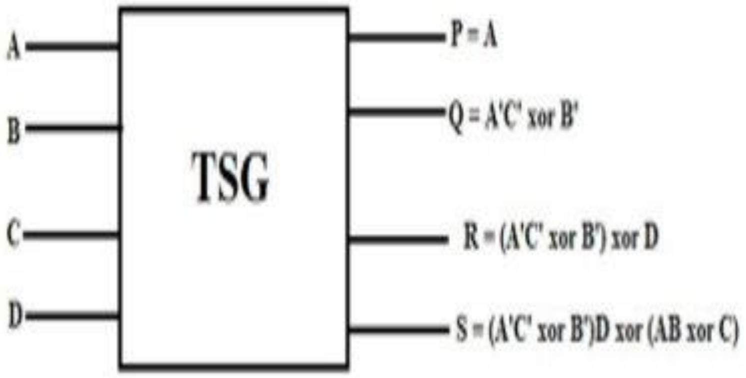


Figure 3.6: T.S.G. Gate

3.7 COG Gate:

Fig 3.7 shows a 3x3 COG gate. The input vector is given as I (A, B, C) and the outp ut vector is given as O (P, Q, R). The output is defined by P=A, Q=AC+A'B and R= BC+B'C'.

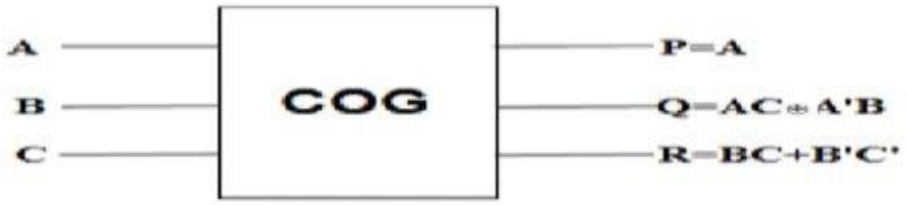


Figure 3.7: C.O.G. Gate

3.8 SRG Gate:

Fig 3.8 shows a 4x4 SRG gate. The input vector is given as I (B, A, C, D) and the o utput vector is given as O (Q, P, R, S). The output is defined by Q=A^C, P=A^B, R=A'B^A'C^BC, S=A^B^C^D.

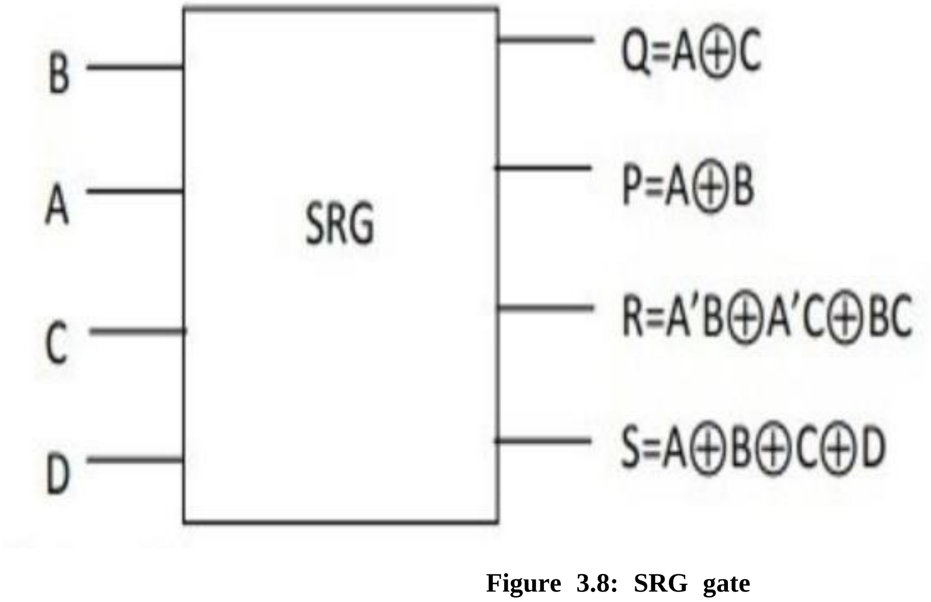


Figure 3.8: SRG gate

3.9 BVF Gate:

Fig 3.9 shows a 4x4 BVF gate. The input vector is given as I (A, B, C, D) and the o utput vector is given as O (P, Q, R, S). The output is defined by P=A, Q=A^B, R=C, S=C^D.

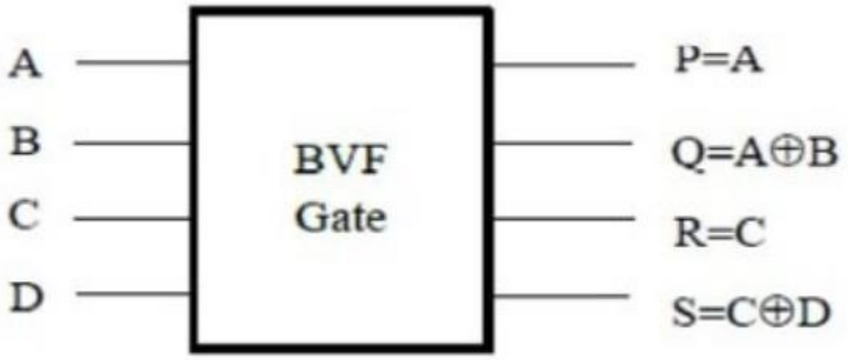


Figure 3.9: BVF Gate

3.10 BJN Gate:

Fig 3.10 shows a 3x3 BJN gate. The input vector is given as I (A, B, C, D) and the output vector is given as O (P, Q, R, S). The output is defined by P=A, Q=B, R=(A+ B)^C.

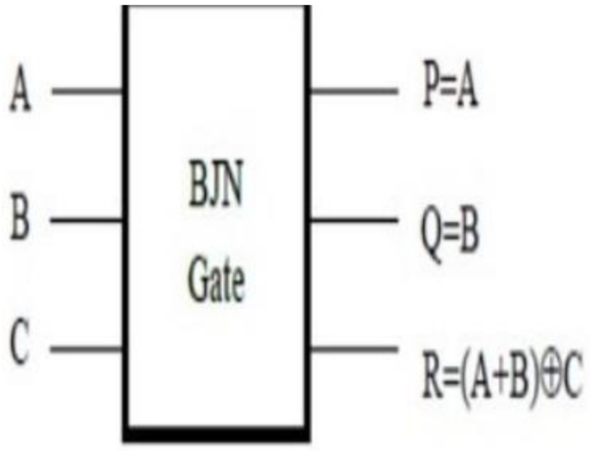


Figure 3.10: B.J.N. Gate

3.11 Double Feynman Gate:

Fig 3.11 shows a 3x3 Double Feynman gate. The input vector is given as I (A, B, C) and the output vector is given as O (P, Q, R). The output is defined by P=A, Q=A^ B, R=A^C.

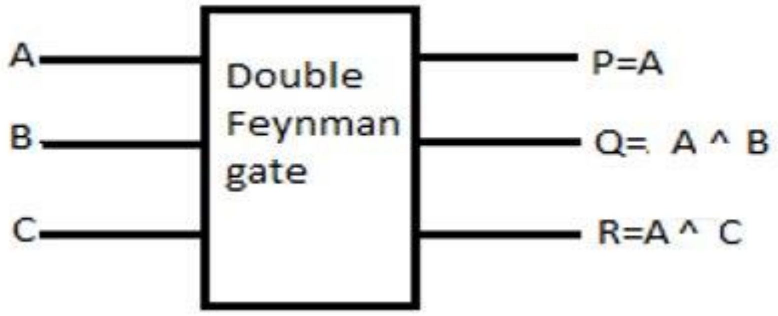


Figure 3.11: Double Feynman Gate

3.12 DINV Gate:

Fig 3.12 shows a 3x3 DINV gate. The input vector is given as I (A, B, C) and the o utput vector is given as O (P, Q, R). The output is defined by P=A'B, Q=AB, R=AC.

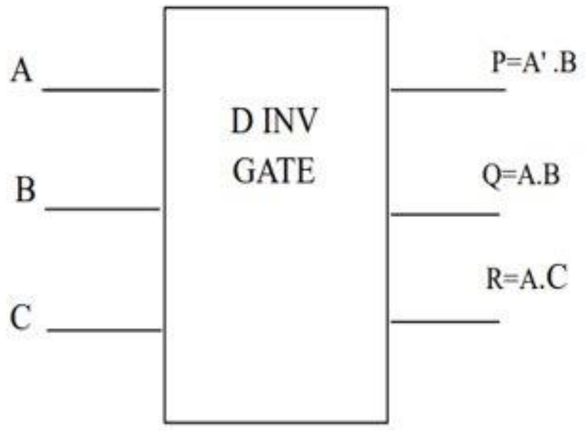


Figure 3.12: DINV Gate

3.13 AND Gate using Fredkin Gate:

Fig 3.13 shows AND gate which is obtained from Fredkin Gate. Here 3x3 Fredkin gate is used with input vector given as I (A, B, C) and output vector given as O (P, Q, R). By a ssigning constant value ‘0’ to C (C=0) we get the outputs as, P=A, Q=A’B and R=AB. O utput R is equivalent to AND Gate.

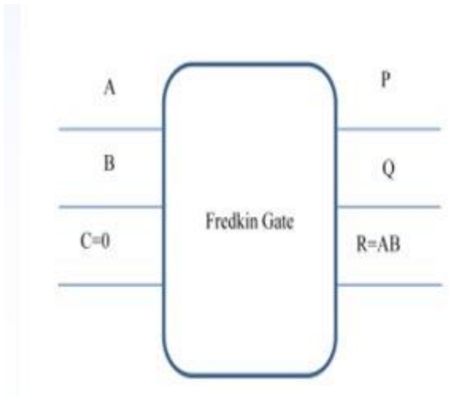


Figure 3.13: AND gate using Fredkin Gate

3.14 OR Gate using Fredkin Gate:

Fig 3.14 shows OR gate which is obtained from Fredkin Gate. Here 3x3 Fredkin gate is u sed with input vector given as I (A, B, C) and output vector given as O (P, Q, R). By as signing constant value ‘1’ to B (B=0) we get the outputs as, P=A, Q=(A’+AC) and R=A+ C. Output R is equivalent to OR Gate.

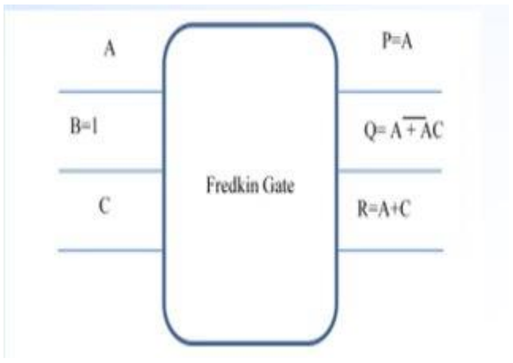


Figure 3.14: OR Gate using Fredkin Gate

3.15 Half Adder using Peres Gate:

Fig 3.15 shows a half adder which is obtained from Peres Gate. Here 3x3 Peres gate i s used with the input vector I (A, B, C) and output vector O (P, Q, R). By assigning constant value ‘0’ to C (C=0), we get the outputs as P=A, Q=A^B, R=AB. Output exp ressions of Q and R are same as the expressions of Half Adder.

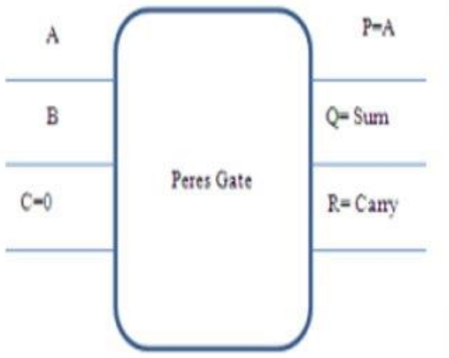


Figure 3.15: Half Adder using Peres Gate

3.16 Full Adder using HNG Gate:

Fig 3.16 shows a full adder which is obtained from HNG Gate. Here 4x4 HNG gate is us ed with the input vector I (A, B, C, D) and output vector O (P, Q, R, S). By assigning c onstant value ‘0’ to D (D=0), we get the outputs as P=A, Q=B, R= A^B^C, S=(A^B)C^A B. Output expressions of R and S are same as the sum and carry expressions of Full Add er.

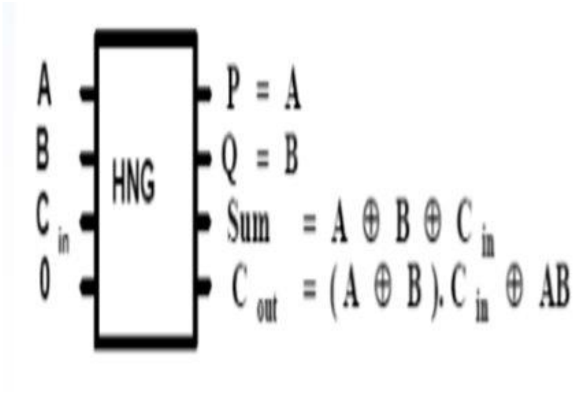

A
B
C_{in}
0
HNG
P = A
Q = B
$Sum = A \oplus B \oplus C_{in}$
$C_{out} = (A \oplus B).C_{in} \oplus AB$

Figure 3.16: Full Adder using HNG gate

3.17 Half Adder using BVF and BJN Gate:

Fig 3.17 shows half adder which is obtained by using one BVF gate and two BJN gat es. Here 4x4 BVF gate with input vector I (A, 1, B, 1) and output vector O (P, Q, R, S), BJN gate with input vector I (0, P, R) with output vector O (G1, G2, X), BJN g ate with input vector I (Q, S, 1) and output vector O (G3, G4, Y) are used. By substit uting and solving the Boolean equations we get X= A^B and Y= AB which are equiva lent to sum and carry expressions of half adder. As G1, G2, G3, G4 are not used they 're considered as garbage outputs.

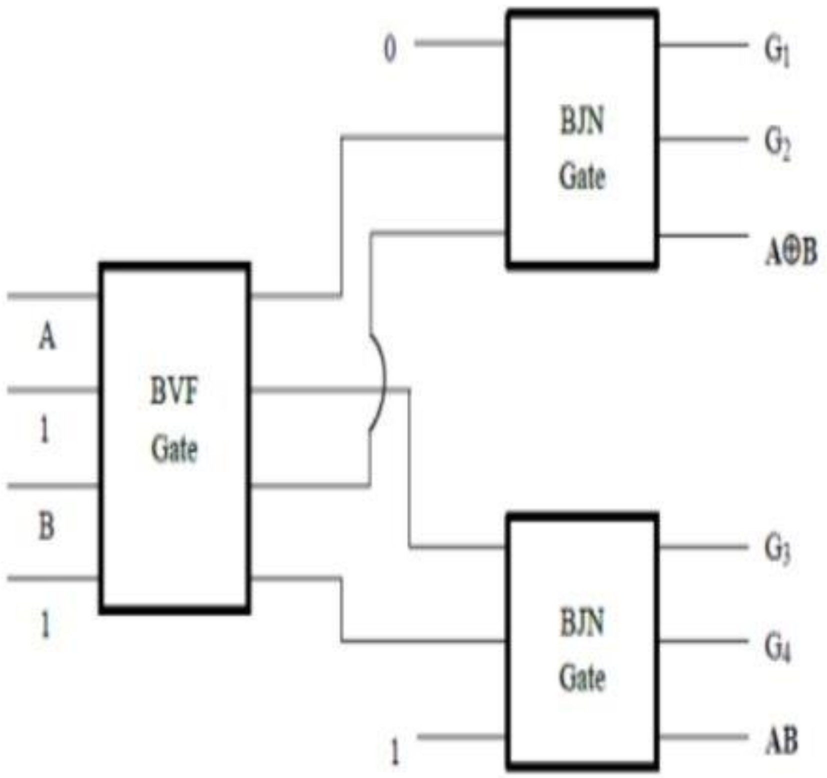


Figure 3.17: Half Adder using B.V.F. and B.J.N. gates

3.18 Half Subtractor using BVF and BJN gate:

Fig 3.18 shows half subtractor which is obtained by using one BVF gate and two BJN gates. Here 4x4 BVF gate with input vector I (A, 0, B, 1) and output vector O (P, Q , R, S), BJN gate with input vector I (0, P, R) with output vector O (G1, G2, X), BJ N gate with input vector I (Q, S, 1) and output vector O (G3, G4, Y) are used. By su bstituting and solving the Boolean equations we get X= A^B and Y= A'B which are e quivalent to difference and borrow expressions of half subtractor. As G1, G2, G3, G4 a re not used they're considered as garbage outputs.

