Investigation of Timing of Logic Gates realized using Probabilistic Spin Logic (PSL)

by

Ishan Bhatia

A thesis submitted in partial fulfillment for the
degree of Master of Technology

under supervision of
Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
Indraprastha Institute of Information Technology, Delhi

July 2019

Investigation of Timing of Logic Gates realized using Probabilistic Spin Logic (PSL)

by

Ishan Bhatia

A thesis submitted in partial fulfillment for the degree of Master of Technology

to

Indraprastha Institute of Information Technology, Delhi

July 2019

Certificate

This is to certify that the thesis titled "Investigation of Timing of Logic Gates realized using Probabilistic Spin Logic (PSL)" being submitted by Ishan Bhatia (Roll No.-MT17092) to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original work carried out by him under my supervision. In my opinion, thesis has reached the standards fulfilling the requirements of the regulations relating to the degree. The results contained in the thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Date: _____

Dr. Sneh Saurabh
Associate Professor
Department of Electronics and Communication Engineering
Indraprastha Institute of Information Technology, Delhi
New Delhi 110020

Abstract

Probabilistic Spin Logic (PSL) is a fascinating computing model composed of fundamental stochastic units called probabilistic bits (p-bits). A p-bit can be realized using low-barrier nanomagnets. It can be combined to implement Boolean functions with accuracy comparable to that of a traditional digital logic circuit. Moreover, a PSL circuit can also be used to compute the inverse of a Boolean function, a trait that is missing from conventional digital circuits. In this research, the application of PSL to realize complex Boolean functions has been examined using simulations. Further, an investigation into the timing of complex logic gates implemented using PSL has been carried out. A methodology to characterize delay as an intrinsic property of the logic gates independent of the way the delay is computed is proposed. The correctness of the proposed methodology using a few Boolean functions has been demonstrated in this work. Additionally, the factors that govern the delay of PSL circuits are examined and a technique to reduce the delay has been proposed.

Acknowledgements

My deepest appreciation goes to Dr. Sneh Saurabh for giving me the opportunity to work on my thesis under his guidance. It has been an honor for me. His advice, expertise and experience have been invaluable for my entire research work as well as my career.

This work is dedicated to my parents for their valuable teachings and lessons passed on to me over the years. I can never thank them enough for understanding and supporting me in every phase of my life. To my brother Pranjal Bhatia for all his support and friendship throughout these years.

Contents

List of Figures

List of Tables

Dedicated to my parents, teachers and friends...

Chapter 1

Introduction

The unrelenting march of technology defining the progress in computing hardware technology has been predominated by the conventional CMOS scaling and the Moores law until now. However, as the pace of innovation accelerates, and dimensional scaling reaches its limits, there exists an immediate need to find the next information processing hardware that can continue to fuel the technology revolution. Among the various alternatives that are being explored, technologies based on electronic spin (*spintronics*) or magnetic orientation of nanomagnets seem promising replacement. Spintronics, also known as spin electronics offers a scope to carry information in the intrinsic spin of the electron and its associated magnetic moment, in addition to the electronic charge possessed otherwise. Due to the fundamental quantum-mechanical properties (*spin*) of the electrons and their ability to store information, it finds its usage both in the fields of quantum computing and probabilistic computing. Probabilistic computing provides a technique to integrate probability and randomness into the basic building blocks of software and hardware. Consequently, the vast capacities of probabilistic computing can be exploited in solving complex optimization problems including quantum annealing and providing efficient hardware accelerators for binary stochastic neurons (BSN) used in machine learning [2].

Traditional semiconductor based logic built using stable, deterministic units such as MOS transistors or the memory devices constructed using nanomagnets with energy barriers in excess of $\approx 40 - 60\ kT$ are getting obsolete with time [1]. Therefore, alternatives such as nanomagnets with much lower energy barriers (unstable low-barrier nanomagnets) have been explored in implementing the probabilistic-bits or the p-bits. These can be used to generate functions such as random number generation (RNG) by sensing randomly fluctuating magnetization to provide a random time varying voltage [1]. In contrast to the classical bits used in digital computing which can take a

value of 0 or 1, p-bits are the entities that fluctuate rapidly between 0 and 1 at a rate determined by the height of the barrier of the nanomagnets. Thus, p-bits are the fundamental hardware building blocks for probabilistic logic. They represent unstable, stochastic units which when interconnected can create strong correlations to implement numerous Boolean functions with utmost precision comparable to that of the standard digital circuits. Moreover, the logic circuits implemented using probabilistic spin logic (PSL) exhibit the property of invertibility, incorporating features reminiscent of quantum circuits that are absent in digital circuits [1]. Additionally, p-bits are different from quantum-bits or q-bits used in quantum computing which are delicate superposition of both 0 and 1.

Any random signal generator whose randomness can be modulated using a third terminal could be a suitable building block for PSL or p-bit. A three-terminal building block with distinct read and write terminals marked as R and W in Fig, 1.1, has been used as a generic model for p-bit. [1, 3].

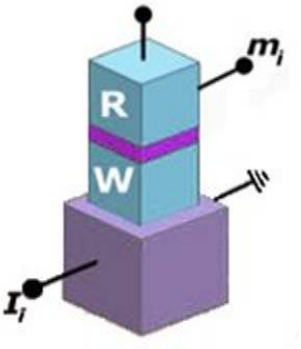


FIGURE 1.1: A generic representative model for a p-bit with distinct READ and WRITE units represented by R/W as described by equation 1.1 [1].

The output m_i depends on the input I_i according to the following equation:

$$m_i(t) = sgn(rand(-1, 1) + tanh(I_i(t))) \tag{1.1}$$

where $rand(-1, +1)$ denotes a uniformly distributed random number in the interval of $[-1, +1]$ and $m_i = \pm 1$ are the bipolar variables that represent the 0 and 1 states. Due to the unstable nature of the low barrier magnets (LBMs), they change their value after every τ seconds. The parameter τ refers to as the retention time of the p-bits. At each time step, in the absence of an input value (I_i is zero), the output hooks on to the value of -1 or +1 with equal probability, as shown in the Fig. 1.2(b) . A negative value of the input I_i makes negative values of the output m_i to be more likely, where as a positive value of the input I_i makes positive values of the output m_i to be more likely, as illustrated in Fig. 1.2(a) and Fig. 1.2(c), respectively.

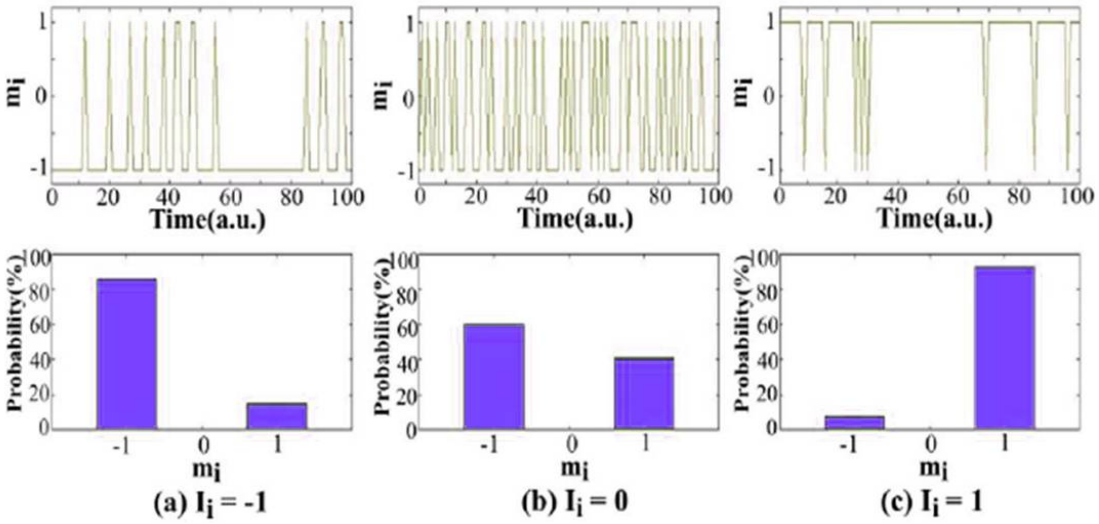


FIGURE 1.2: The idealized behavior of a p-bit is depicted at different bias points of $(-1, 0, +1)$ along with their corresponding distributions [1].

The shift in m_i as input I_i is ramped from negative to positive values has been illustrated in Fig. 1.3. The time-averaged value of m_i plotted in the above figure closely follows $tanh(I_i)$ curve and represents the essence of a p-bit.