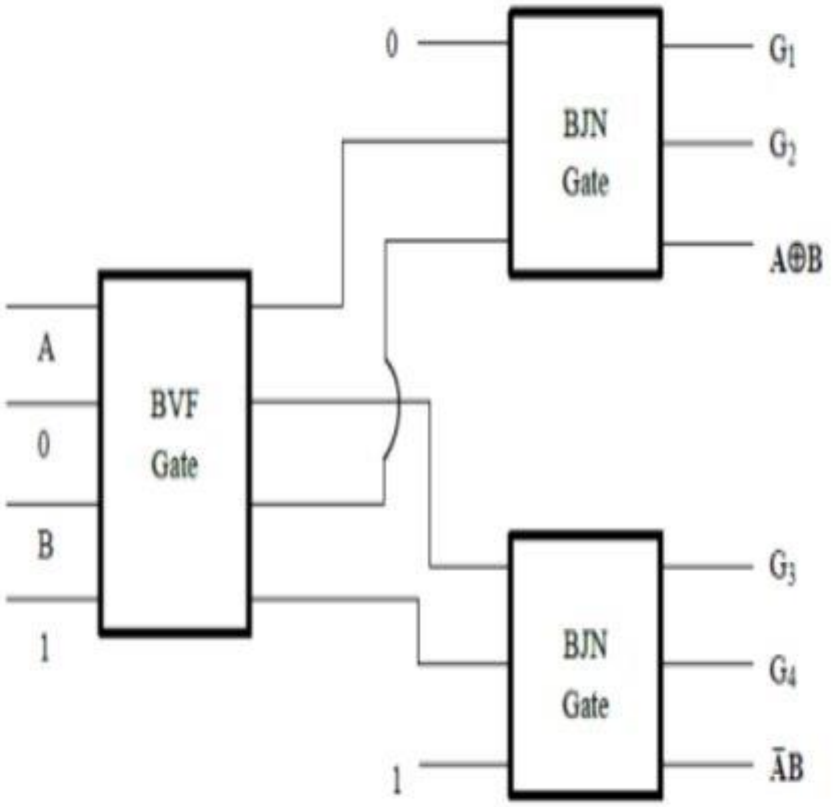


Figure 3.18: Half Subtractor using B.V.F. and B.J.N. Gates

3.19 1-Bit Comparator using BVF and BJN gates:

Fig 3.19 shows 1bit comparator which is obtained by using one BVF gate and three B JN gates. Here 4x4 BVF gate with input vector I (A, 1, B, 1) and output vector O (P, Q, R, S), BJN gate with input vector I (P, S, 1) with output vector O (G1, G2, X), BJN gate with input vector I (Q, R, 1) and output vector O (G3, G4, Y), BJN gate wi th input vector I (X, Y, 1) and output vector O (G5, G6, Z) are used. By substituting and solving the Boolean expressions, we get X= A'B and Y= AB', Z=(A^B)' which ar e equivalent to 'A less than B', 'B less A', and 'A equal to B' respectively, so it rese mbles a comparator. As G1, G2, G3, G4, G5, G6 are not used they're considered as g arbage outputs.

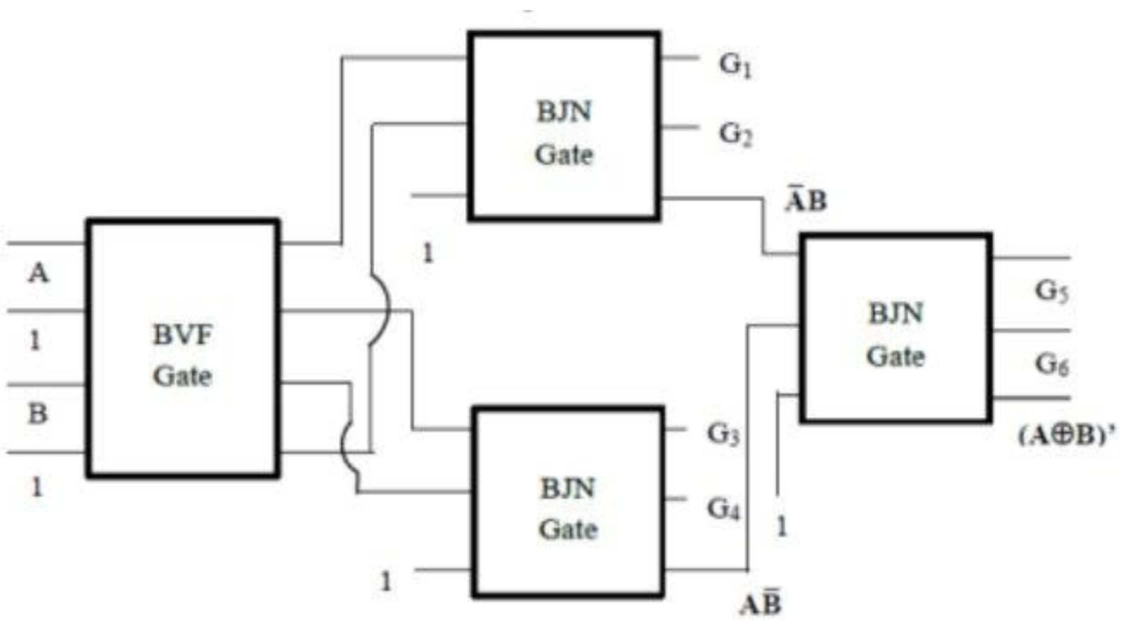


Figure 3.19: 1-Bit Comparator using B.V.F. and B.J.N. gates

3.20 2x1 MUX using Reversible Gates:

Fig 3.20 shows 2x1 MUX which is nothing but a 2x2 Fredkin gate with input vector I (S, I0, I1) and output vector O (P, Q, R). The outputs are defined as P=S, Q=S'I0+SI 1 and R=S'I1+SI0'. Boolean expression of Q is equivalent to that boolean expression of 2x1 mux output.

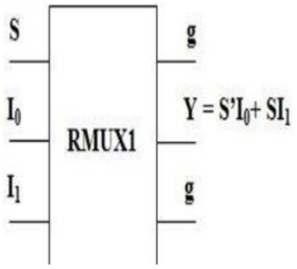


Figure 3.20: 2x1 Mux

3.21 4x1 MUX using Reversible Gates:

Fig 3.21 shows 4x1 MUX which consists of two selection lines: S0 and S1, and four i nputs I0, I1, I2, I3 and an output Y. Here, three 2x1 Mux are used in which the input vector to the first Mux is given as I (S0, I0, I1) and for second Mux the input vecto r is I (S0, I2, I3). The outputs from first Mux will be P=S0'I0+S0I1 and from second Mux it is Q=S0'I2+S0I3. The input vector to the third Mux is given as I (S1, P, Q). Based on selection line S1, corresponding output will be obtained at Y, as remaining o utputs are not of any use, they are considered as garbage outputs.

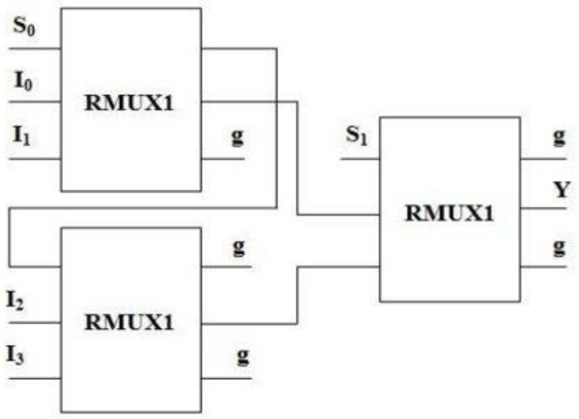


Figure 3.21: 4x1 Mux

3.22 8x1 Mux using Reversible gate

Fig 3.22 shows 8x1 MUX which consists of three selection lines: S0, S1 and S2 and e ight inputs I0, I1, I2...., I7 and an output Y. Here, seven 2x1 Mux are used in which t he input vector to the first Mux is given as I (S0, I0, I1), second Mux the input vecto r is I (S0, I2, I3), third Mux the input vector is I (S0, I4, I5), fourth Mux the input v ector is I (S0, I6, I7). The outputs from first Mux will be P=S0'I0+S0I, from second Mux it is Q=S0'I2+S0I3, from third mux it is R=S0'I4+S0I5 and from fourth Mux it i s S=S0'I6+S0I7. The input vector to the fifth Mux is given as I (S1, P, Q) and to sixt h Mux it is given as I (S1, R, S). The selection line given to last Mux is S2 and out puts from fifth and sixth mux are given as inputs to the last mux. Based on selection l ine S2, corresponding output will be obtained at Y, as remaining outputs are not of an y use, they are considered as garbage outputs.

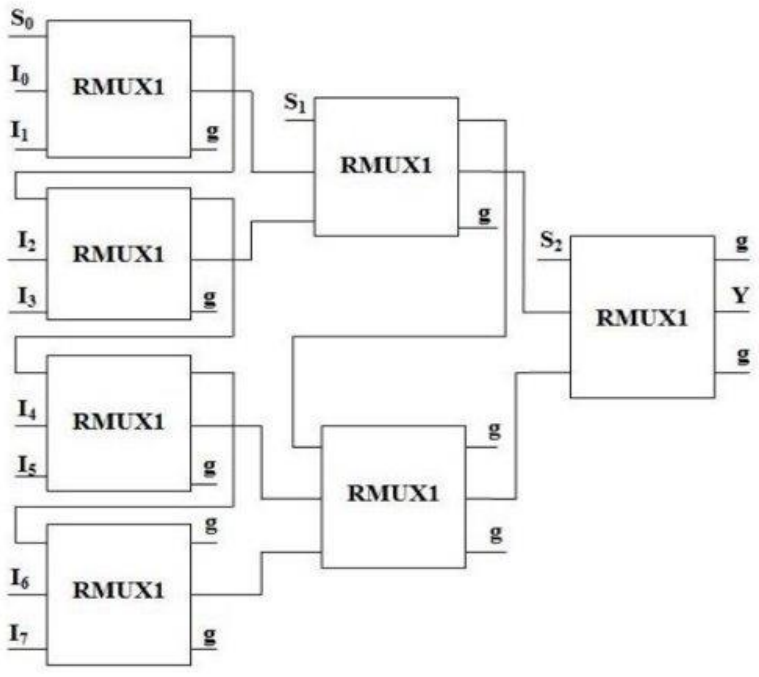


Figure 3.22: 8x1 Mux

3.23 1-Bit ALU:

Fig 3.23 shows the block diagram of 1Bit ALU which consists of two main blocks Co ntrol unit and a reversible full adder. There are three inputs A, B and Cin, four control inputs Cinput1, Cinput2, Cinput3 and Cinput4, and three selection lines S0, S1 and S2 given to Control unit. The output vector for Control unit is given as O (X, Y, Z). Th e input vector for reversible full adder is given as I (X, Y, Z) and Cinput5 i.e., a cont rol input is given to a reversible full adder. The output vector is given as O (F, cout).

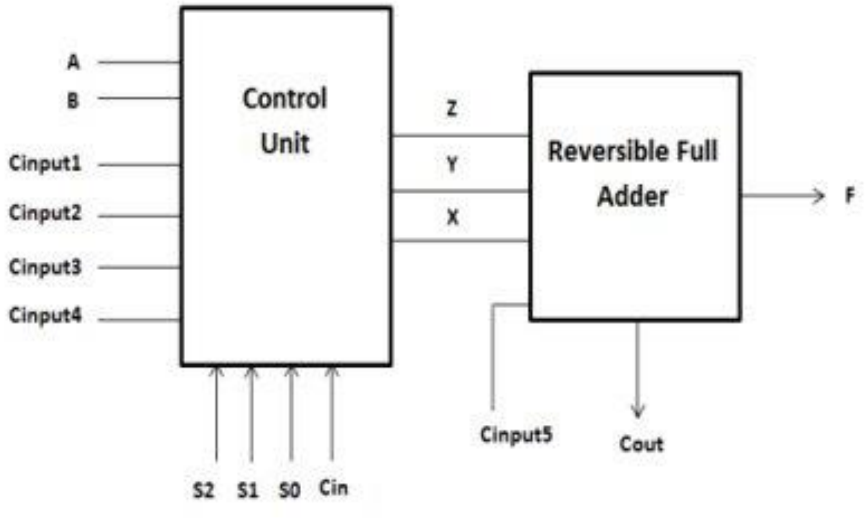


Figure 3.23: 1-Bit ALU Block Diagram

Fig 3.24 shows the block diagram of control unit. It consists of three Feynman gates, o ne Fredkin gate and three R-
I gates. Constant value of '1' is assigned to Cinput1 and Cinput2, and a constant value of '0' is assigned to Cinput3 and Cinput4. By substituting and solving the boolean ex pressions we get the values of X, Y and Z, as remaining outputs are not used, they ar e considered as garbage outputs.

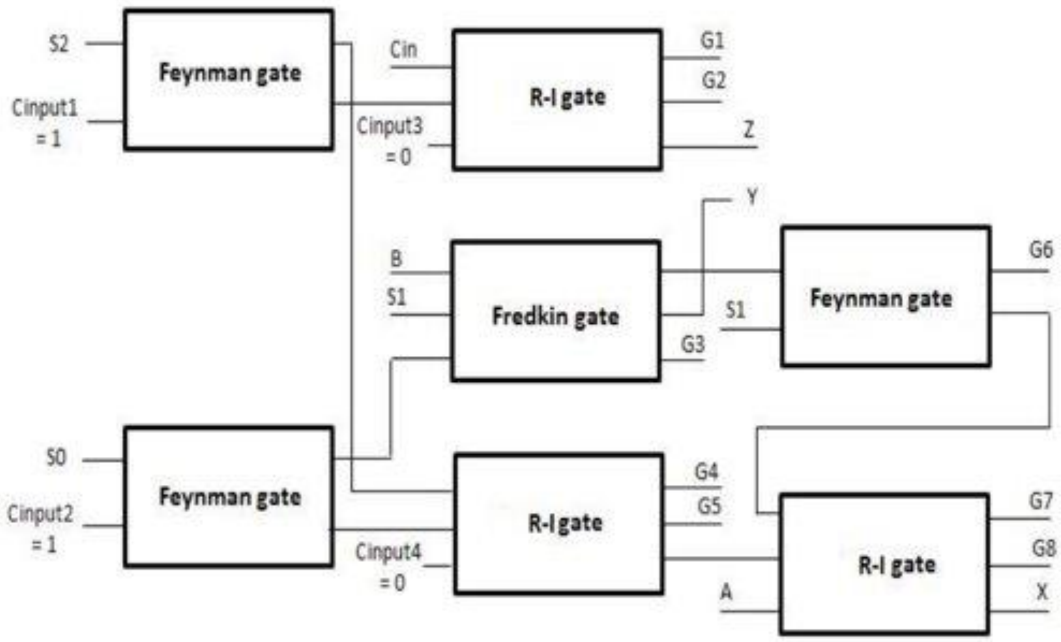


Figure 3.24: Block diagram of Control Unit

Fig 3.25 shows RI gate which has been used in Control unit. It's a 3x3 reversible gate with input vector given as I (A, B, C) and output vector given as O (P, Q, R). The outputs are defined as P=B, Q= AB'+BC, R= AB^C.

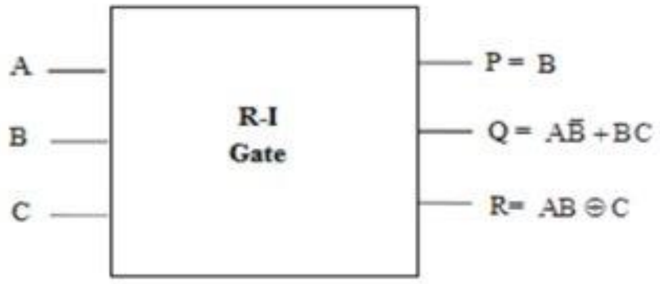


Figure 3.25: R-I Gate

3.24 SR Flip Flop

Fig 3.26 shows the block diagram of sequential circuit SR flip flop which contains two CNOT gates and four Fredkin gates. There are three inputs S, R and clk and two out puts Q and Qb. The input vector of first Fredkin gate is given as I (R, 0, clk) and se cond Fredkin gate is given as I (clk, 0, S). The output vector of first and second Fred kin gate are given as O (g1, X, g2) and O (g3, Y, g4). The input vector of third and fourth Fredkin gate are given as I (X, 1, Qb) and I (Y, 1, Q) respectively. Q and Qb are given as feedback to third and fourth Fredkin gate. The outputs of third and fourth Fredkin gate are given to CNOT with one input "1" and we get the output as Q and Qb, and the outputs which are not used are considered as garbage outputs.

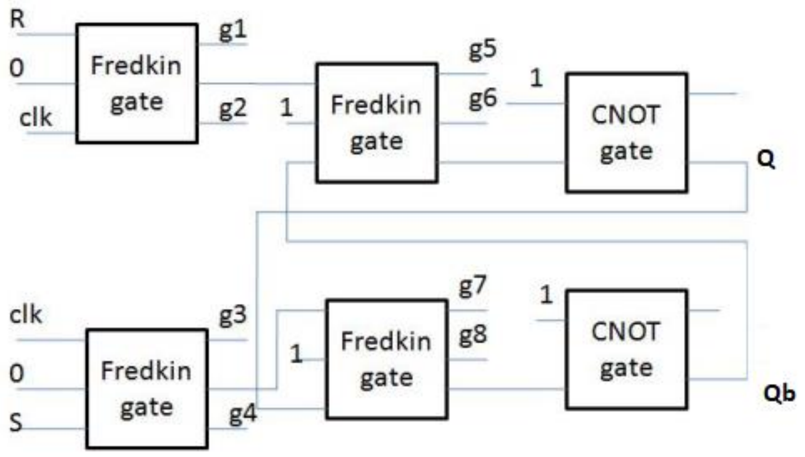


Figure 3.26: SRFF

3.24 D Flip Flop

Fig 3.27 shows the block diagram of sequential circuit D flip flop which contains three CNOT gates and four Fredkin gates. There are two inputs D and clk and two outputs Q and Qb. D is given to a CNOT gate whose one input is given as ‘1’ and output v ector for this CNOT gate is given as O (D, Db). The input vector of first Fredkin gate is given as I (D, 0, clk) and second Fredkin gate is given as I (clk, 0, Db). The outp ut vector of first and second Fredkin gate are given as O (g1, X, g2) and O (g3, Y, g 4). The input vector of third and fourth Fredkin gate are given as I (X, 1, Qb) and I (Y, 1, Q) respectively. Q and Qb are given as feedback to third and fourth Fredkin gat e. The outputs of third and fourth Fredkin gate are given to CNOT with one input as “1” and we get the output as Q and Qb, and the outputs which are not used are consi dered as garbage outputs.

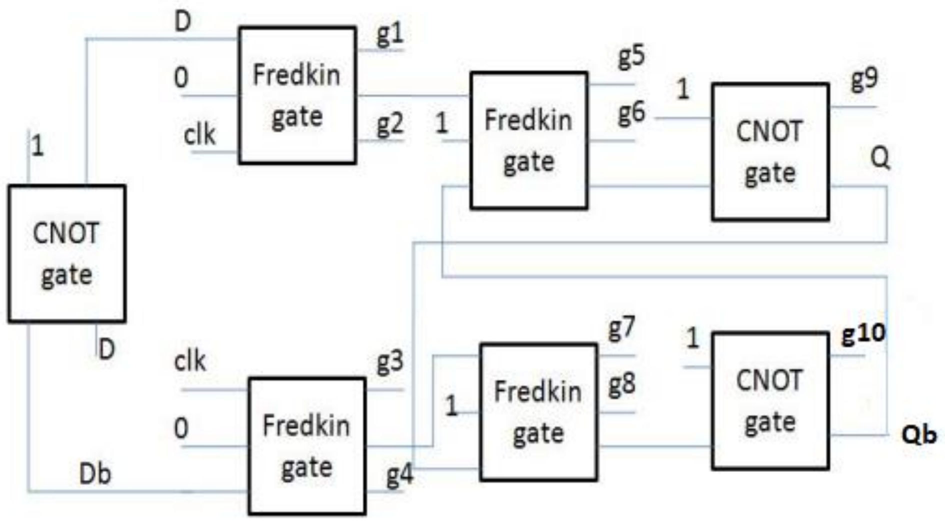


Figure 3.27: D flip flop

4. IMPLEMENTATION

4.1 Verilog codes for Reversible Logic Gates:

Verilog codes are written for reversible logic gates and simulated.

4.1.1 Fredkin Gate:

```
module  fredkin(a,b,c,x,y,z);
input  a,b,c;
output  x,y,z;
assign  x=a;
assign  y=(~a&b)|(a&c);
assign  z=(a&b)|(~a&c);
endmodule
```

4.1.2 FeynmanpGate

```
module  feynman(a,b,x,y);
input  a,b;
output  x,y;
xor(y,a,b);
buf(x,a);
endmodule
```

4.1.3 ToffolipGate

```
module  toffoli(a,b,c,x,y,z);
input  a,b,c;
output  x,y,z;
wire  w1;
buf(x,a);
buf(y,b);
and(w1,a,b);
xor(z,w1,c);
endmodule
```

4.1.4 Peres Gate:

```
module  peres(  a,b,c,x,y,z);
input  a,b,c;
output  x,y,z;
wire  w1;
buf(x,a);
xor(y,a,b);
and(w1,a,b);
xor(z,w1,c);
endmodule
```

4.1.5 HNG Gate:

```
module  hng(  a,b,c,d,p,q,r,  s);
input  a  b  c,d;
output  p  q  r  s;
assign  p=a  ;
assign  q=b  ;
assign  r=  a^b^c;
assign  s=(  a^b)&c^(a&b)^  d;
endmodule
```

4.1.6 TSG Gate:

```
module  tsg(a,b,c,d,p,q,r,s);
input  a,b,c,d;
output  p,q,r,s;
assign  p=a;
assign  q=(~a&~c^~b);
assign  r=(~a&~c^~b)^d;
assign  s=(~a&~c^~b)&d^(a&b^c);
endmodule
```

4.1.7 COG Gate:

```
module  cog(a,b,c,x,y,z);
input  a,b,c;
output  x,y,z;
assign  x=a;
assign  y=(~  a&b)^(a&c);
assign  z=~(  b^c);
endmodule
```

4.1.8 SRG Gate:

```
module  srg(a,b,c,d,p,q,r,s);
input  b,a,c,d;
output  q,p,r,s;
assign  q=a^c;
assign  p=a^b;
assign  r=(~a&b)^(~a&c)^(b&c);
assign  s=a^b^c^d;
endmodule
```

4.1.9 BVF Gate:

```
module  bvf(  a,b,c,d,p,q,r,s);
input  a,b  c,d;
output  p  q  r  s;
buf(p  a);
xor(  q,a,b);
buf(  r,c);
xor(s  c,d);
endmodule
```

4.1.10 BJN Gate:

```
module  project(a,b,c,p,q,r);
input  a,b,c;
output  p,q,r;
wire  w;
buf(  p,a);
buf(  q,b);
or(w  a,b);
xor(r,w,c);
endmodule
```

4.1.11 Double Feynman Gate:

```
module  project(a,b  c,p,q,r);
input  a  b,c;
output  p  q,r;
assign  p=a  ;
assign  q=a^b;
assign  r=a^c;
endmodule
```

4.1.12 DINV Gate:

```
module  project(a  b,c,p,q,r);
input  a  b,c;
output  p  q,r;
assign  p=~  a&b;
assign  q=a&b;
assign  r=a&c;
endmodule
```

4.2 AND gate and OR gate using Reversible Gates

4.2.1 AND Gate using Fredkin Gate:

```
module fredkin(a,b,c,x,y,z);
input a,b,c;
output x,y,z;
assign x=a;
assign y=(~a&b)|(a&c);
assign z=(a&b)|(~a&c);
endmodule

module andf(a b,c,p,q,r);
input a b,c;
output p q,r;
fredkin f1( a,b,c,p,q,r);
endmodule
```

4.2.2 OR Gate using Fredkin Gate:

```
module fredkin(a,b,c,x,y,z);
input a,b,c;
output x,y,z;
assign x=a;
assign y=(~a&b)|(a&c);
assign z=(a&b)|(~a&c);
endmodule

module orf(a,b,c,p,q,r);
input a,b,c;
output p,q,r;
fredkin f2(a,b,c,p,q,r);
endmodule
```

4.3 Combinational circuits using Reversible gates:

Combinational circuits are simulated by using reversible gates and Verilog codes for the combinational circuits are given below

4.3.1 Half adder using Peres Gate:

```
module peres(a,b,c,x,y,z);
input a,b,c;
output x,y,z;
wire w1;
buf(x,a);
xor(y,a,b);
and(w1,a,b);
xor(z,w1,c);
endmodule

module adder(a,b,c,x,s,cout);
input a,b,c;
output x,s,cout;
peres p1(a,b,c,x,s,cout);
endmodule
```

4.3.2 Full adder using HNG Gate

```
module hng( a,b,c,d,p,q,r,s);
input a,b c,d;
output p q,r,s;
assign p=a ;
assign q=b ;
assign r= a^b^c;
assign s=( a^b)&c^(a&b)^ d;
endmodule
```

```
module  fa_hng(a,b,cin,d,p,q,sum,cout);
input  a,b,cin,d;
output  p,q,sum,cout;
hng  h1(a,b,cin,d,p,q,sum,cout);
endmodule
```

4.3.3 Half Adder using BVF and BJN gate:

```
module  bjnga(a  b,c,p,q,r);
input  a  b,c;
output  p  q,r;
wire  w;
buf(p,a);
buf(q,b);
or(w,a,b);
xor(r,w,c);
endmodule

module  bvfg(  a,b,c,d,p,q,r,s);
input  a  b,c,d;
output  p  q,r,s;
buf(  p,a);
xor(  q,a,b);
buf(  r,c);
xor(s  c,d);
endmodule

module  ha(  a,b,g1,g2,sum,g3,g4,cout);
input  a,b;
output  g1,g2,sum,g3,g4,cout;
wire  w1,w2,w3,w4;
bvfg  b1(a,1,b,1,w1,w3,w2,w4);
bjnga  b2(0,w1,w2,g1,g2,sum);
bjnga  b3(w3,w4,1,g3,g4,cout);
endmodule
```

4.3.4 Half Subtractor using BVF and BJN Gate:

```
module bjnga( a b,c,p,q,r);
input a b,c;
output p q,r;
wire w;
buf(p,a);
buf(q,b);
or(w,a,b);
xor(r,w,c);
endmodule

module bvfg( a,b,c,d,p,q,r,s);
input a b,c,d;
output p q,r,s;
buf( p,a);
xor( q,a,b);
buf( r,c);
xor(s c,d);
endmodule

module hs( a,b,g1,g2,diff,g3,g4,bor);
input a,b;
output g1,g2,diff,g3,g4,bor;
wire w1,w2,w3,w4;
bvfg b1(a,0,b,1,w1,w3,w2,w4);
bjnga b2(0,w1,w2,g1,g2,diff);
bjnga b3(w3,w4,1,g3,g4,bor);
endmodule
```

4.3.5 1-Bit Comparator using BVF and BJN gates:

```
module bjnga(a b,c,p,q,r);
input a b,c;
output p,q r;
wire w;
buf(p,a);
buf(q,b);
or(w,a,b);
xor(r,w,c);
endmodule

module bvfg( a,b,c,d,p,q,r,s);
input a,b c,d;
output p q,r,s;
buf(p a);
xor(q a,b);
buf(r c);
xor(s c,d);
endmodule


module comp(a,b,g1,g2,g3,g4,g5,g6,e,l,g);
input a,b;
output g1,g2,g3,g4,g5,g6,e;
inout l,g;
wire w1,w2,w3,w4;
bvfg b1(a,1,b,1,w1,w3,w4,w2);
bjnga b2(w1,w2,1,g1,g2,l);
bjnga b3(w3,w4,1,g3,g4,g);
bjnga b4(l,g,1,g5,g6,e);
endmodule
```

4.3.6 2x1 Mux

```
module  rmux1(a  b,c,p,q,r);
input  a,b  c;
output  p,q  r;
assign  p=a  ;
assign  q=((~  a)&b)+(a&c  );
assign  r=(~  a)&c+a&(~b);
endmodule

module  mux2x1(s,i0,i1,g1,y,g2);
input  s,i0,i1;
output  g1,y,g2;
rmux1  r1(s,i0,i1,g1,y,g2);
endmodule
```

4.3.7 4x1 Mux

```
module  rmux1(a  b,c,p,q,r);
input  a,b  c;
output  p,q  r;
assign  p=a  ;
assign  q=((~  a)&b)+(a&c  );
assign  r=(~  a)&c+a&(~b);
endmodule


module  mux4x1(s,i,g,y);
input  [1:0]s;
input  [3:0]i;
output  [5:0]g;
output  y;
wire  w1,w2,w3;
rmux1  r1(s[  0],i[0],i[1],w1,w2,  g[0]);
rmux1  r2(w1,  i[2],i[3],g[1],w3,g[2]);
rmux1  r3(s[1],w2,w3,g[3],y,g[4]);
endmodule
```

4.3.8 8x1 Mux

```
module  rmux1(a  b,c,p,q,r);
input  a,b  c;
output  p,q  r;
assign  p=a  ;
assign  q=((~  a)&b)+(a&c  );
assign  r=(~  a)&c+a&(~b);
endmodule


module  mux8x1(s,i,g,y);
input  [2:0]s;
input  [7:0]i;
output  [9:0]g;
output  y;
wire  w1  w2,w3,w4,w5,w6,w7,w8,w9,w10;
rmux1  r1(s[  0],i[0],i[1],w1,w2,  g[0]);
rmux1  r2(w1  i[2]  i[3],w7,w3,g[  2]);
rmux1  r3(s[  1],w2,w3,w8,w9,g[4]);
rmux1  r4(w7,i[4],i[5],w4,w5,g[5]);
rmux1  r5(w1,i[6],i[7],g[6],w6,g[7]);
rmux1  r6(w8,w5,w6,g[3],w10,g[8]);
rmux1  r7(s[2],w9,w10,g[1],y,g[3]);
endmodule
```

4.3.9 1-Bit ALU:

```
module  feynman(p,q,a,b);
input  a,b;
output  p,q;
buf(p,a);
xor(q,a,b);
endmodule



module  fredkin(u,v,w,x,y,z);
input  x,y,z;
output  u,v,w;
assign  u=x;
assign  v=(~x&y)^(x&z);
assign  w=(x&y)^(~x&z);
endmodule



module  ri(d,e,f,l,m,n);
input  l,m,n;
output  d,e,f;
assign  d=l;
assign  e=(l&~m)+(m&n);
assign  f=(l&m)^n;
endmodule
```

```
module  hf(  a,b,c,d,s  cout);
input  a  b,c,d;
output  s,cout;
assign  s=a^b^c^d;
assign  cout=(a^b)&(c)^(a&b)^d;
endmodule


module  alu(a,b,s,cin,c,cout,f,x,y,z,g);
input  a,b;
input  [2:0]s;
input  cin;
input  [5:1]c;
output  cout,f;
inout  x,y,z;
wire  w1,w2,w3,w4,w5,w6,w7;
output  [8:0]g;
feynman  f1(w1,w2,s[2],c[1]);
feynman  f2(w3,w4,s[0],c[2]);
ri  r1(g[1],g[2],z,cin,w2,c[3]);
fredkin  f3(w5,y,g[3],b,s[1],w3);
ri  r2(g[4],g[5],w6,w1,w4,c[4]);
feynman  f4(g[6],w7,w5,s[1]);
ri  r3(g[7],g[8],x,w7,w6,a);
hf  h1(z,y,x,c[5],f,cout);
endmodule
```

4.4 Sequential Circuits

Sequential circuits like SR Flip flop and D Flip flop are designed and simulated using basic reversible gates.

4.4.1 SR Flip Flop

```
module feynman(p,q,a,b);
input a,b;
output p,q;
buf(p,a);
xor(q,a,b);
endmodule


module fredkin(u,v,w,x,y,z);
input x,y,z;
output u,v,w;
assign u=x;
assign v=(~x&y)^(x&z);
assign w=(x&y)^(~x&z);
endmodule


module reversible_srff(S,R,Q,Qb,clk);
input S,R,clk;
output Q,Qb;
wire g1,g2,g3,g4,g5,g6,g7,g8,g9,g10;
wire w1,w2,w3,w4,w5,w6;
fredkin fr1(g1,w1,g2,R,1'b0,clk);
fredkin fr2(g3,w2,g4,clk,1'b0,S);
fredkin fr3(g5,g6,w3,w1,1'b1,Qb);
fredkin fr4(g7,g8,w5,w2,1'b1,Q);
feynman fe1(g9,Q,1'b1,w3);
feynman fe2(g10,Qb,1'b1,w5);
endmodule
```

4.4.2 D-Flip Flop

```
module  feynman(p,q,a,b);
input  a,b;
output  p,q;
buf(p,a);
xor(q,a,b);
endmodule

module  fredkin(u,v,w,x,y,z);
input  x,y,z;
output  u,v,w;
assign  u=x;
assign  v=(~x&y)^(x&z);
assign  w=(x&y)^(~x&z);
endmodule

module  dff(D,Q,Qb,clk);
input  D,clk;
output  Q,Qb;
wire  g1,g2,g3,g4,g5,g6,g7,g8,g9,g10;
wire  w1,w2,w3,w4,w5,w6,w7,w8;
wire  Db;
feynman  fe3(w7,Db,1,D);
fredkin  fr1(g1,w1,g2,Db,1'b0,clk);
fredkin  fr2(g3,w2,g4,clk,1'b0,D);
fredkin  fr3(g5,g6,w3,w1,1'b1,Qb);
fredkin  fr4(g7,g8,w5,w2,1'b1,Q);
feynman  fe1(g9,Q,1'b1,w3);
feynman  fe2(g10,Qb,1'b1,w5);
endmodule
```

4.5 Testbench

4.5.1 Feynman Gate

```
module tb_feynman();
reg a,b;
wire x,y;
feynman uut(a,b,x,y);
initial begin
  a=0;b=0;
  #10 a=0;b=1;
  #10 a=1;b=0;
  #10 a=1;b=1;
end
endmodule
```

The above testbench is similar for all two input circuits.