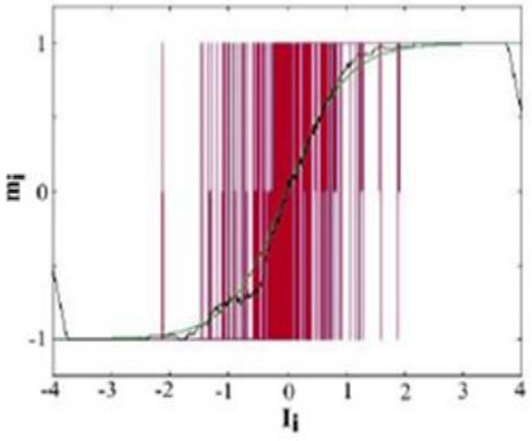


FIGURE 1.3: The red trace represents the "magnetization" (m_i) of nanomagnets as the current (I_i) is ramped as per equation 1.1. The green trace represents the sigmoid response of a RC circuit that provides a moving average of the time-dependent "magnetization" which agrees very well with tanh(I_i)

Although purely CMOS implementations of a p-bit is possible, the sigmoidal response seems to be a natural characteristic of spin current driven nanomagnets and thus can be exploited in PSL. Each p-bit acts like a tunable random number generator, having an intrinsic sigmoidal response for the time-averaged magnetization as a function of the spin current. An interconnection of these p-bits when allowed to fluctuate in a correlated manner yields useful functions similar to digital logic circuits. The interaction among

different p-bits and the influence of all those p-bits on a single p-bit i can be modeled using the following equation:

$$I_i(t) = I_0 * \{h_i(t) + \sum_j J_{ij} m_j(t)\} \tag{1.2}$$

where h_i provides a local bias to the i^{th} nanomagnet, J_{ij} defines the impact of the j^{th} bit on the i^{th} bit, and I_0 sets a global scale for the strength of interactions between the p-bits. The behavior modeled in Eq. (1.2) represents the weight logic which could be implemented in numerous ways such as memristors, floating-gate based devices, domain-wall based devices, standard CMOS etc. [2, 4–6]. It is the range of h/J values and sparsity of the h/J matrix that defines the suitability of these options.

1.1 Motivation and Objective

PSL is a fascinating alternative to the traditional digital circuits as it offers comparable accuracy in implementing the Boolean logic. Although a physical demonstration of a PSL-based digital circuit is still in progress, simulation based research on PSL highlights several distinct key features by demonstrating a 32-bit adder in regular and inverted mode operation implemented as a hybrid Boltzmann Machine [1].

The h_i/J_{ij} coefficients (or the h/J-matrices in the matrix notation) introduced in Eq. (1.2) play a crucial role in determining the functionality of a PSL circuit. It is interesting to note that for a given functionality, h/J matrices are not unique. As different combination of logic gates can be used to implement a particular Boolean logic, similarly different sets of h/J matrices can still yield the same functionality. However, different implementations of a Boolean function in PSL can have different characteristics such as timing (delay), area (number of p-bits and their interconnections) and accuracy (error measure). However, currently there is no method available in literature to characterize delay in a PSL circuit. Therefore, a methodology to quantify the delay of logic gates realized using PSL implementation has been proposed in this work. Additionally, currently it is not discussed in the literature about various factors that govern the delay. Therefore, in this work different factors that affect delay in a PSL circuit are examined and modeled. Moreover, a technique to reduce the delay of a PSL circuit has also been deduced. This work can be treated as a step towards automatic timing analysis and synthesis of PSL circuits.

1.2 Scope of thesis

In this research, after obtaining one of the possible h and J matrices for an arbitrary Boolean function, the system is simulated using the universal model approach defined in [1]. All the simulations in this research have been performed on MATLAB. The probabilistic nature of the p-bits is emulated in simulations using the random number generator (RNM) of MATLAB. The weights used in the h and J matrices define the strength of interconnections among p-bits.

1.3 Structure of the thesis

The thesis is divided into 6 chapters as follows.

- **Chapter 2: Background Work**
  This chapter discusses the background work that has already been established in construction of basic gates along with combination of those basic gates to form other logic gates using PSL.
- **Chapter 3: Realization of complex Boolean Function using PSL**
  This chapter illustrates realization of complex gates (combinational logic circuits) using basic and universal gates in PSL and explores the trade-off between different implementations of those circuits in terms of accuracy and the resources used. This chapter also highlights the difficulties faced in realization of a sequential circuit using PSL.
- **Chapter 4: Quantifying Delay in PSL circuits**
  This chapter explains different methodologies to compute and quantify delay of any arbitrary Boolean function implemented using PSL.
- **Chapter 5: Modeling Delay in PSL circuits**
  This chapter investigates into different factors that govern the delay of different logic gates implemented using PSL. Further it provides a timing model to support the observed behavior of delay in PSL circuits.
- **Chapter 6: Conclusions**
  This chapter concludes all the findings and contributions of this work and possible future extension of this work.

Chapter 2

Background work

To illustrate the interconnection among p-bits, several approaches such as Ising Hamiltonian for quantum computers [7], principles developed for Hopfield networks [8], etc. have been presented in the literature. Ising model being the more favored approach defines discrete binary variables that represent the magnetic dipole moment of atomic spins or the p-bits (in this research) that switch between the two possible Ising states (+1 *or* 1). The Ising Hamiltonian for a Boolean network can be defined as [9],

$$H_{Ising} = \sum_{i} h_i m_i + \sum_{<i,j>} J_{ij} m_i m_j \tag{2.1}$$

where J_{ij} represents the exchange interaction strength between two neighboring p-bits m_i and m_j while h_i represents the bias provided to the p-bit m_i. The term h_i is composed of two parts, a local bias and a fix bias as per the following equation:

$$h_i = h_{local-bias} + h_{fix-bias} \tag{2.2}$$

where $h_{local-bias}$ is the constant local bias internal to the Boolean network, while $h_{fix-bias}$ refers to the fixed bias that acts as an input from the user on the input/output terminals of Boolean network. Depending on the values taken by these p-bits, the network can have many possible outcomes which refer to as the possible states of the system. These states fluctuate in a correlated manner until the correct state is found to produce functionality similar to a digital logic circuit with comparable accuracy in general.

FIGURE 2.1: The interaction among individual p-bits is represented using distinct read and write units. Each p-bit m_i in influenced by a local bias(h_i) and some weight(J_{ij}) which couples the effect of all the neighbouring p-bits m_j's on p-bit m_i. [1]

2.1 Basic Logic Gates

2.1.1 NOT Gate

A NOT gate consists of an input (A) and an output (Y) represented using 2 p-bits m_1 and m_2 respectively. The Ising Hamiltonian for a NOT gate ($m_2 = \neg m_1$) is given as follows [9]:

$$H_{NOT} = 1 - m_1 m_2 \tag{2.3}$$

The absence of independent terms consisting of m_1 and m_2 in the above Hamiltonian equation implies the h-matrix for a NOT gate to be a zero column matrix. A negative value of $m_1.m_2$ implies the interaction strength between the two p-bits as -1. On comparing Eq. (2.3) with Eq. (2.1), the h and J-matrices for a NOT gate are defined by:

$$h = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad J = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \tag{2.4}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 2.2(a). The simulation results for the same shown in Fig. 2.2(b) demonstrate the correct functionality of a NOT gate.

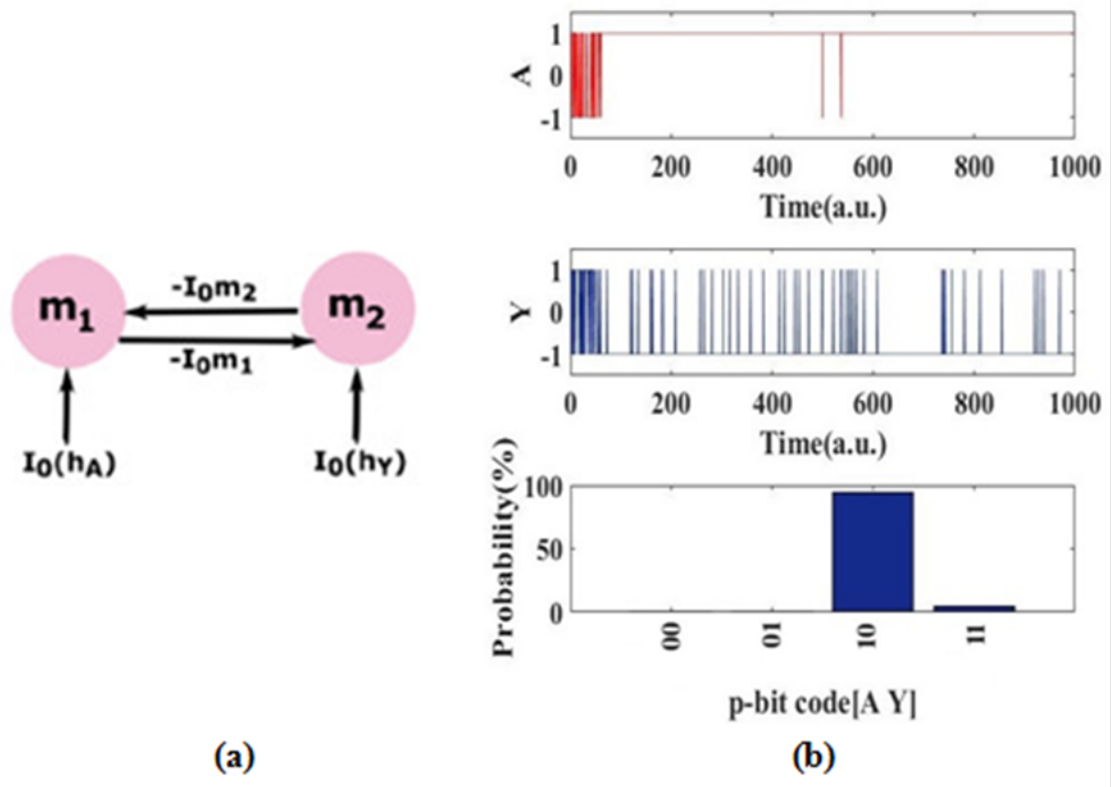


FIGURE 2.2: **NOT gate:** (a) Implementation of NOT gate using h/J described by Eq. (2.4) and representation of the PSL circuit as per Eq. (1.2) and Eq. (2.3). The fixed external bias is represented by h_A and h_Y. (b) Simulation results of the NOT gate in which A is clamped to logic 1, while Y is left floating. It is found that Y stays in logic 0-state for most of the time after I_0 is increased from 0 to 1.5. The histogram plotted has been obtained by averaging over 1000 time samples.