4.5.2 Fredkin Gate:

```
module tb_fredkin();
reg a,b,c;
wire x,y,z;
fredkin uut(a,b,c,x,y,z);
initial begin
  a=0;b=0; c=0;
  #10 a=0 ;b=1 ;
  #10 a=1 ;b=0 ;
  #10 a=1 ;b=1 ;
  #10 a=0;b=0;c=1;
  #10 a=0;b=1;
  #10 a=1;b=0;
  #10 a=1;b=1;
end
endmodule
```

The above testbench is similar for all three input circuits.

4.5.3 HNG Gate:

```
module tb_hng();
reg a,b,c d;
wire p q,r,s;
hng uut(a b,c,d,p,q,r,s);
initial begin
 a=0 ;b=0;c=0;d=0;
 #10 a=0 ;b=1;
 #10 a=1;b=0;
 #10 a=1;b=1;
 #10 a=0;b=0;c=1;d=0;
 #10 a=0;b=1;
 #10 a=1;b=0;
 #10 a=1;b=1;
 #10 a=0;b=0;c=0;d=1;
 #10 a=0;b=1;
 #10 a=1;b=0;
 #10 a=1;b=1;
 #10 a=0;b=0;c=1;d=1;
 #10 a=0;b=1;
 #10 a=1;b=0;
 #10 a=1;b=1;
end
endmodule
```

The above test bench is similar for all four input circuits.

4.5.4 Half Adder

```
module  tb_ha();
reg  a,b;
wire  x,y;
ha  uut(a,b,x,y);
initial  begin
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=0;
  #10  a=1;b=1;
end
endmodule
```

4.5.5 Half Subtractor

```
module  tb_hs();
reg  a,b;
wire  x,y;
hs  uut(a,b,x,y);
initial  begin
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=0;
  #10  a=1;b=1;
end
endmodule
```

4.5.6 1-bit Comparator

```
module  tb_comp();
reg  a,b;
wire  x,y;
comp  uut(a,b,x,y);
initial  begin
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=0;
  #10  a=1;b=1;
end
endmodule
```

4.5.7 Full Adder

```
module  tb_fa_hng();
reg  a,b,c;
wire  x,y,z;
fa_hng  uut(a,b,c,x,y,z);
initial  begin
  a=0;b=0;  c=0;
  #10  a=0  ;b=1p;
  #10  a=1  ;b=0  ;
  #10  a=1  ;b=1;
  #10  a=0;b=0;c=1;
  #10  a=0;b=1;
  #10  a=1;b=0;
  #10  a=1;b=1;
end
endmodule
```

4.5.8 2x1 Mux

```
module  tb_mux2x1();
reg  s;
reg  [1:0]i;
wire  [1:0]g;
wire  y;
project  uut(s,i,g,y);
initial  begin
  s=0;
  i[0]=0;
  #10  i[0]=1;
  #10  s=1;  i[1]=0;
  #10  i[1]=1;
  end
endmodule
```

4.5.9 4x1 Mux

```
module  tb_mux4x1();
reg  [1:0]s;
reg  [3:0]i;
wire  [5:0]g;
wire  y;
mux4x1  uut(s,i,g,y);
initial  begin
  s[0]=0;s[1]=0;
  i=0;
  #10  i[0]=1;
  #10  s[0]=1;s[1]=0;
  i=0;
  #10  i[1]=1;
  #10  s[0]=0;s[1]=1;
  i=0;
  #10  i[2]=1;

  #10s[0]=1;s[1]=1;
  i=0;
  #10  i[3]=1;
  end
endmodule
```

4.5.10 8x1 Mux:

```
module  tb_mux8x1();
reg  [2:0]s;
reg  [7:0]i;
wire  [9:0]g;
wire  y;
mux8x1  uut(s,i,g,y);
initial  begin
  s[0]=0;s[1]=0;s[2]=0;
  i=0;
  #10  i[0]=1;

  #10s[0]=1;s[1]=0;s[2]=0;
  i=0;
  #10  i[1]=1;

  #10s[0]=0;s[1]=1;s[2]=0;
  i=0;
  #10  i[2]=1;

  #10s[0]=1;s[1]=1;s[2]=0;
  i=0;
  #10  i[3]=1;

  #10s[0]=0;s[1]=0;s[2]=1;
  i=0;
  #10  i[4]=1;

  #10s[0]=1;s[1]=0;s[2]=1;
  i=0;
  #10  i[5]=1;

  #10s[0]=0;s[1]=1;s[2]=1;
  i=0;
  #10  i[6]=1;

  #10s[0]=1;s[1]=1;s[2]=1;
  i=0;
  #10  i[7]=1;
  end
endmodule
```

4.5.11 1-Bit ALU

```
module  tb_alu();
reg  a,b;
reg  [2:0]s;
reg  cin;
reg  [5:1]c;
wire  cout,f;
wire  x,y,z;
wire  [8:0]g;
alu  uut(a,b,s,cin,c,cout,f,x,y,z,g);

initial  begin
  c[1]=1;c[2]=1;c[3]=0;c[4]=0;c[5]=0;
  {s,cin}=0;
  a=0;b=0;
  #10  a=1;b=0;
  #10{s,cin}=1;
  a=0;b=0;
  #10  a=1;b=0;
  #10{s,cin}=2;
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=1;
  #10{s,cin}=3;
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=1;
  #10{s,cin}=4;
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=1;
  #10{s,cin}=5;
  a=0;b=0;
  #10  a=0;b=1;
  #10  a=1;b=1;
  #10{s,cin}=8;
```

```
      a=0;b=1;
  #10  a=1;b=1;
  #10  c[1]=1;c[2]=1;c[3]=0;c[4]=0;c[5]=1;
  {s,cin}=8;
  a=0;b=1;
  #10  a=1;b=1;
  #10{s,cin}=10;
  a=0;b=1;
  #10  a=1;b=1;
  #10  c[1]=1;c[2]=1;c[3]=0;c[4]=0;c[5]=0;
  {s,cin}=8;
  a=0;b=1;
  #10  a=1;b=1;
  end
Endmodule
```

4.5.12 SRFF

```
module  tb_srff;
reg  S,R,clk;
wire  Q,Qb;
reversible_srff  r_s1(S,R,Q,Qb,clk);
initial  begin
  clk  =  1;
  S=  1;  R=  0;
  #10;  S=  0;  R=  1;
  #10;  S=  0;  R=  0;
  #10;  S=  1;  R=1;
end
endmodule
```

4.5.13 DFF

```
module  tb_dff;
reg  D,clk;
wire  Q,Qb;
dff  dff1(D,Q,Qb,clk);
initial
begin
  clk  =  1;
  D  =  1;
  #100;  D  =  0;
end
endmodule
```

5. SIMULATION RESULTS

5.1 Fredkin Gate

The truth table of Fredkin gate is shown in Table 5.1

A	B	C		P	Q	R
0	0	0		0	0	0
0	0	1		0	0	1
0	1	0		0	1	0
0	1	1		0	1	1
1	0	0		1	0	0
1	0	1		1	1	0
1	1	0		1	0	1
1	1	1		1	1	1

Table 5.1: Fredkin Gate Truth Table

Simulation result for Fredkin gate is shown in Figure 5.1. The input vectors are given as I (a, b, c) and output vector is given as O (x, y, z).

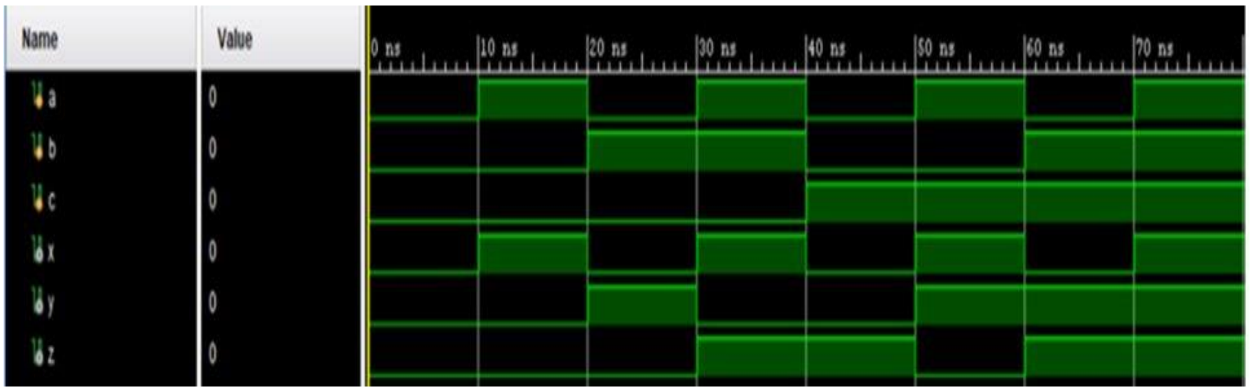


Figure 5.1: Simulation result for Fredkin Gate

5.2 Feynman Gate

The truth table for Feynman Gate is shown in Table 5.2.

INPUT		OUTPUT	
P	Q	X = P	Y = P XOR Q
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 5.2 Feynman gate Truth table

The simulation result for Feynman gate is shown in Figure 5.2 and the input vector is given as I(a, b, c) and output vector is given as O(x, y).

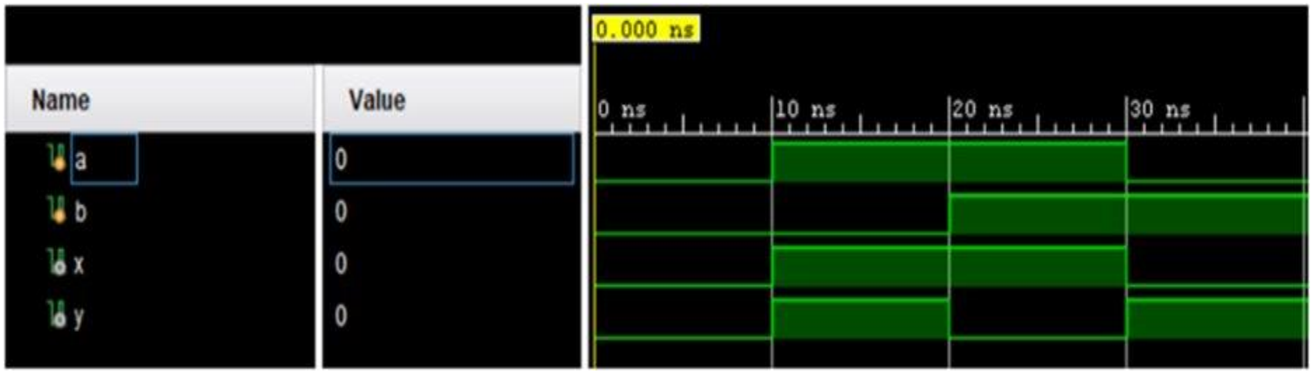


Figure 5.2: Simulation result for Feynman gate

5.3 Toffoli Gate

The truth table for Toffoli Gate is shown in Table 5.3.

INPUT			OUTPUT		
A	B	C	P = A	Q = B	R = AB XOR C
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	1	1	1
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Table 5.3: Toffoli gate Truth table

The simulation result for Feynman gate is shown in figure 5.3 and input vectors is given as I(a, b, c) and output vector is given as O (x, y, z).

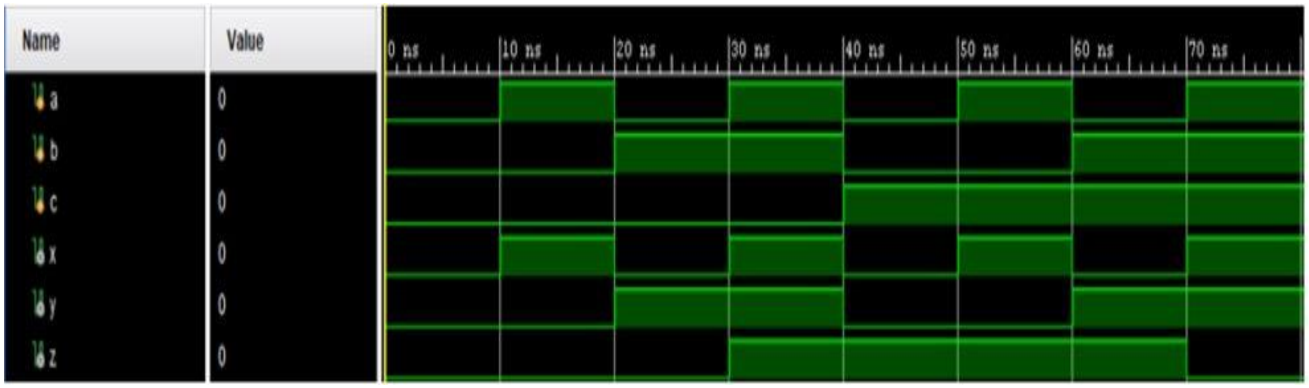
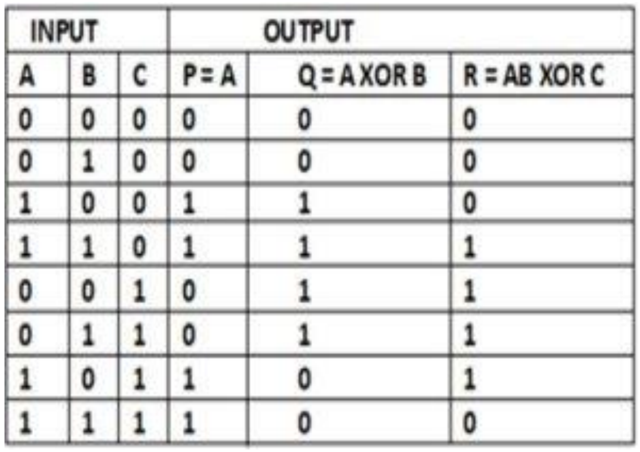


Figure 5.3: Simulation result for Toffoli gate

5.4 Peres Gate

The truth table for Peres Gate is shown in Table 5.4.



INPUT			OUTPUT		
A	B	C	P = A	Q = A XOR B	R = AB XOR C
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	1	1	0
1	1	0	1	1	1
0	0	1	0	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	0	0

Table 5.4: Peres Gate Truth table

The simulation result for Peres gate is shown in figure 5.4 and the input vectors are gi ven as I(a, b, c) and output vector is given as O (x, y, z).

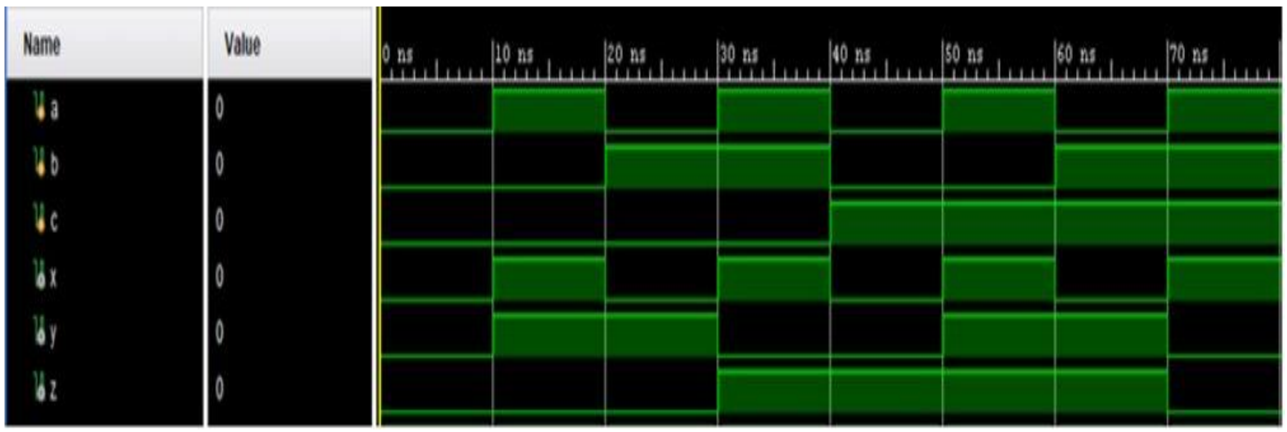
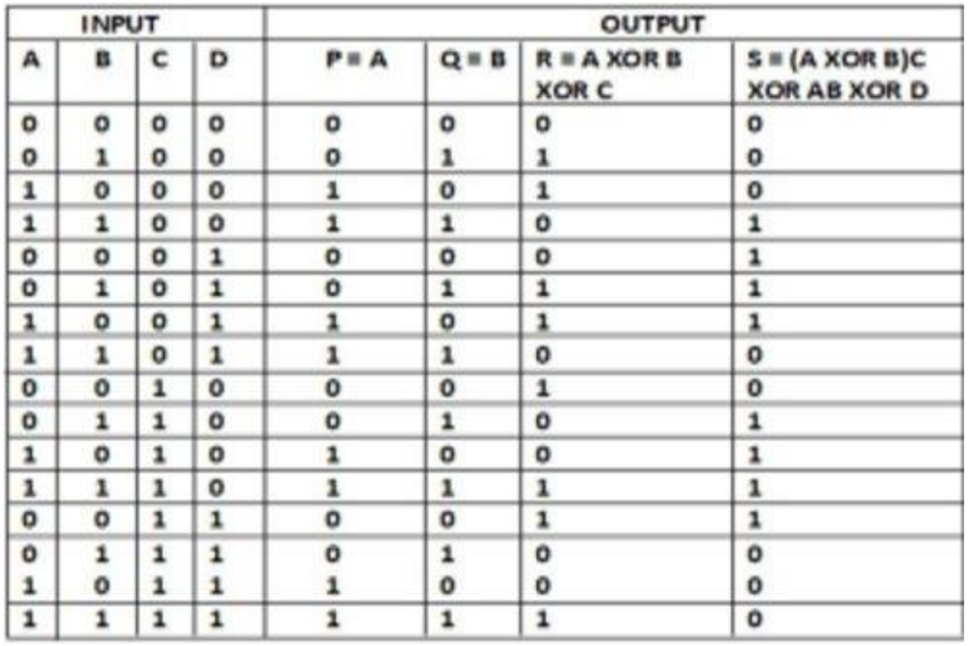


Figure 5.4: Simulation result for Peres gate

5.5 HNG Gate

The truth table for HNG Gate is shown in Table 5.5



INPUT				OUTPUT			
A	B	C	D	P = A	Q = B	R = A XOR B XOR C	S = (A XOR B)C XOR AB XOR D
0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	1	0	1
0	0	0	1	0	0	0	1
0	1	0	1	0	1	1	1
1	0	0	1	1	0	1	1
1	1	0	1	1	1	0	0
0	0	1	0	0	0	1	0
0	1	1	0	0	1	0	1
1	0	1	0	1	0	0	1
1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	1	1	0	1	0	0
1	0	1	1	1	0	0	0
1	1	1	1	1	1	1	0

Table 5.5: H.N.G. Gate Truth table

The simulation result for H.N.G. gate is shown in figure 5.5 and input vectors is given as I(a, b, c, d) and output vector is given as O (p, q, r, s).

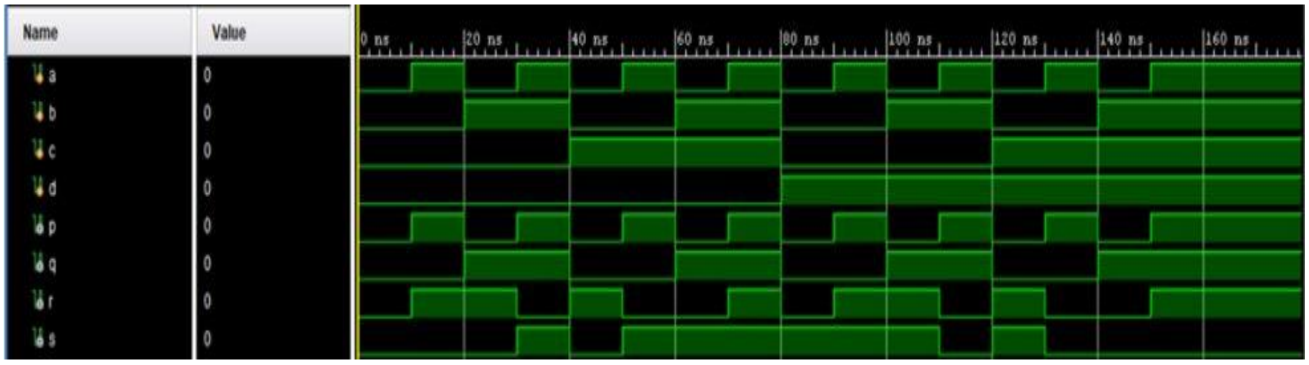
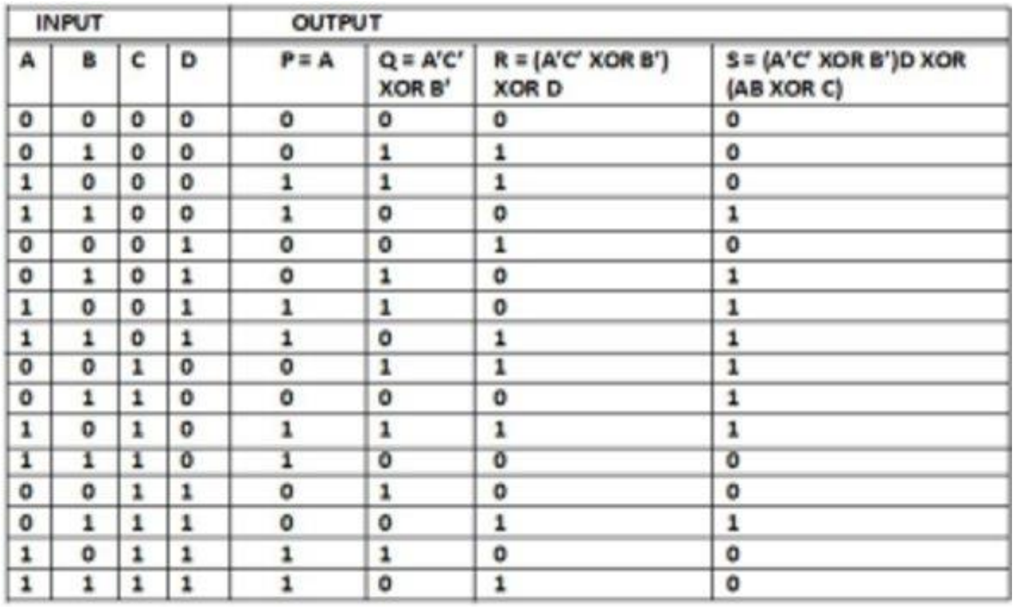


Figure 5.5: Simulation result for H.N.G. gate

5.6 TSG Gate

The truth table for TSG Gate is shown in Table 5.6.



INPUT				OUTPUT			
A	B	C	D	P = A	Q = A'C' XOR B'	R = (A'C' XOR B') XOR D	S = (A'C' XOR B')D XOR (AB XOR C)
0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0
1	0	0	0	1	1	1	0
1	1	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	1	0	1	0	1	0	1
1	0	0	1	1	1	0	1
1	1	0	1	1	0	1	1
0	0	1	0	0	1	1	1
0	1	1	0	0	0	0	1
1	0	1	0	1	1	1	1
1	1	1	0	1	0	0	0
0	0	1	1	0	1	0	0
0	1	1	1	0	0	1	1
1	0	1	1	1	1	0	0
1	1	1	1	1	0	1	0

Table 5.6: T.S.G. Gate Truth table

The simulation result for HNG gate is shown in figure 5.6 and input vectors is given a s I(a, b, c, d) and output vector is given as O (p, q, r, s).

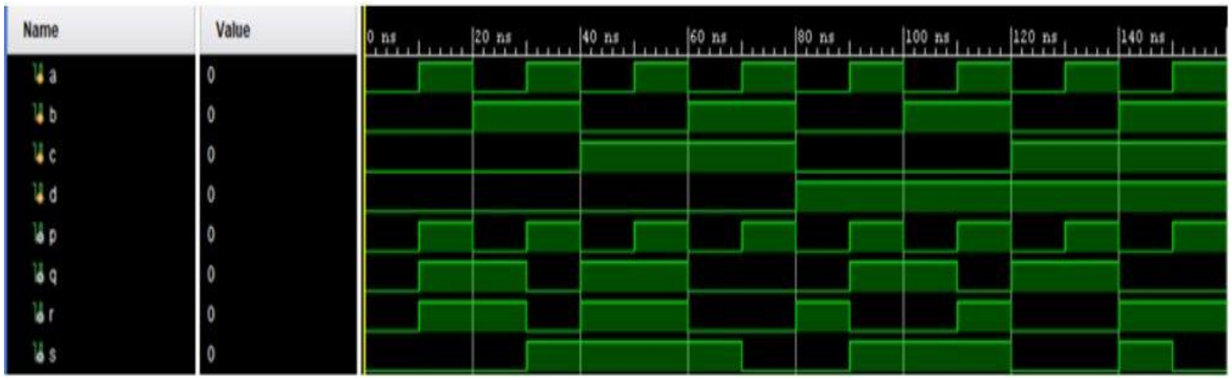


Figure 5.6: Simulation result for T.S.G. gate

5.7 COG Gate

The truth table for COG Gate is shown in table 5.7.

INPUTS			OUTPUTS		
A	B	C	P	Q	R
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Table 5.7: C.O.G. Gate Truth table

The simulation result for COG Gate is shown in figure 5.7 and input vectors is given as I(a, b, c) and output vector is given as O (x, y, z).

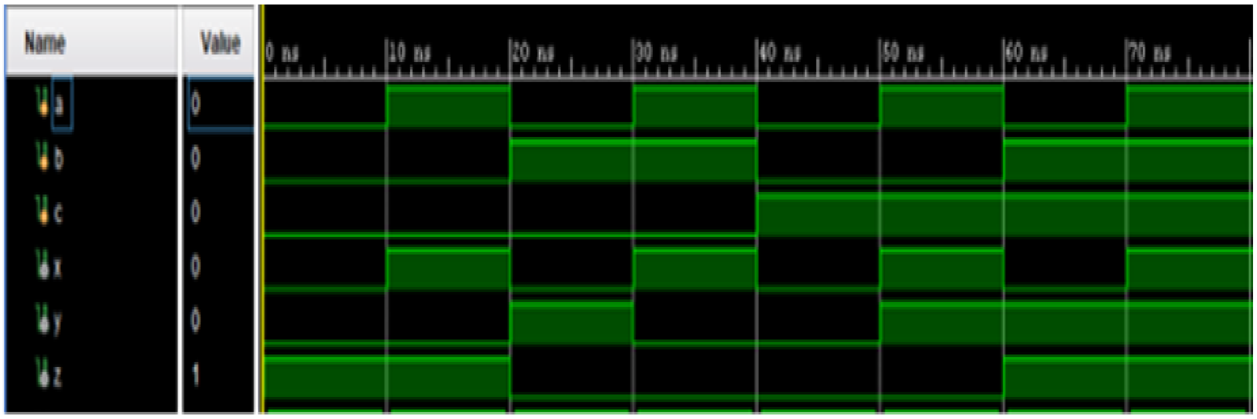


Figure 5.7: Simulation result for C.O.G. gate

5.8 SRG Gate

The truth table for SRG Gate is shown in Table 5.8.

A	B	C	D	P	Q	R	S
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	1	1	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1
1	0	0	1	1	1	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	1	0	1
1	1	1	0	0	0	1	1
1	1	1	1	0	0	1	0

Table 5.8: S.R.G. Gate Truth table

The simulation result for SRG Gate is shown in figure 5.8 and input vectors is given a s I(a, b, c, d) and output vector is given as O (p, q, r, s).

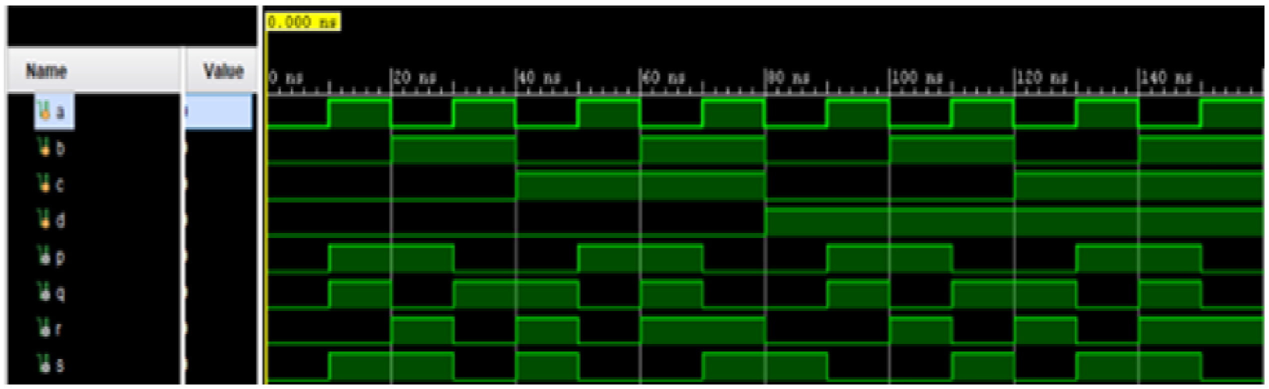
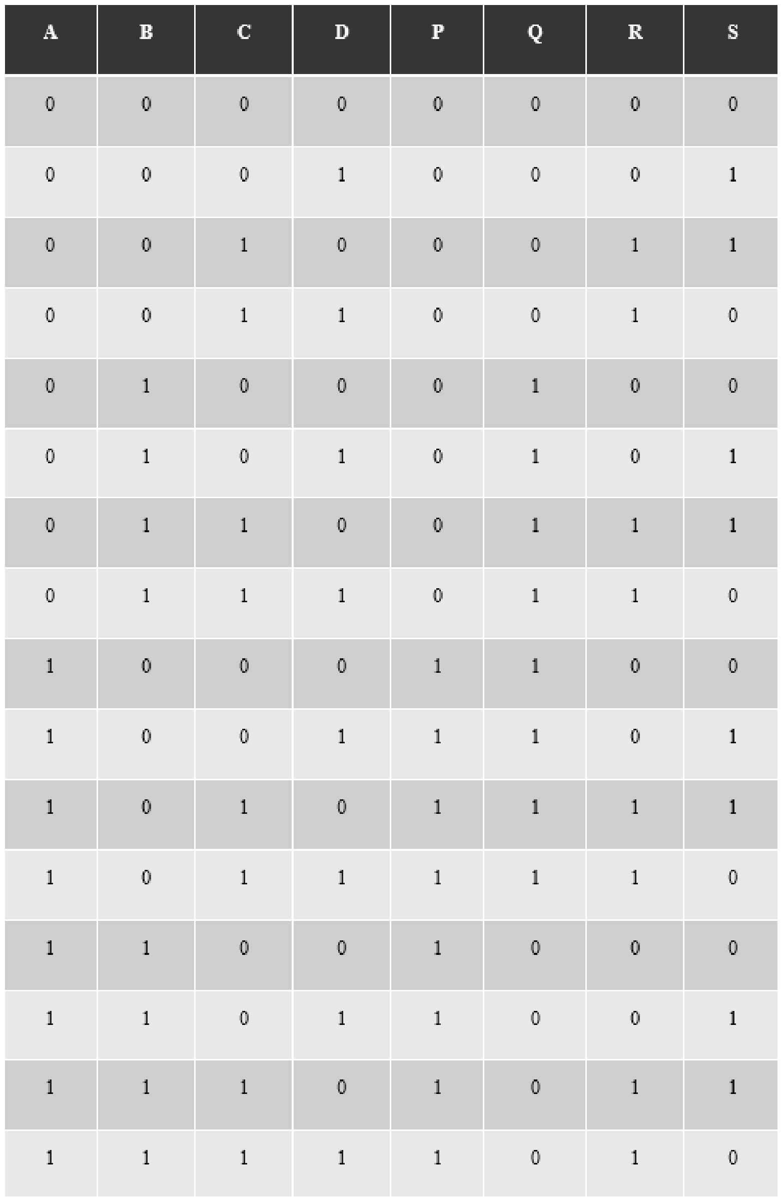


Figure 5.8: Simulation result for S.R.G. gate

5.9 BVF Gate

The truth table for BVF Gate is shown in Table 5.9.



A	B	C	D	P	Q	R	S
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	1
0	1	1	1	0	1	1	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

Table 5.19: BVF Gate Truth table

The simulation result for BVF Gate is shown in figure 5.9 and input vectors is given a s I(a, b, c, d) and output vector is given as O (p, q, r, s).