2.1.2 Two-input AND Gate

A 2-input AND gate consists of inputs (A, B) and an output (Y) represented using 3 p-bits m_1, m_2 and m_3 respectively. The Ising Hamiltonian for an AND gate ($m_3 = m_1 \wedge m_2$) is given as follows [9]:

$$H_{AND} = 3 - (m_1 + m_2 - 2m_3) - (-m_1m_2 + 2m_2m_3 + 2m_1m_3) \tag{2.5}$$

On comparing Eq. (2.5) with Eq. (2.1), the h and J-matrices for a 2-input AND gate are defined by [7]:

$$h_{AND} = \begin{bmatrix} +1 \\ +1 \\ -2 \end{bmatrix} \qquad J_{AND} = \begin{bmatrix} 0 & -1 & 2 \\ -1 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \tag{2.6}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 2.3(a). The simulation results for the same shown in Fig.

2.3(b) demonstrate the correct functionality of a 2-input AND gate.

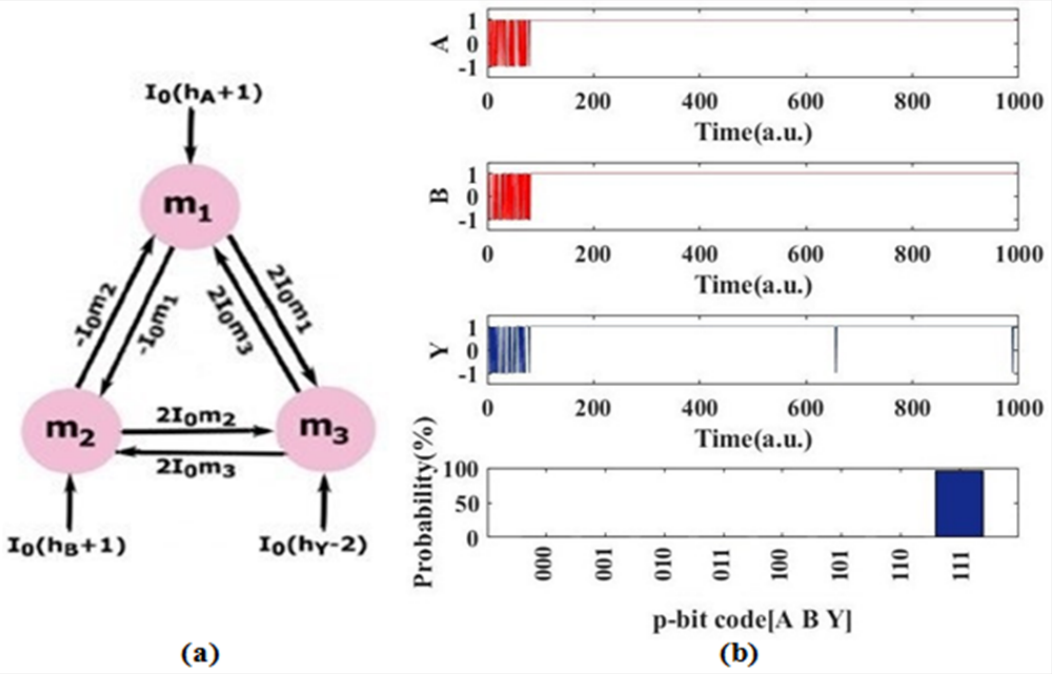


FIGURE 2.3: **AND gate:** (a) Implementation of AND gate using h/J described by Eq. (2.6) and representation of the PSL circuit as per Eq. (1.2) and Eq. (2.5). The fixed external bias is represented by h_A, h_B and h_Y. (b) Simulation results of the AND gate where A and B are clamped to logic 1, while Y is left floating. It is found that Y stays in logic 1-state for most of the time after I_0 is increased from 0 to 1.5. The histogram plotted has been obtained by averaging over 1000 time samples.

2.1.3 Two-input OR Gate

A 2-input OR gate consists of inputs (A, B) and an output (Y) represented using 3 p-bits m_1, m_2 and m_3 respectively. The Ising Hamiltonian for an OR gate ($m_3 = m_1 \vee m_2$) is given as follows [9]:

$$H_{OR} = 3 - (-m_1 - m_2 + 2m_3) - (-m_1m_2 + 2m_2m_3 + 2m_1m_3) \tag{2.7}$$

On comparing Eq. (2.7) with Eq. (2.1), the h and J-matrices for a 2-input OR gate are defined by [7]:

$$h_{AND} = \begin{bmatrix} -1 \\ -1 \\ +2 \end{bmatrix} \qquad J_{AND} = \begin{bmatrix} 0 & -1 & 2 \\ -1 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \tag{2.8}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 2.4(a). The simulation results for the same shown in Fig. 2.4(b) demonstrate the correct functionality of a 2-input OR gate. It is interesting to

note that the J-matrix of an OR gate is identical to that of AND gate. Its the h-matrix of OR gate that has an opposite polarity to that of the AND gate implying that the interaction strength among p-bits is same in both the logic gates where as it is the difference in the internal local bias (h_i) given to each p-bit that defines the corresponding functionality.

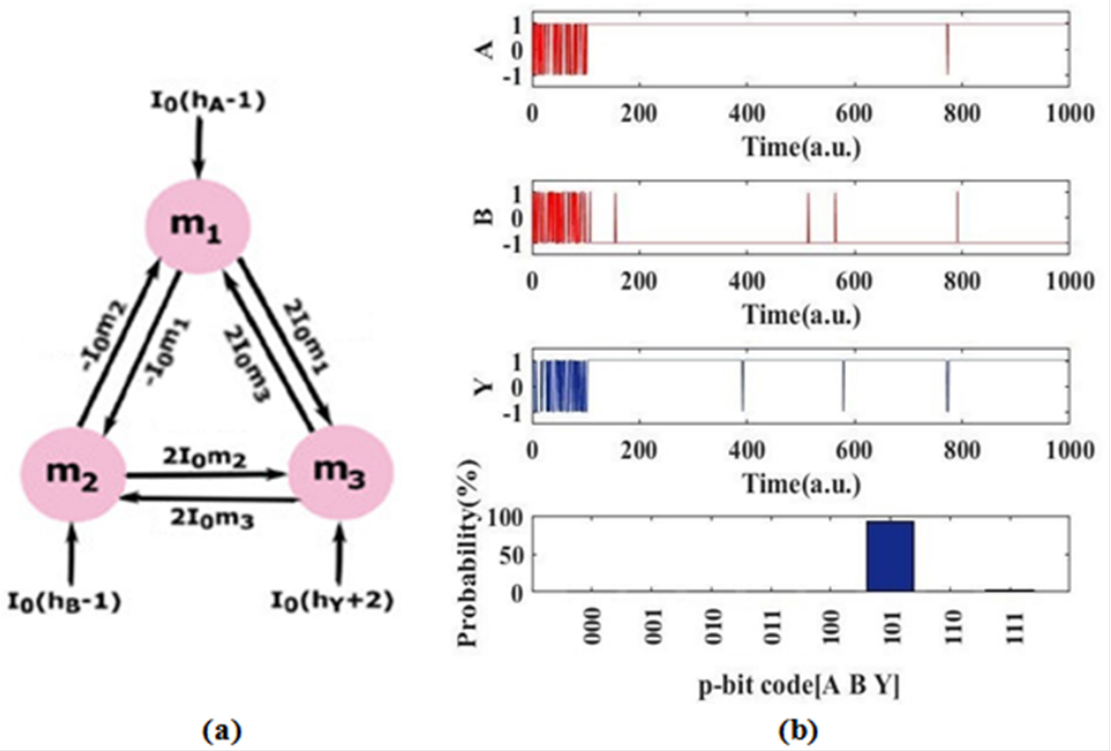


FIGURE 2.4: **OR gate:** (a) Implementation of OR gate using h/J described by Eq. (2.8) and representation of the PSL circuit as per Eq. (1.2) and Eq. (2.7). The fixed external bias is represented by h_A, h_B and h_Y. (b) Simulation results of the OR gate where A and B are clamped to logic 1 and 0 respectively, while Y is left floating. It is found that Y stays in logic 1-state for most of the time after I_0 is increased from 0 to 1.5. The histogram plotted has been obtained by averaging over 1000 time samples.

2.2 Combination of Basic Logic Gates

The basic logic gates discussed above can be combined to synthesize more complex gates such as the XOR gate or the NAND gate and their implementation using PSL is discussed below:

2.2.1 Two-input XOR Gate realized using basic gates

A two input XOR gate can be realized in terms of basic AND/OR/NOT gates as $Y = (A \wedge \bar{B}) \vee (\bar{A} \wedge B)$, where A, B are the inputs while Y is the output. In PSL, a two-input

XOR gate realized using basic gates can be implemented with the help of 7 p-bits and their interconnection is shown in Fig. 2.5(a). P-bits m_1 and m_2 are connected to the inputs A and B; p-bit m_7 is connected to the output Y while the remaining four p-bits are internal p-bits of the system under realization.

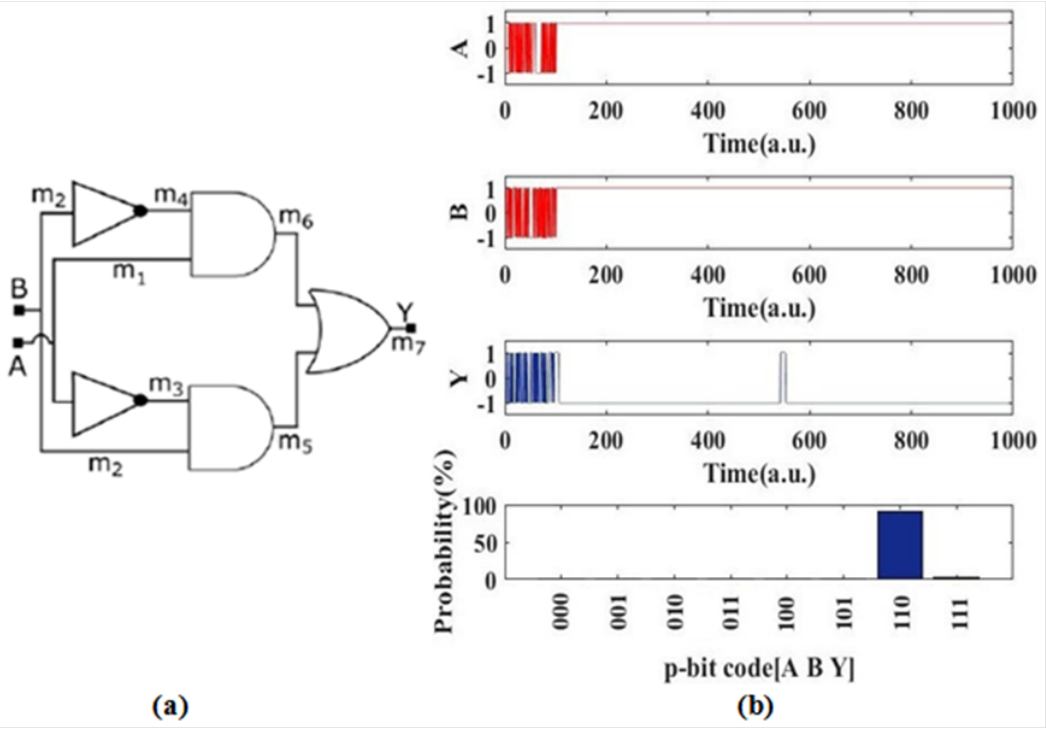


FIGURE 2.5: **XOR gate:** (a) Implementation of a two-input XOR gate using h/J described by Eq. (2.9) using PSL. (b) Simulation results of the XOR gate where A and B both are clamped to logic 1, while Y is left floating. It is found that Y stays in logic 0-state with quite a high probability. The histogram plotted has been obtained by averaging over 1000 time samples.

Depending on the number of p-bits used (7 in this case); the h matrix will be of dimensions 7×1 while the J matrix will be of dimensions 7×7. In order to determine the exact h and J matrices from the implementation shown in Fig. 2.5(a), following convention has been followed: the i^{th} row of the h matrix refers to the local bias provided to the i^{th} p-bit while the interaction between the i^{th} p-bit and the j^{th} pbit is captured as the $(i, j)^{th}$ element in the J matrix. Using the h and J matrices of AND/OR/NOT gates as reported above, the h and J matrix for a XOR gate can be determined. For example, the p-bits m_1 and m_4 are connected to the p-bit m_6 by the means of an AND gate. Therefore, the 1^{st}, 4^{th} and 6^{th} rows of h matrix and the 1^{st}, 4^{th} and 6^{th} row/column of the J matrix corresponds to the p-bits m_1, m_4 and m_6 respectively and contain elements as that of an AND gate derived from Eq.2.6. It is crucial to note that the p-bit m_6 serves as both the output of an AND gate as well as the input to an OR gate, and hence according to Eq.2.1 , the h and J matrices for a two-input XOR gate can be defined as follows:

$$h_{XOR-Basic-Gates} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ -3 \\ -3 \\ 2 \end{bmatrix} \quad J_{XOR-Basic-Gates} = \begin{bmatrix} 0 & 0 & -1 & -1 & 0 & 2 & 0 \\ 0 & 0 & -1 & -1 & 2 & 0 & 0 \\ -1 & -1 & 0 & 0 & 2 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 2 & 0 & 0 & -1 & 2 \\ 2 & 0 & 0 & 2 & -1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 0 \end{bmatrix} \tag{2.9}$$

2.2.2 Two-input NAND Gate

A two input NAND gate can be realized in terms of AND and NOT gates as $Y = A \bar{\wedge} B$, where A, B are the inputs while Y is the output. In PSL, a two-input NAND gate realized using basic gates can be implemented with the help of 4 p-bits and their interconnection is shown in Fig. 2.6(a).

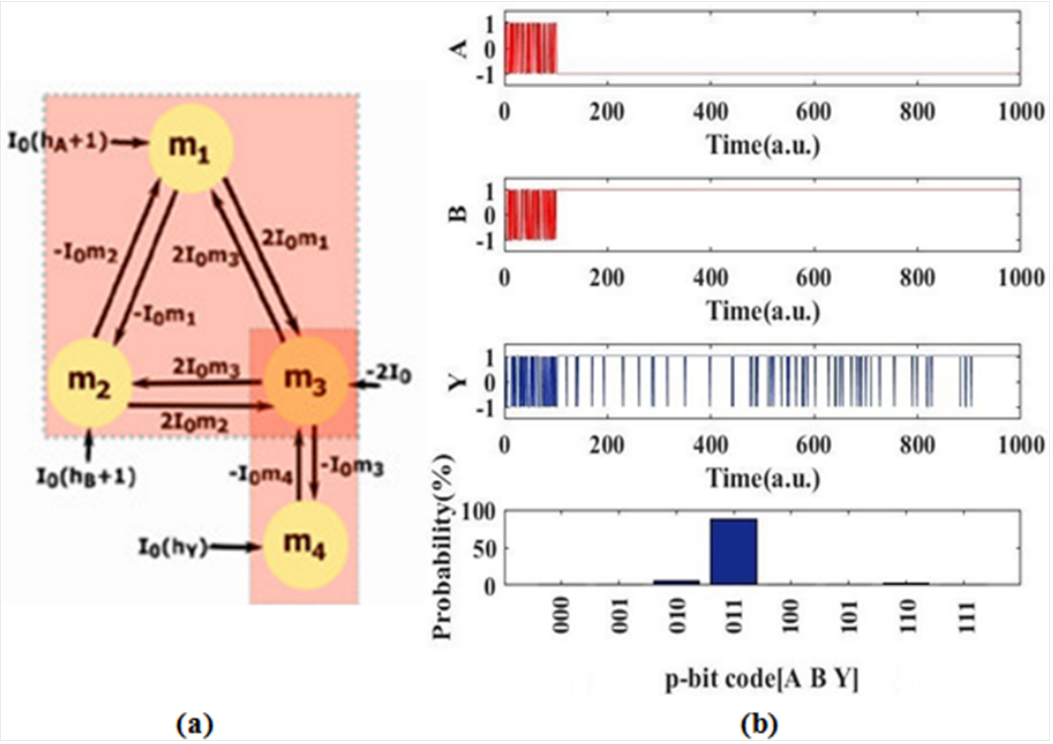


FIGURE 2.6: **NAND gate:** (a) Implementation of a two-input NAND gate using h/J described by Eq. (2.10) using the basic gates in PSL. (b) Simulation results of the NAND gate where A and B are clamped to logic 0 and 1 respectively, while Y is left floating. It is found that Y stays in logic 1-state with a high probability. The histogram plotted has been obtained by averaging over 1000 time samples.

P-bits m_1 and m_2 are connected to the inputs A and B; p-bit m_4 is connected to the output Y while the remaining p-bit m_3 is the internal p-bit. As observed, p-bit m_3 is common to both the output of AND gate and the input to NOT gate, hence according to Eq.2.1, the h and J matrices for a two-input NAND gate are defined as follows:

$$h = \begin{bmatrix} +1 \\ +1 \\ -2 \\ 0 \end{bmatrix} \qquad J = \begin{bmatrix} 0 & -1 & 2 & 0 \\ -1 & 0 & 2 & 0 \\ 2 & 2 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2.10}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 2.6(a). The simulation results for the same shown in Fig. 2.6(b) demonstrate the correct functionality of a 2-input NAND gate realized using the basic gates.