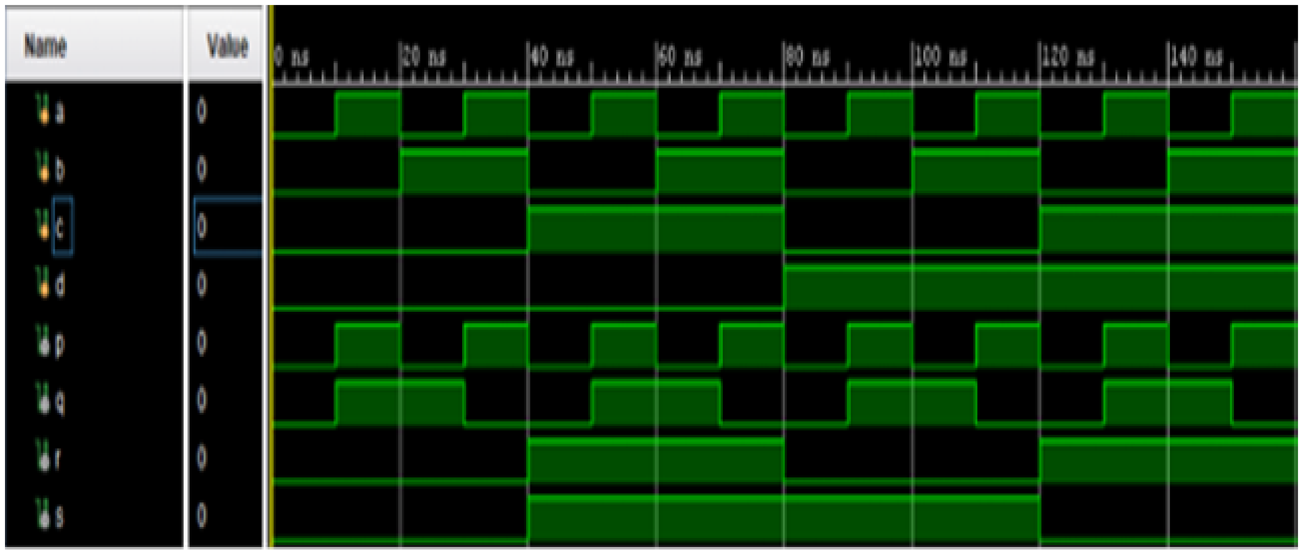


Figure 5.9: Simulation result for B.V.F. gate

5.10 BJN Gate

The truth table for BJN Gate is given in Table 5.10.

A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	0

Table 5.10: B.J.N. Gate Truth table

The simulation result for B.J.N. Gate is shown in figure 5.10 and input vectors is given as I(a, b, c) and output vector is given as O (p, q, r).

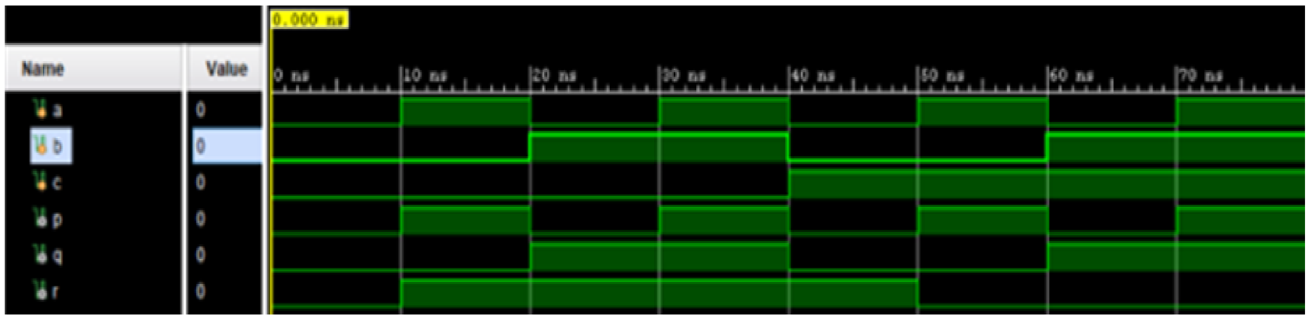


Figure 5.10: Simulation result for B.J.N. gate

5.11 Double Feynman

The truth table for Double Feynman Gate is given in Table 5.11

A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	1	0	0

Table 5.11: Double Feynman Gate Truth table

The simulation result for Double Feynman Gate is shown in figure 5.11 and input vect ors is given as I(a, b, c) and output vector is given as O (p, q, r).

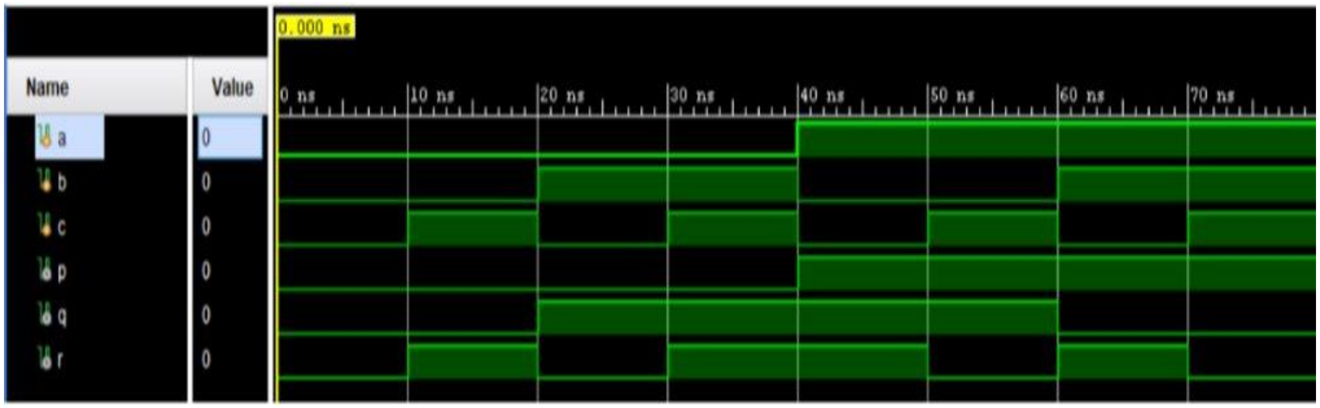


Figure 5.11: Simulation result for Double Feynman gate

5.12 DINV Gate

The truth table for Double Feynman Gate is given in Table 5.12.

A	B	C	P	Q	R
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	0	1	1

Table 5.12: DINV Gate Truth table

The simulation result for DINV Gate is shown in figure 5.12 and input vectors is given as I(a, b, c) and output vector is given as O (p, q, r).

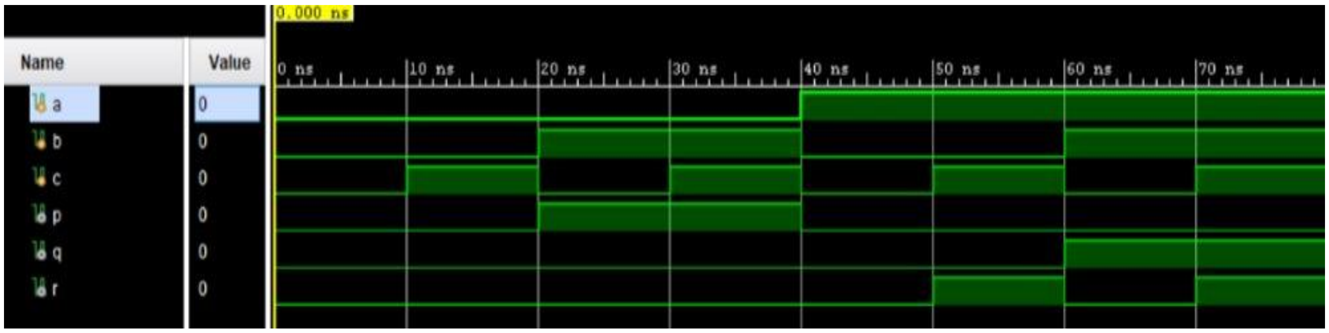
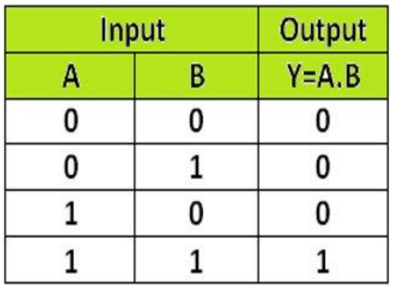


Figure 5.12: Simulation result for DINV gate

5.13 AND gate using Fredkin

The truth table for AND Gate is given in Table 5.13.



Input		Output
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Table 5.13: AND Gate Truth table

The simulation result for AND Gate is shown in figure 5.13 and input vectors is given as I(a, b) and output vector is given as O (r).

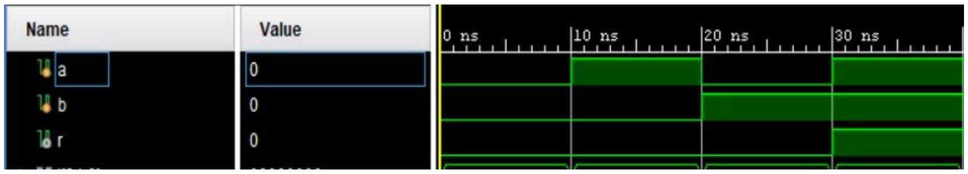
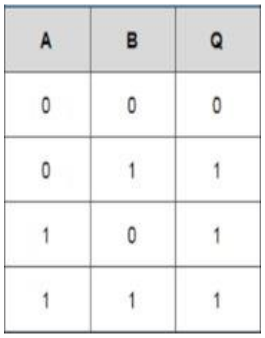


Figure 5.13: Simulation result for AND gate

5.14 OR Gate using Fredkin Gate

The truth table for OR Gate is given as Table 5.14.



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Table 5.14: OR Gate Truth table

The simulation result for OR Gate is shown in figure 5.14 and input vectors is given as I(a, c) and output vector is given as O (r).

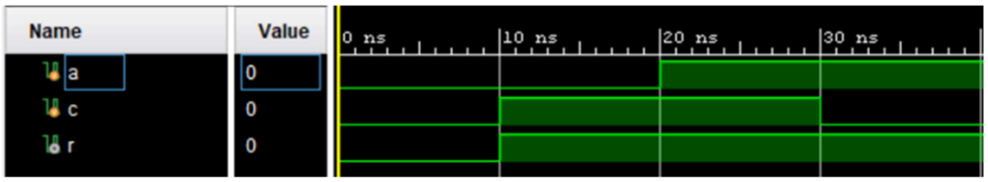
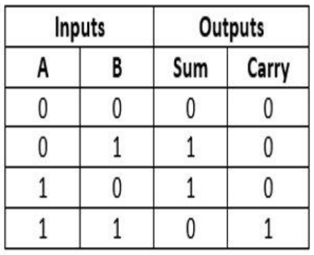


Figure 5.14: Simulation result for OR gate

5.15 Half Adder using Peres gate

The truth table for Half Adder is given as Table 5.15.



Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 5.15: Half Adder Truth table

The simulation result for Half adder using Peres gate is shown in figure 5.15 and input vectors is given as I(a, b) and output vector is given as O (s, cout).

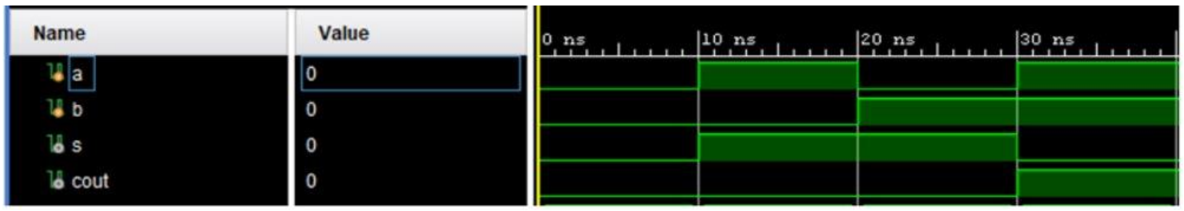


Figure 5.15: Simulation result for Half Adder using Peres Gate

5.16 Half adder using BVF and BJN Gate

The truth table for Half Adder is shown in Table 5.15.

The simulation result for Half adder using BVF and BJN Gates is shown in figure 5.4 6 and input vectors is given as I(a, b) and output vector is given as O (sum, cout).

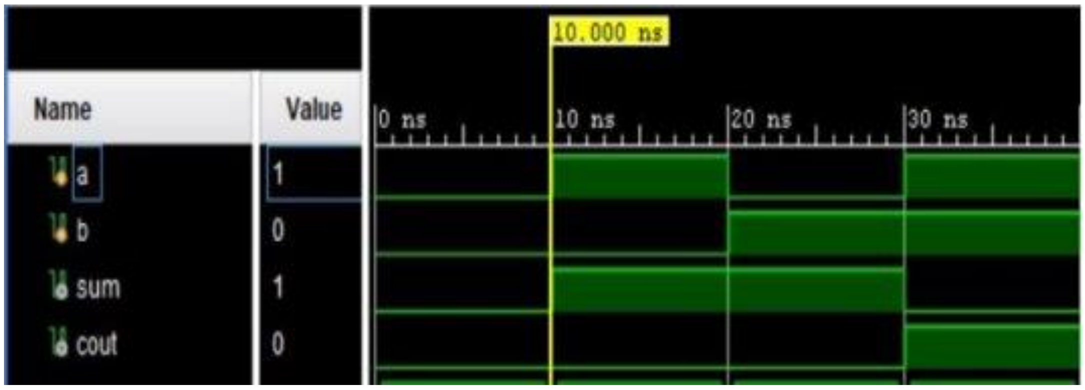


Figure 5.16: Simulation result for Half Adder using B.V.F. and B.J.N. gates

5.17 Half Subtractor using BVF and BJN Gates

The truth table for Half Subtractor is shown in Table 5.16.

INPUTS		OUTPUTS	
A	B	DIFFERENCE	BORROW
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 5.16: Half Subtractor Truth table

The simulation result for Half subtractor using BVF and BJN gates is shown in figure 5.17 and input vectors is given as I(a, b) and output vector is given as O (diff, bor).

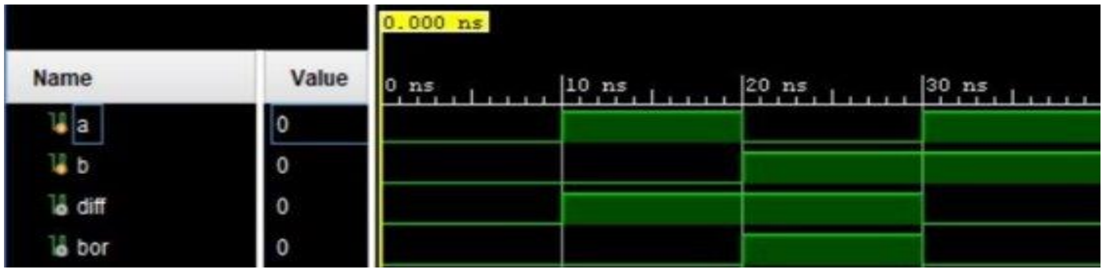


Figure 5.17: Simulation result for Half Subtractor using B.V.F. and B.J.N. gates

5.18 1-Bit Comparator using B.V.F. and B.J.N. Gates

The truth table for 1-bit comparator is given in Table 5.17.

INPUTS		OUTPUTS		
A	B	$F_{A>B}$	$F_{A=B}$	$F_{A<B}$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Table 5.17: 1-Bit Comparator Truth table

The simulation result for 1bit comparator using BVF and BJN Gates is shown in figure 5.18 and input vectors is given as I(a, b) and output vector is given as O (l, e, g).