Chapter 3

Realization of complex Boolean logic using PSL

3.1 Universal Logic Gates

3.1.1 Two-input NAND Gate

A 2-input NAND gate consists of inputs (A, B) and an output (Y) represented using 3 p-bits m_1, m_2 and m_3 respectively. On contrary to the four p-bits used earlier to realize the NAND gate in terms of basic gates, the same could be realized in three p-bits using the Ising Hamiltonian for a NAND gate ($Y = A \bar{\wedge} B$) which is given as follows:

$$H_{NAND} = 3 - (m_1 + m_2 + 2m_3) - (-m_1m_2 - 2m_2m_3 - 2m_1m_3) \tag{3.1}$$

On comparing Eq. (3.1) with Eq. (2.1), the h and J-matrices for a 2-input NAND gate are defined by:

$$h_{NAND} = \begin{bmatrix} +1 \\ +1 \\ +2 \end{bmatrix} \qquad J_{NAND} = \begin{bmatrix} 0 & -1 & -2 \\ -1 & 0 & -2 \\ -2 & -2 & 0 \end{bmatrix} \tag{3.2}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.1(a). The simulation results for the same shown in Fig. 3.1(b) demonstrate the correct functionality of a 2-input NAND gate.

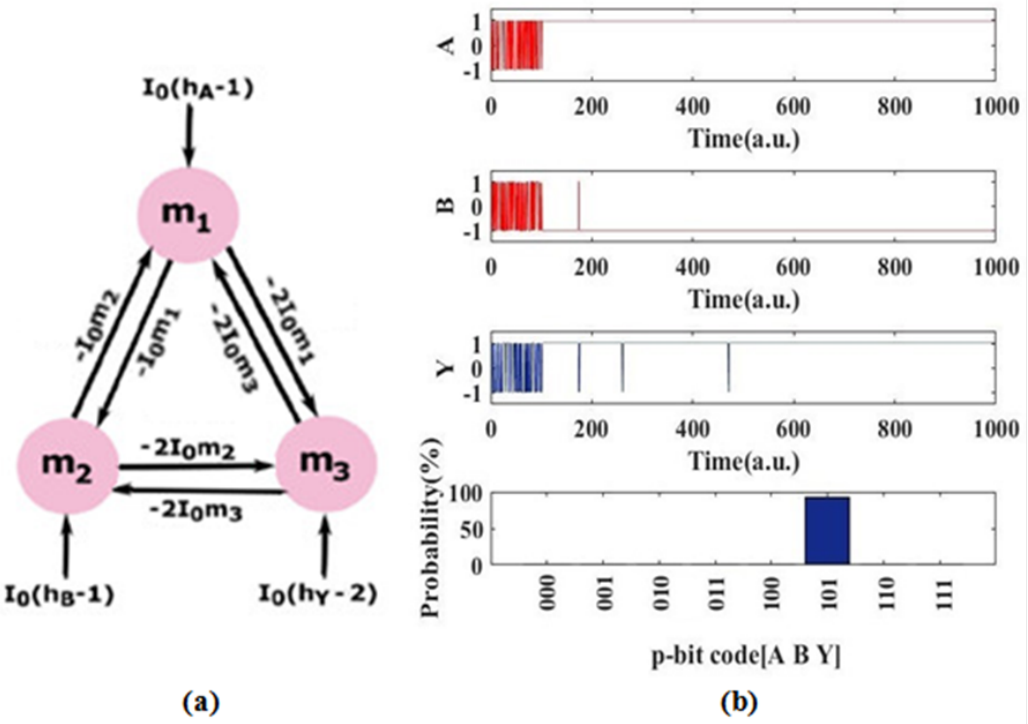


FIGURE 3.1: **NAND gate:** (a) Implementation of NAND gate using h/J described by Eq. (3.2) and representation of the PSL circuit as per Eq. (1.2) and Eq. (3.1). The fixed external bias is represented by $h_A - 1$, $h_B - 1$ and $h_Y - 2$. (b) Simulation results of the NAND gate where A and B are clamped to logic 1 and 0, respectively, while Y is left floating. It is found that Y stays in logic 1-state with a high probability matching the expected behaviour thus, verifying the Ising Hamiltonian equation 3.1 assumed. The histogram plotted has been obtained by averaging over 1000 time samples.

3.1.2 Two-input NOR Gate

A 2-input NOR gate consists of inputs (A, B) and an output (Y) represented using 3 p-bits m_1, m_2 and m_3 respectively. The Ising Hamiltonian for a NOR gate ($Y = A \bar{\vee} B$) is given as follows:

$$H_{NOR} = 3 - (-m_1 - m_2 - 2m_3) - (-m_1m_2 - 2m_2m_3 - 2m_1m_3) \tag{3.3}$$

On comparing Eq. (3.3) with Eq. (2.1), the h and J-matrices for a 2-input NOR gate are defined by:

$$h_{NOR} = \begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix} \qquad J_{NOR} = \begin{bmatrix} 0 & -1 & -2 \\ -1 & 0 & -2 \\ -2 & -2 & 0 \end{bmatrix} \tag{3.4}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.2(a). The simulation results for the same shown in Fig. 3.2(b) demonstrate the correct functionality of a 2-input NOR gate. It is interesting to note that the J-matrix of a NOR gate is identical to that of NAND gate. It is the h-matrix of NOR gate that has an opposite polarity to that of the NAND gate implying that the interaction strength among p-bits is same in both universal logic gates where as it is the difference in the internal local bias (h_i) given to each p-bit that defines the corresponding functionality.

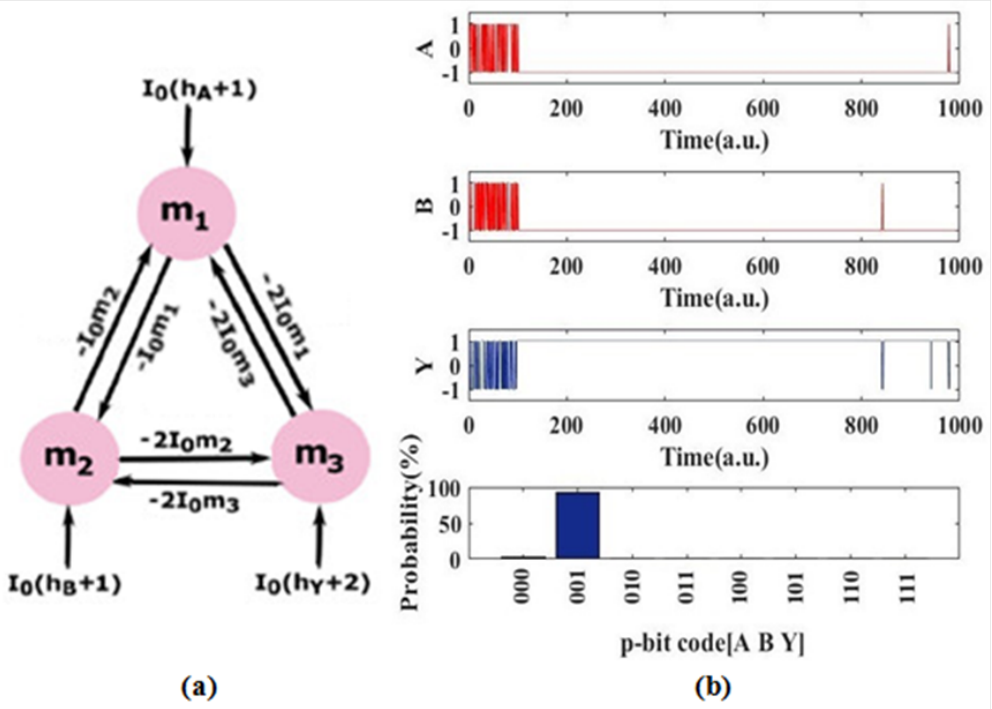


FIGURE 3.2: **NOR gate:** (a) Implementation of NOR gate using h/J described by Eq. (3.4) and representation of the PSL circuit as per Eq. (1.2) and Eq. (3.3). The fixed external bias is represented by $h_A + 1$, $h_B + 1$ and $h_Y + 2$. (b) Simulation results of the NOR gate where both A and B are clamped to logic 0, while Y is left floating. It is found that Y stays in logic 1-state with a high probability matching the expected behaviour thus, verifying the Ising Hamiltonian equation 3.3 assumed. The histogram plotted has been obtained by averaging over 1000 time samples.

3.2 Complex Logic Gates

Complex Boolean logic gates can be realized either as function of basic gates such as the NOT($\neg$), AND($\wedge$) ,and OR($\vee$) gates or those could be a logical function of the universal gates such as NAND or NOR gates. Different complex logic circuits realized using basic and universal gates and their implementation in PSL has been discussed below.

3.2.1 Two-input XOR Gate realized using NAND Gates

A two input XOR gate can be realized in terms of NAND gates as $Y = \overline{\overline{(A \wedge \overline{B})} \wedge \overline{(\overline{A} \wedge B)}}$, where A, B are the inputs while Y is the output. In PSL, a two-input XOR synthesized using only NAND gates can be implemented with the help of 6 p-bits on contrary to the 7-pbits used while realizing it using basic gates, and their interconnection is shown in Fig. 3.3(a).

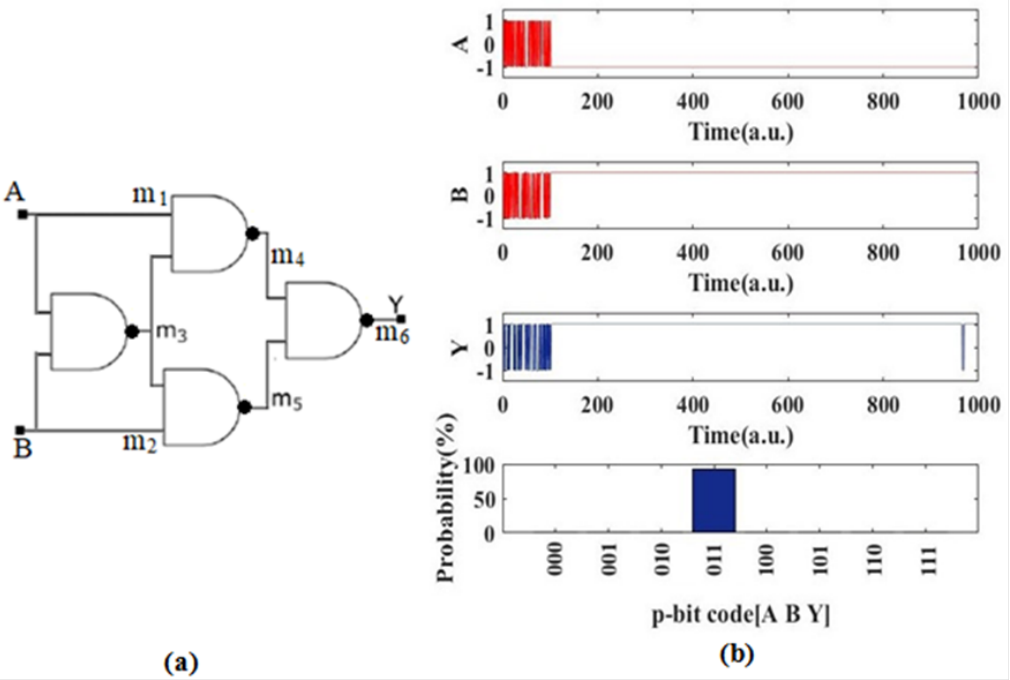


FIGURE 3.3: **XOR gate:** (a) Implementation of a two-input XOR gate using universal NAND gates. (b) Simulation results of the two-input XOR gate where A and B are clamped to logic 0 and 1 respectively, while Y is left floating. It is found that Y stays in logic 1-state with a high probability. The histogram plotted has been obtained by averaging over 1000 time samples.

P-bits m_1 and m_2 are connected to the input; p-bit m_6 is connected to the output while the remaining three p-bits are the internal p-bits to the system under realization. Depending on the number of p-bits used (6 in this case); the h matrix will be of dimensions 6×1 while the J matrix would be of dimensions 6×6. Using the h and J matrices of NAND gates derived from Eq. 3.2, the h and J matrices for a two-input XOR gate realized using NAND gates can be defined as follows:

$$h_{XOR2} = \begin{bmatrix} 2 \\ 2 \\ 4 \\ 3 \\ 3 \\ 2 \end{bmatrix} \qquad J_{XOR2} = \begin{bmatrix} 0 & -1 & -3 & -2 & 0 & 0 \\ -1 & 0 & -3 & 0 & -2 & 0 \\ -3 & -3 & 0 & -2 & -2 & 0 \\ -2 & 0 & -2 & 0 & -1 & -2 \\ 0 & -2 & -2 & -1 & 0 & -2 \\ 0 & 0 & 0 & -2 & -2 & 0 \end{bmatrix} \tag{3.5}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.3(a). The simulation results for the same shown in Fig. 3.3(b) demonstrate the correct functionality of a 2-input XOR gate as when A and B are fixed to logic 0 and 1 respectively, and all the other p-bits are allowed to fluctuate in a correlated manner, the output p-bit settles to logic 1 thus, only the possible valid states of the XOR gate are visited with high probability.

3.2.2 Three-input XOR Gate/ Full Adder realized using Basic Gates

A full adder is a logical circuit that performs addition operation on three one-bit binary inputs and generates sum and carry as the output. The sum generated can be represented as the XOR operation of the three binary inputs. Thus a three input XOR gate ($A \oplus B \oplus C$) can be realized in terms of basic gates as $Y = (\bar{A} \wedge \bar{B} \wedge C) \vee (\bar{A} \wedge B \wedge \bar{C}) \vee (A \wedge \bar{B} \wedge \bar{C}) \vee (A \wedge B \wedge C)$, where A, B, C are the inputs while Y is the output. In PSL, a three-input XOR synthesized using basic gates can be implemented with the help of 13 p-bits and their interconnection is shown in Fig. 3.4(a). P-bits m_1, m_2 and m_3 are connected to the three inputs A, B, C respectively; p-bit m_{13} is connected to the output while the remaining *nine* p-bits are the internal p-bits of the system. As the PSL circuit for a three input XOR gate constitutes of *thirteen* p-bits, the h matrix will be of dimensions 13×1 while the J matrix would be of the dimensions 13×13. Following the same methodology to assign weights as done for the two-input XOR gate earlier, and according to Eq. (2.1) , h and J matrices for a 3-input XOR gate realized using basic gates can be defined as follows:

$$h_{XOR3-Basic} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & -3 & -3 & 3 & -3 & -3 & 2 \end{bmatrix}^T$$

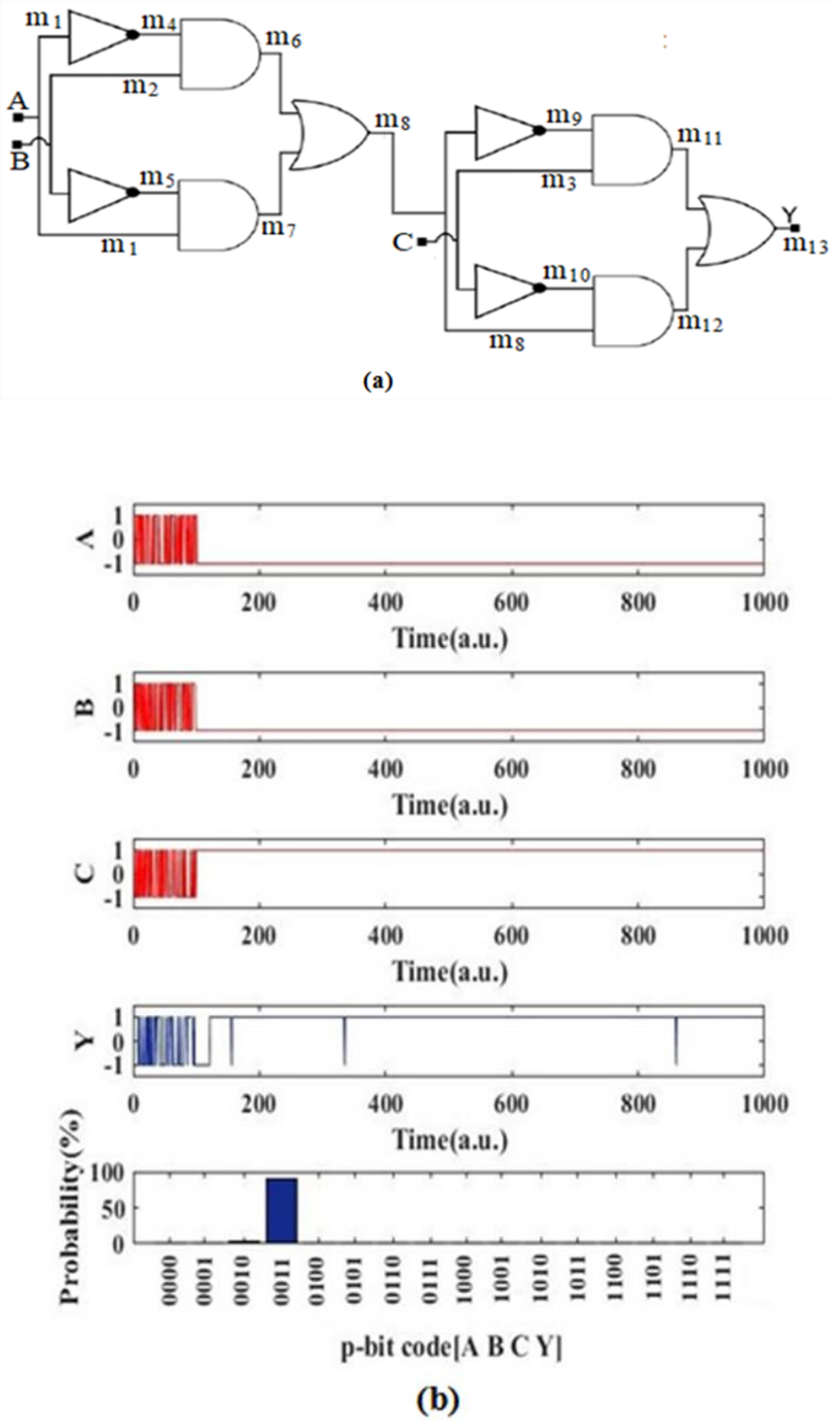


FIGURE 3.4: **3-input XOR gate realized using Basic Gates:** (a) Implementation of a three-input XOR gate using basic gates. (b) Simulation results of the three-input XOR gate where A, B and C are clamped to logic 0, 0 and 1 respectively, while Y is left floating. It is found that Y stays in logic 1-state with a high probability. The histogram plotted has been obtained by averaging over 1000 time samples.