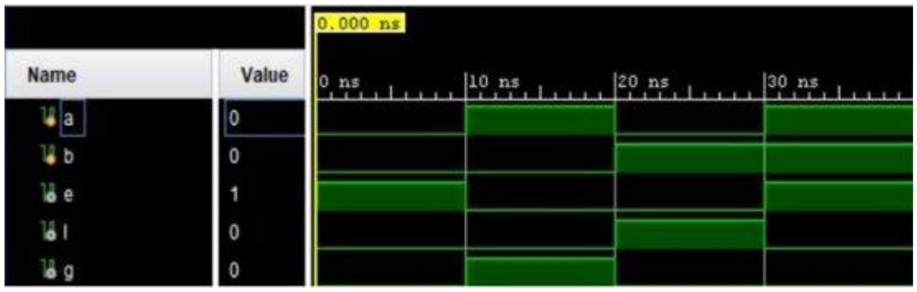
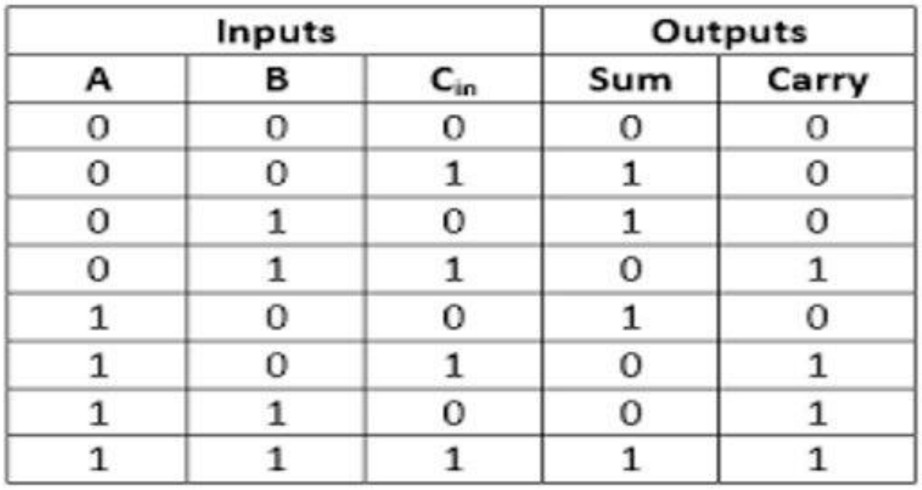


Figure 5.18: Simulation result for 1-bit Comparator using B.V.F. and B.J.N. gates

5.19 Full Adder using HNG Gate

The truth table for Half Adder is given in Table 5.18.



Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 5.18: Full Adder Truth table

The simulation result for Full adder using HNG Gate is shown in figure 5.19 and input vectors is given as I(a, b, cin) and output vector is given as O (sum, cout).

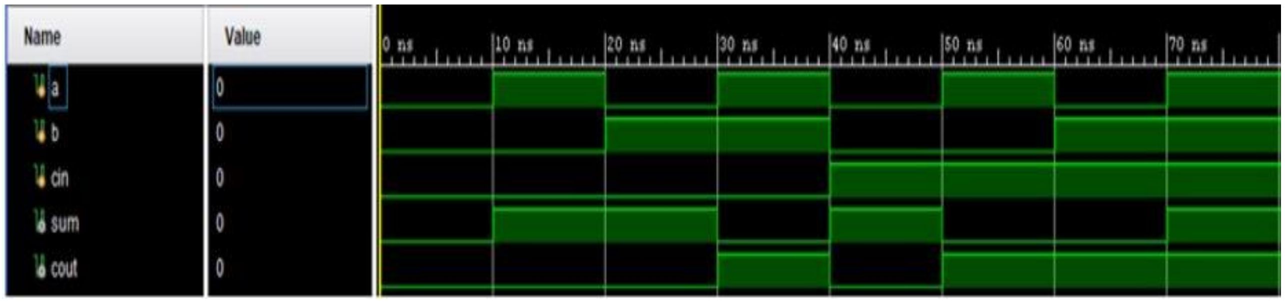


Figure 5.19: Simulation result for Full Adder using HNG

5.20 2x1 Mux

The truth table for 2x1 Mux is shown in Table 5.19.

S	I_0	I_1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 5.19: 2x1 Mux Truth table

The simulation result for 2x1 Mux is shown in figure 5.20 and input vectors is given as I(s, i0, i1) and output vector is given as O (y).

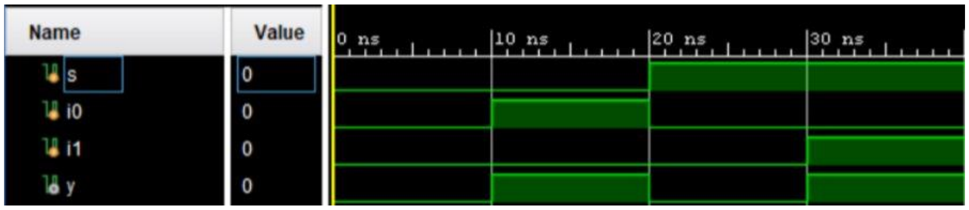


Figure 5.20: Simulation result for 2x1 Mux

5.21 4x1 Mux

The truth table for 4x1 Mux is shown in Table 5.20.

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 5.20: 4x1 Mux Truth table

The simulation result for 4x1 Mux is shown in figure 5.21 and input vectors is given as I(s[1:0], i[3:0]) and output vector is given as O (y).

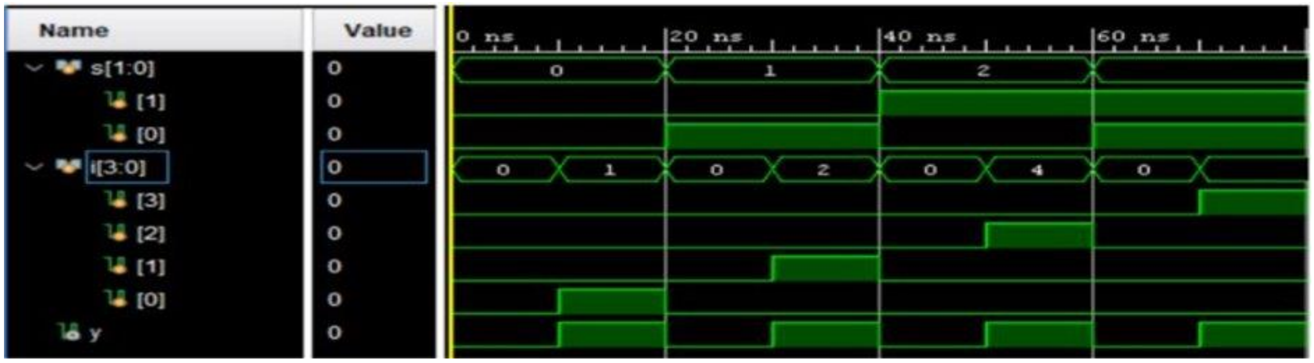


Figure 5.21: Simulation result for 4x1 Mux

5.22 8x1 Mux

The truth table for 8x1 Mux is shown in Table 5.21,

S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Table 5.21: 8x1 Mux Truth table

The simulation result for 8x1 Mux is shown in figure 5.22 and input vectors is given as I(s[2:0], i[7:0]) and output vector is given as O (y).

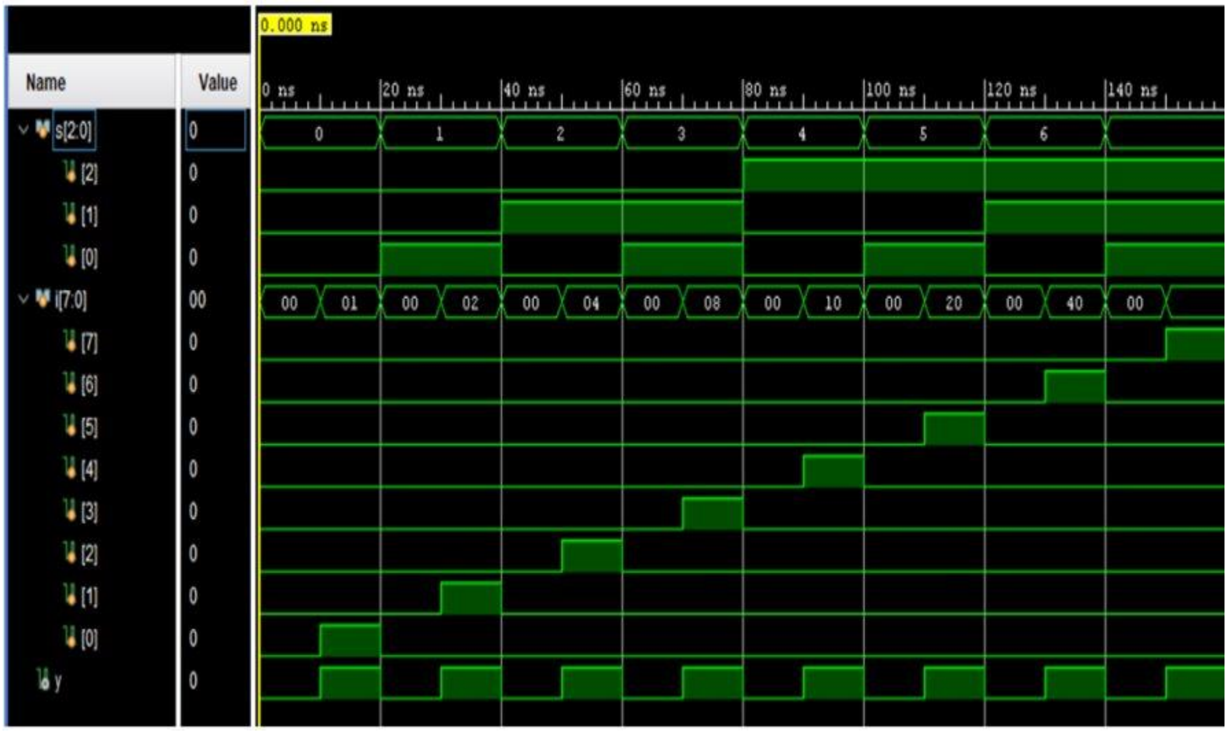


Figure 5.22: Simulation result for 8x1 Mux

5.23 1-Bit ALU

The functional table for ALU operations is shown in Table 5.22.

S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A+1	Increment A
0	0	1	0	F = A+B	Addition
0	0	1	1	F = A+B+1	Add with carry
0	1	0	0	F = A+B'	Subtract with borrow
0	1	0	1	F = A+B'+1	Subtraction
0	1	1	0	F = A-1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	X	F = A∨B	OR
1	0	0	X	F = A⊕B	EX-OR
1	0	1	X	F = A^B	AND
1	0	1	X	F = A'	NOT

Table 5.22: Functional table for ALU operations

The simulation result for 1-Bit ALU is shown in figure 5.23 and input vectors is given as I(a, b, s[2:0], cin) and output vector is given as O (f, cout, c[5:1]).

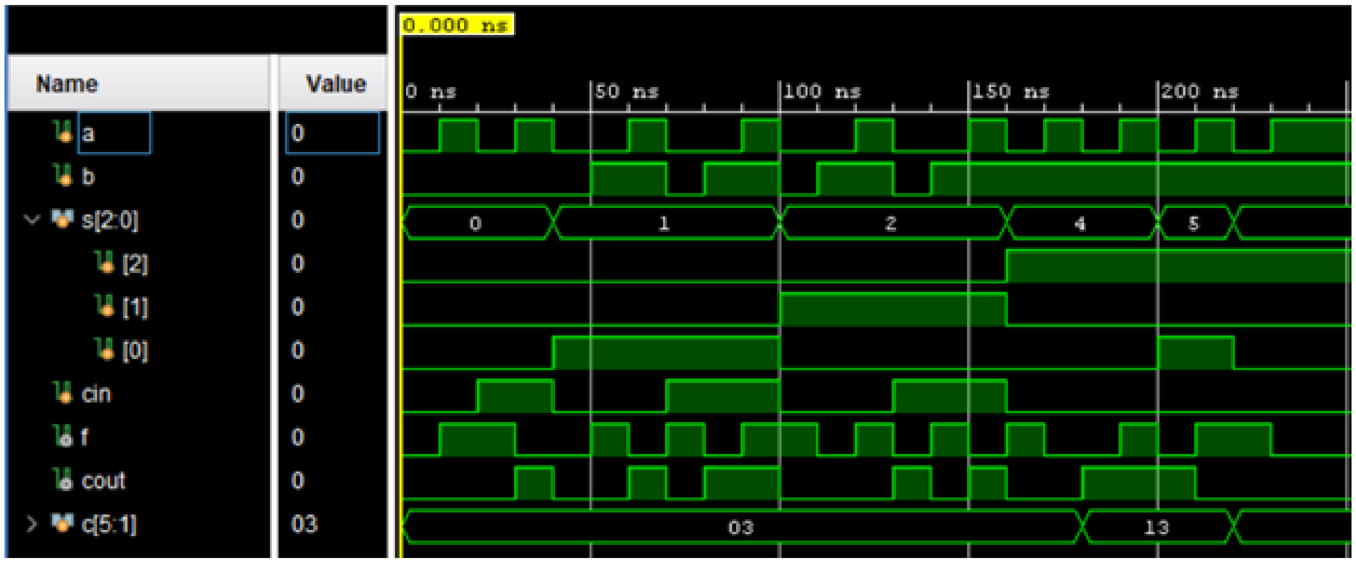
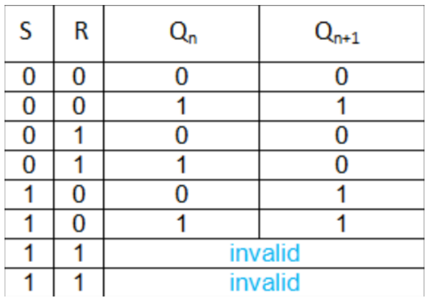


Figure 5.23: Simulation result for 1-Bit ALU

5. 24 SR Flip Flop

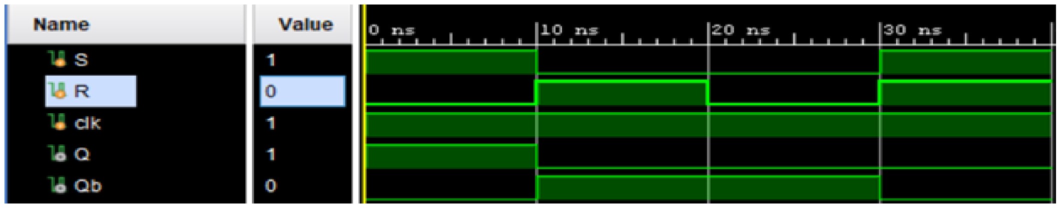
The truth table for SR Flip Flop is shown in Table 5.23.



S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	invalid	
1	1	invalid	

Table 5.23: SR Flip Flop Truth Table

The simulation result for SR Flip Flop is shown in figure 5.24 and input vectors is giv en as I(S, R, clk) and output vector is given as O (Q, Qb).

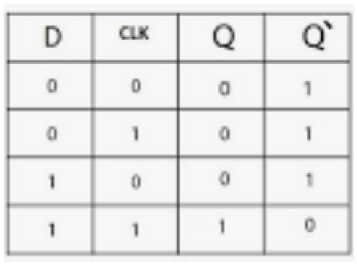


Name	Value
S	1
R	0
clk	1
Q	1
Qb	0

Figure 5.24: Simulation results for SR Flip Flop

5. 25 D-Flip Flop

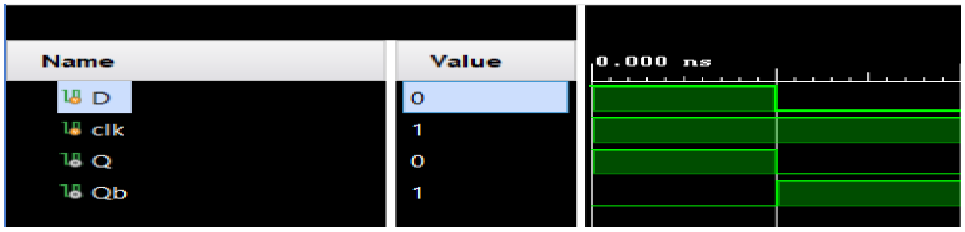
The truth table for D-Flip Flop is shown in Table 5.24.



D	CLK	Q	Q`
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Table 5.24: Dff truth table

The simulation result for DFlip Flop is shown in figure 5.25 and input vectors is given as I(d, clk) and output vector is given as O (Q, Qb).



Name	Value
D	0
clk	1
Q	0
Qb	1

Figure 5.25: Simulation results for D-Flip Flop

5.26 Power Dissipation of Xor/ Xnor

The schematic of reversible Xor gate is shown in Figure 5.26 which is implemented in LTspice where ten transistors are used, five PMOS and five NMOS. The inputs ports are given by a and b, and output ports are given by out1 and out2.