$$J_{XOR3-Basic} = \begin{bmatrix} 0 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 2 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 2 & 0 & 0 & -1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & -1 & 2 & 2 & 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & -1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 \end{bmatrix} \quad (3.6)$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.4(a). The simulation results for the same shown in Fig. 3.4(b) demonstrate the correct functionality of a 3-input XOR gate. When the three input p-bits are biased to a value of 0, 0 and 1 and all the other p-bits are allowed to float, only the possible valid state of the XOR gate with the correct output of 1 for the corresponding set of inputs that is the 0011 state is visited with high probability.

3.2.3 Three-input XOR Gate realized using NAND Gates

A three input XOR gate can be realized in terms of universal (NAND) gates as $Y = \overline{(\overline{A} \wedge \overline{B} \wedge C)} \wedge \overline{(\overline{A} \wedge B \wedge \overline{C})} \wedge \overline{(A \wedge \overline{B} \wedge \overline{C})} \wedge \overline{(A \wedge B \wedge C)}$, where A, B, C are the inputs while Y is the output. In PSL, a three-input XOR synthesized using NAND gates can be implemented with the help of 11 p-bits on contrary to the 13-pbits used while realizing it using basic gates and their interconnection is shown in Fig. 3.5(a). P-bits m_1, m_2 and m_3 are connected to the three inputs A, B, C respectively; p-bit m_{11} is connected to the output while the remaining *seven* p-bits are internal p-bits of the system under realization. As the PSL circuit for a three input XOR gate constitutes of *eleven* p-bits, the h matrix will of dimensions 11×1 while the J matrix would be of the dimensions 11×11. Following the same methodology to assign weights as done for the two-input XOR gate earlier, and according to Eq. (2.1), h and J matrices for a 3-input XOR gate realized using NAND gates can be defined as follows:

$$h_{XOR3-NAND} = \begin{bmatrix} 2 & 2 & 4 & 3 & 3 & 4 & 4 & 3 & 3 & 2 \end{bmatrix}^T$$

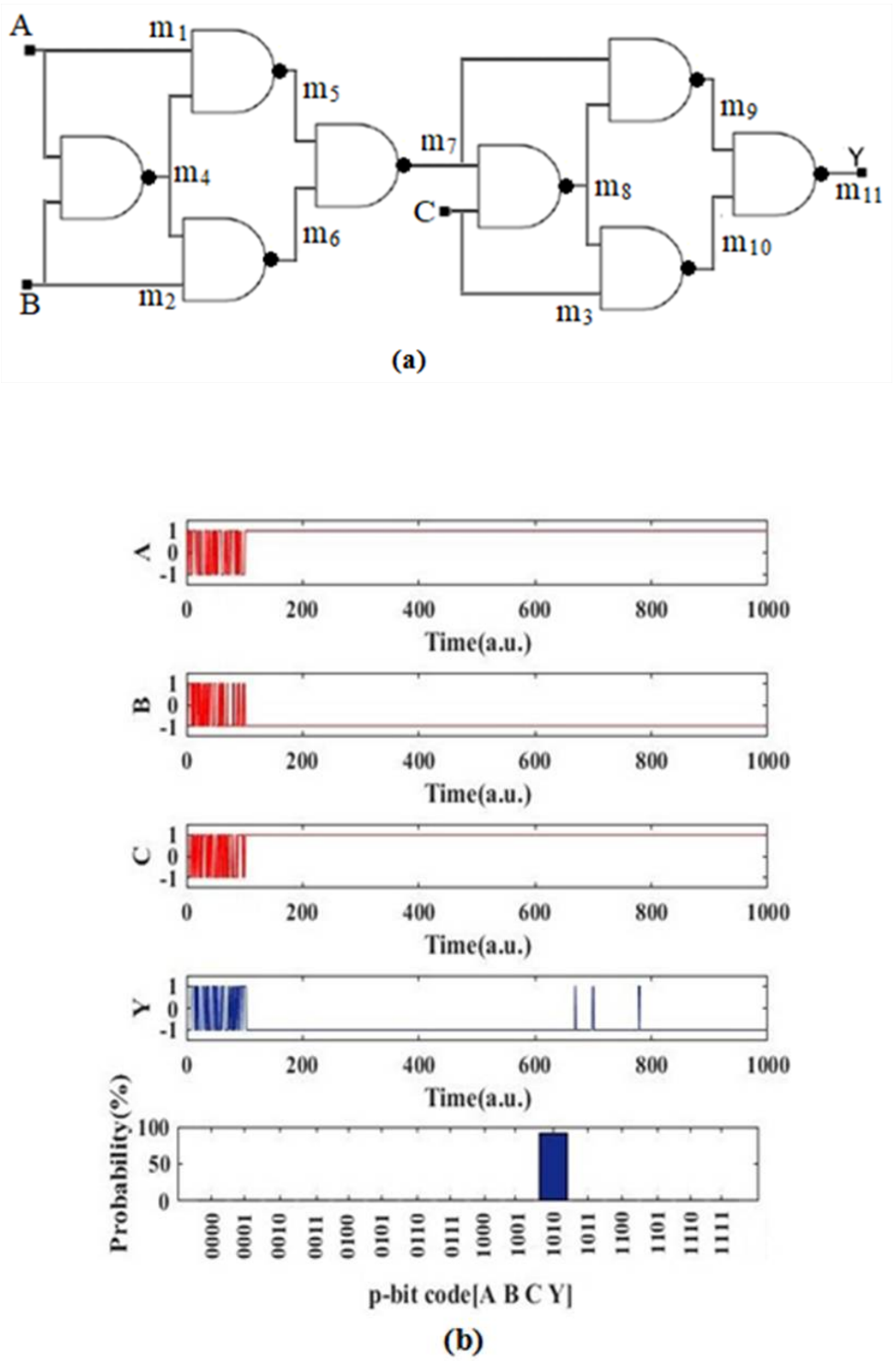


FIGURE 3.5: **3-input XOR gate realized using NAND Gates:** (a) Implementation of a three-input XOR gate using universal NAND gates. (b) Simulation results of the three-input XOR gate where A, B and C are clamped to logic 1, 0 and 1 respectively, while Y is left floating. It is found that Y stays in logic 0-state with a high probability. The histogram plotted has been obtained by averaging over 1000 time samples.

$$\mathrm{J}_{XOR3-NAND} = \begin{bmatrix} 0 & -1 & 0 & -3 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -3 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -3 & 0 & -2 & 0 \\ -3 & -3 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & -2 & 0 & -1 & -2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & -2 & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -2 & -2 & 0 & -3 & -2 & 0 & 0 \\ 0 & 0 & -3 & 0 & 0 & 0 & -3 & 0 & -2 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & -1 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & -1 & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 \end{bmatrix} \quad (3.7)$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.5(a). The simulation results for the same shown in Fig. 3.5(b) demonstrate the correct functionality of a 3-input XOR gate.

3.2.4 2:1 Multiplexer realized using NAND Gates

A 2:1 multiplexer (MUX) can be realized in terms of NAND gates as

$Y = \overline{\overline{(A \wedge \overline{Sel})} \wedge \overline{(B \wedge Sel)}}$, where A, B are the two inputs , Sel is the select line and Y is the output. In PSL, a 2:1 MUX synthesized using NAND gates can be implemented with the help of 7 p-bits and their interconnection is shown in Fig. 3.6(a). P-bits m_1 and m_2 are connected to the two inputs; p-bit m_3 is connected to the select line, p-bit m_7 is connected to the output and the remaining *three* p-bits are the internal p-bits of the system under realization. As the PSL circuit for a 2:1 MUX constitutes of *seven* p-bits, the h matrix will be of dimensions 7×1 while the J matrix would be of the dimensions 7×7. Following the same methodology to assign weights as done for the two-input XOR gate earlier, and according to Eq. (2.1), h and J matrices for a 2:1 MUX realized using universal (NAND) gates can be defined as follows:

$$h_{2:1MUX} = \begin{bmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \\ 2 \end{bmatrix} \qquad J_{2:1MUX} = \begin{bmatrix} 0 & 0 & 0 & -1 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -2 & 0 \\ 0 & -1 & 0 & -4 & 0 & -2 & 0 \\ -1 & 0 & -4 & 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & -2 & 0 & -1 & -2 \\ 0 & -2 & -2 & 0 & -1 & 0 & -2 \\ 0 & 0 & 0 & 0 & -2 & -2 & 0 \end{bmatrix} \quad (3.8)$$