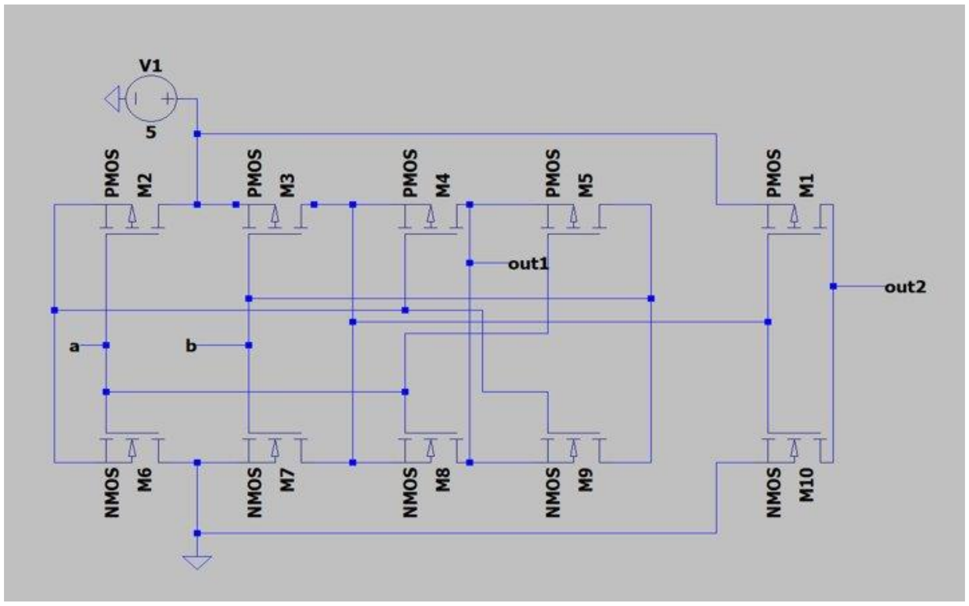


Figure 5.26: Reversible Xor gate

The transient analysis of power dissipation of Reversible Xor gate is shown in the graph below. Y axis in Figure 5.27 denotes power dissipation (in pW) and X axis denotes time (in seconds).

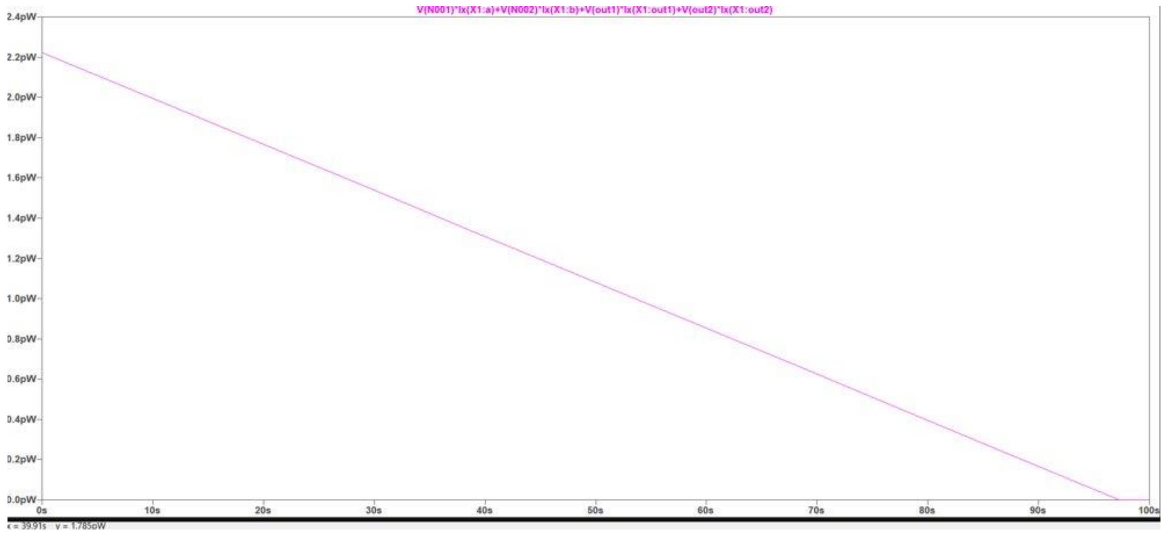


Figure 5.27: Power dissipation of reversible Xor

The schematic of reversible Xnor gate is shown in Figure 5.28 which is implemented in LTspice where twelve transistors are used, six PMOS and six NMOS. The inputs ports are given by a and b, and output ports are given by out1 and out2.

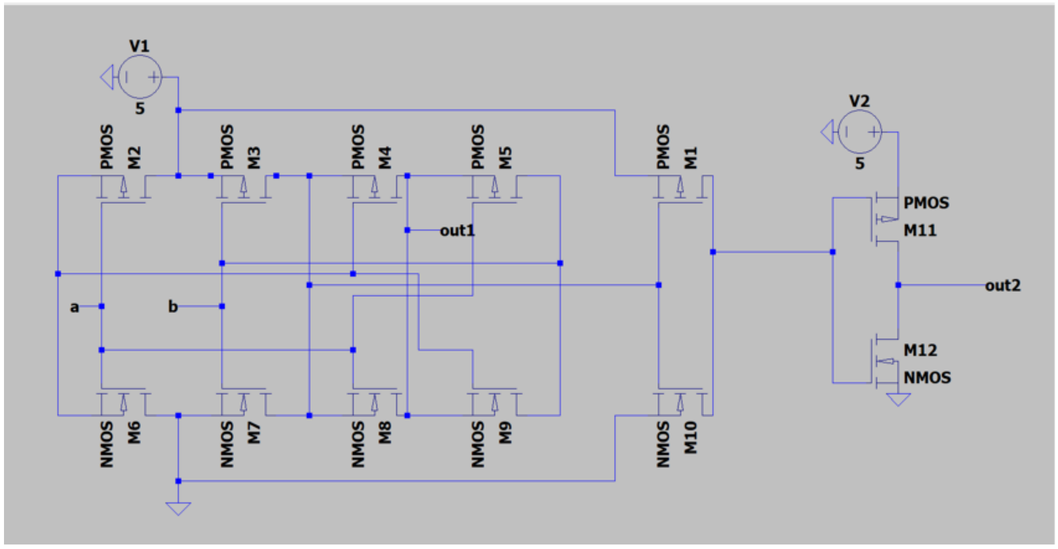


Figure 5.28: Schematic of Reversible XNOR

The transient analysis of power dissipation of Reversible Xnor gate is shown in the graph below. Y axis in Figure 5.29 denotes power dissipation (in fW) and X axis denotes time (in seconds).

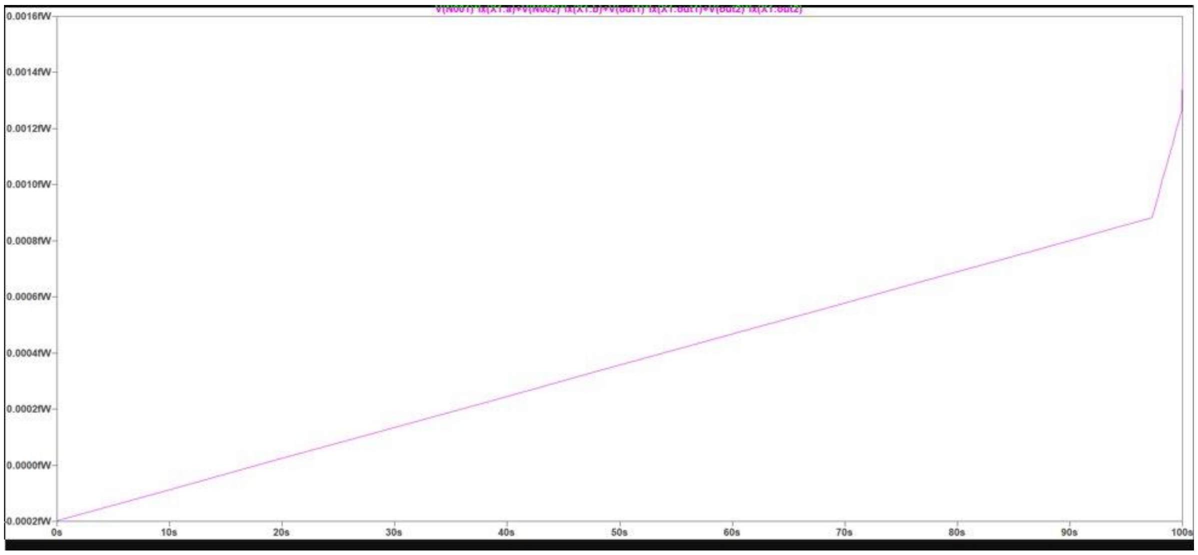


Figure 5.29: Power dissipation of Reversible XNOR

The schematic of irreversible Xor gate is shown in Figure 5.30 which is implemented in LTspice where eight transistors are used, four PMOS and four NMOS. The inputs ports are given by a and b, and output port is given by output.

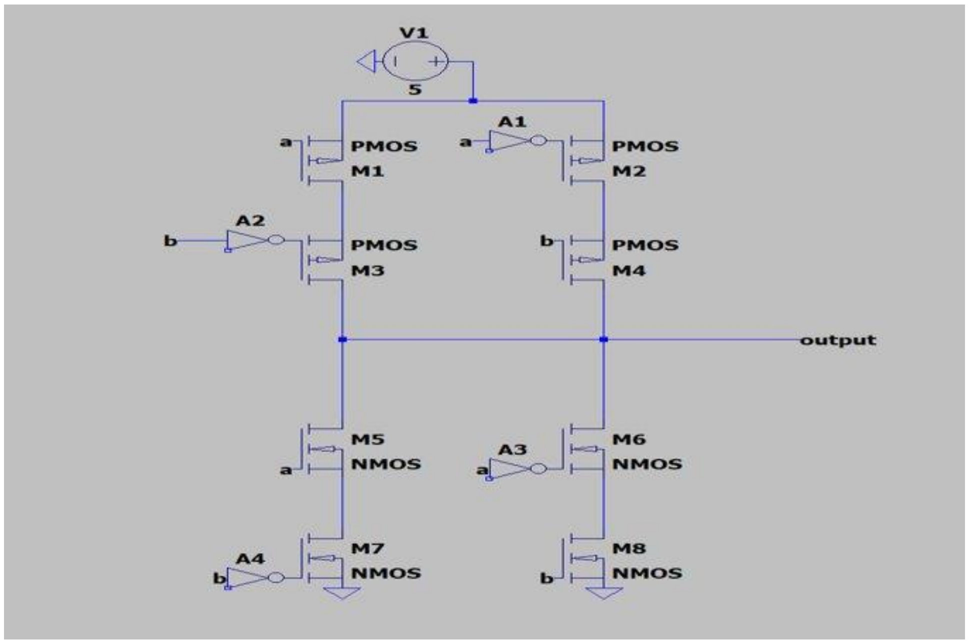


Figure 5.30: Schematic of XOR

The transient analysis of power dissipation of irreversible Xor gate is shown in the graph below. Y axis in Figure 5.31 denotes power dissipation (in micro-Watt) and X axis denotes time (in seconds).

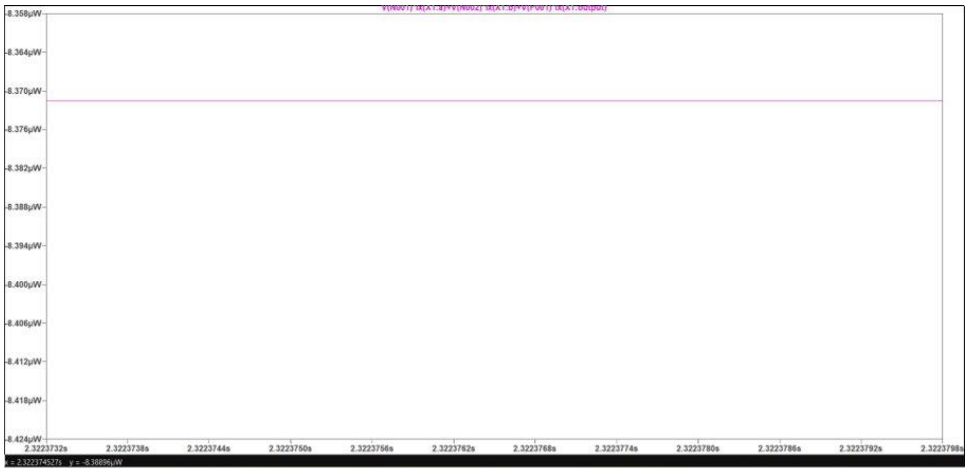
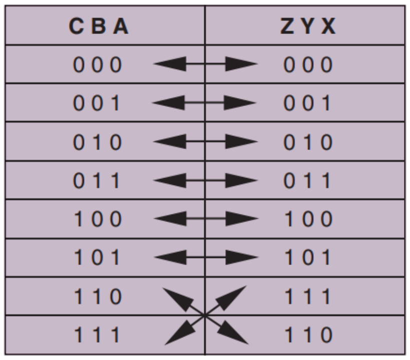


Figure 5.31: Power dissipation of XOR

6. RESULTS

1. Reversible gates are implemented using Verilog codes. Combinational and sequential circuits are also implemented using reversible logic gates.

2. The concept of reversible computing is proved which is shown in Fig 6.1 and Fig 6.2



C B A	Z Y X
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 1
1 1 1	1 1 0

Figure 6.1: Toffoli gate truth table

Fig 6.1 shows Toffoli gate truth table, the arrows represent the permutations of set of input vectors.

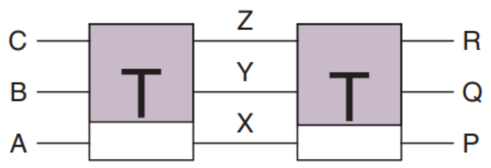
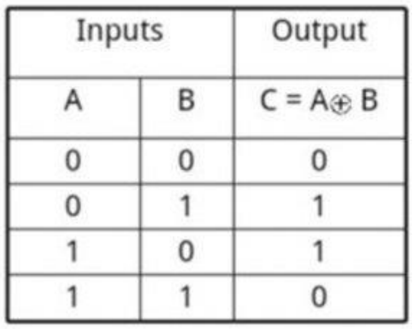


Figure 6.2 Two Toffoli gates Cascaded

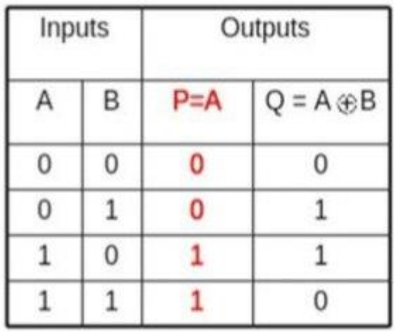
Fig 6.2 shows two identical Toffoli gates are cascaded. The outputs of the second gate are the same as the inputs to the first gate: R = C, Q = B, and P = A. As the inputs can be retrieved from the outputs, there is one-to-one mapping between inputs and outputs. Hence the concept of reversible computing is proved.

3. Reversible circuits are most widely used circuits because of their advantages over conventional circuits like:

1. In reversible computing, as inputs are uniquely mapped to outputs it is useful to rectify, if there are any errors



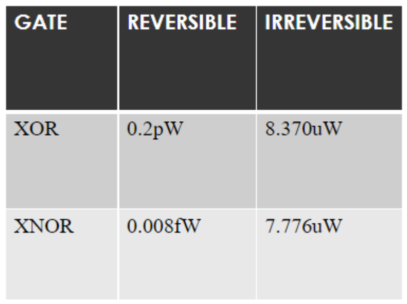
Inputs		Output
A	B	C = A⊕ B
0	0	0
0	1	1
1	0	1
1	1	0



Inputs		Outputs	
A	B	P=A	Q = A⊕B
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 6.1: Xor Truth Table **Table 6.2: Reversible Xor truth Table**

2. In reversible computing, as there is no information loss even after computation, this leads to a major advantage of less or no power dissipation.



GATE	REVERSIBLE	IRREVERSIBLE
XOR	0.2pW	8.370uW
XNOR	0.008fW	7.776uW

Table 6.3: Power Comparison

Table 6.3 shows that power dissipated by reversible gates is very less when compared to irreversible gates.

7. CONCLUSIONS AND FUTURE SCOPE

7.1 Conclusions:

1. In this project, different reversible logic circuits are designed using Verilog and simulated.
2. Reversible logic gates like Fredkin gate, Feynman gate, Toffoli gate, Peres gate, H.N.G. gate, T.S.G. gate, C.O.G. gate, S.R.G. gate, B.V.F. gate, B.J.N. gate, Double Feynman gate, DINV gate are designed using Verilog and simulated.
3. AND gate and OR gate are designed using reversible gates.
4. Combinational circuits like half adder using Peres gate, Full adder using H.N.G. gate, Half adder, Half subtractor and 1bit Comparator using B.V.F. and B.J.N. gates, 2x1 Mux, 4x1 Mux, 8x1 Mux and 1-bit ALU are designed and simulated.
5. Sequential circuits like S.R. Flipflop and D Flipflop are designed and simulated using reversible gates.
6. Schematic simulation is done for Reversible and conventional Xor and Xnor in LTspice.
7. From power dissipation results, shown in Table 6.28, its evident that power dissipation in reversible logic circuits is very less when compared with conventional circuits.

7.2 Future Scope:

In this project, different reversible logic circuits are designed and simulated. However, backend process (floor planning, placing, routing and physical verification) for all these gates can be done and further optimize the performance.