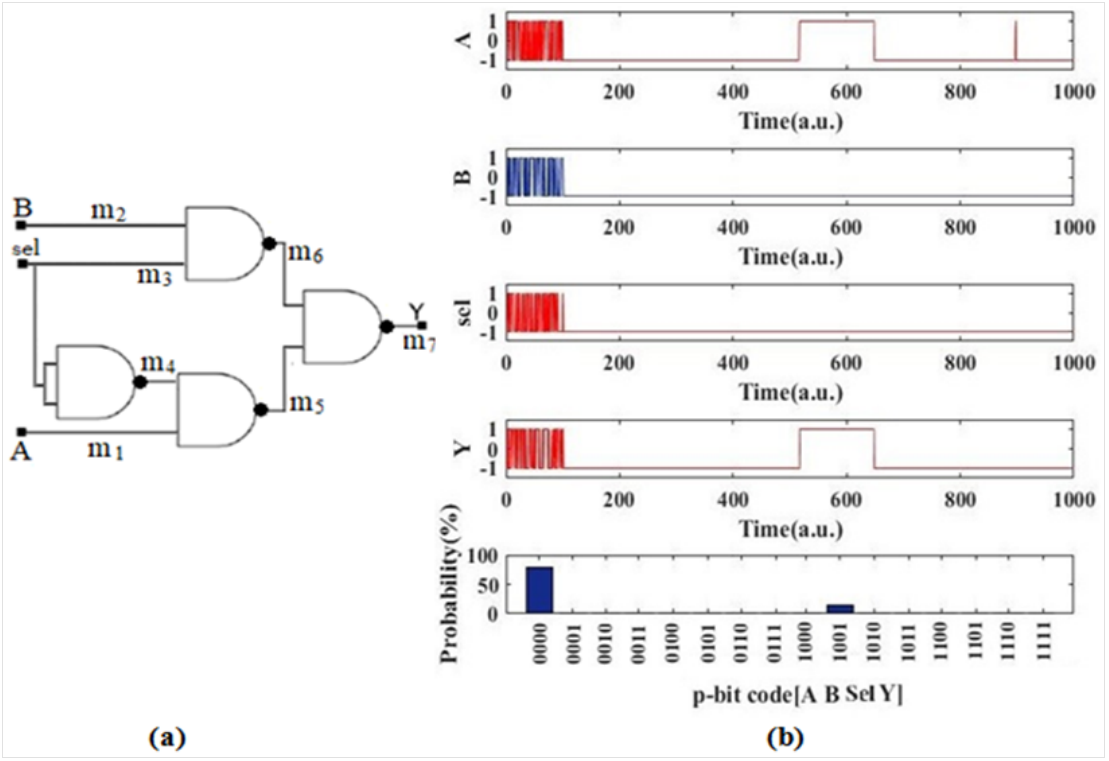


FIGURE 3.6: **2:1 MUX realized using NAND Gates:** (a) Implementation of a 2:1 MUX using universal NAND gates. (b) Simulation results of the multiplexer where A, B and sel are each clamped to logic 0, while Y is left floating. It is found that as the select signal is held low, the output Y follows the input A and hence stays in logic 0-state with a higher probability. The histogram plotted has been obtained by averaging over 1000 time samples.

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.6(a). The simulation results for the same shown in Fig. 3.6(b) demonstrate the correct functionality of a 2:1 MUX as when Sel is clamped to a value of '0', and all the other intermediate and output p-bits are allowed to fluctuate in a correlated manner, the output Y closely follows the input A thus closely demonstrating a good implementation of the Boolean function.

3.2.5 4:1 Multiplexer realized using NAND Gates

A 4:1 multiplexer can be realized in terms of universal (NAND) gates as

$Y = \overline{\overline{(A \wedge \overline{S1} \wedge \overline{S0})} \wedge \overline{(B \wedge \overline{S1} \wedge S0)} \wedge \overline{(C \wedge S1 \wedge \overline{S0})} \wedge \overline{(D \wedge S1 \wedge S0)}}$, where A, B, C, D are the four inputs; $S1$ and $S0$ are the two select lines and Y is the output. In PSL, a 4:1 MUX synthesized using NAND gates can be implemented with the help of 25 p-bits on contrary to the 7 p-bits used in constructing the 2:1 MUX implying a significant jump in the number of resources (p-bits) used at the expense of two extra inputs and a more complex circuit. P-bits m_1, m_2, m_3 and m_4 are connected to the four inputs A, B, C, D;

p-bits m_5 and m_6 are connected to the *two* select lines $S0$ and $S1$ respectively; p-bit m_{25} is connected to the output and the remaining *eighteen* p-bits are the internal p-bits of the system under realization. As the PSL circuit for a 4:1 MUX constitutes of $twenty-five$ p-bits, the h matrix will be of dimensions 25×1 while the J matrix would be of the dimensions 25×25. Following the same methodology to assign weights as done for the two-input XOR gate earlier, and according to Eq. (2.1), h and J matrices for a 4:1 MUX realized using universal (NAND) gates can be defined as follows:

$$h_{4:1MUX} = \begin{bmatrix} 1 & 1 & 1 & 1 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 4 & 3 & 3 & 2 \end{bmatrix}^T$$

$$\mathrm{J}_{4:1MUX} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & -4 & -1 & 0 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & -4 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -4 & -1 & 0 & -1 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -4 & -1 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\
-2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -1 & 0 & -2 & 0 & 0 & 0 \\
0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -1 & 0 & 0 & -2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & -4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & -4 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & -1 & -2 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & -1 & 0 & -2 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 &
\end{bmatrix} \quad (3.9)$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.7(a). The simulation results for the same shown in Fig. 3.7(b) demonstrate the correct functionality of a 4:1 MUX where $S1S0$ are clamped at

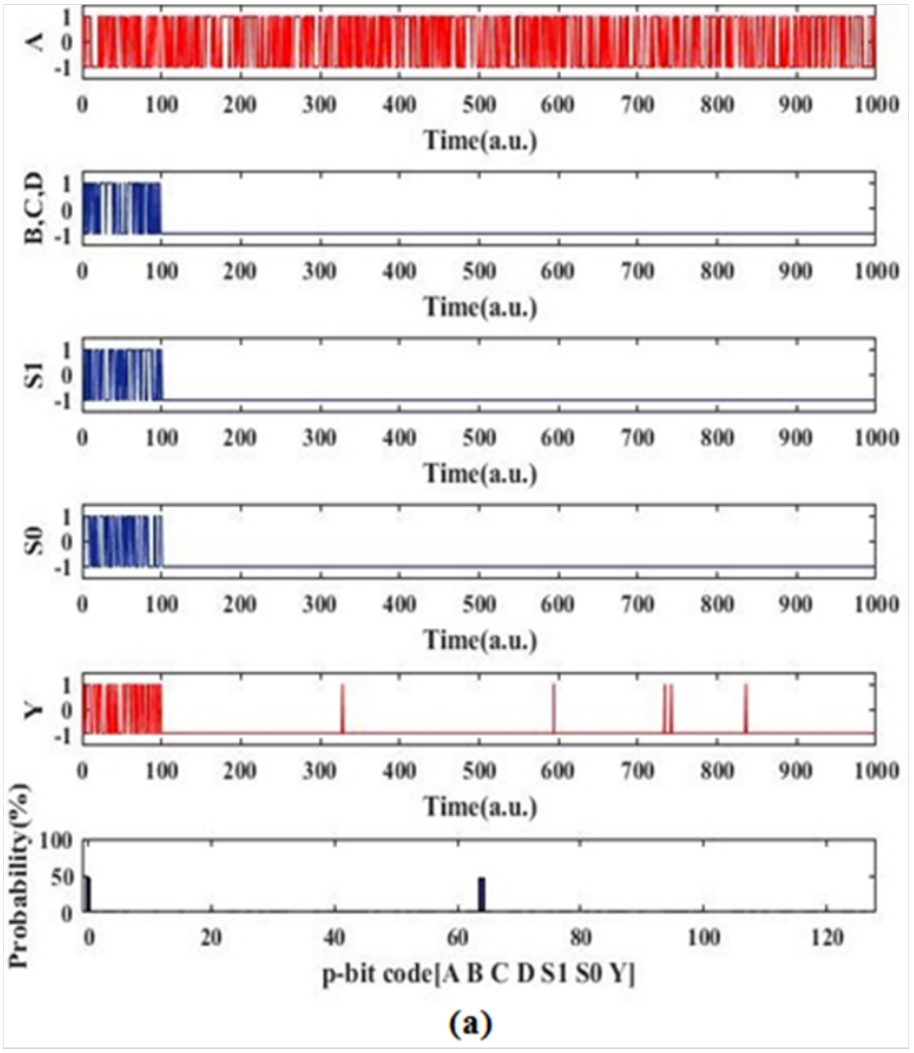


FIGURE 3.7: **4:1 MUX realized using NAND Gates:** Simulation results of the multiplexer where input A is floated, B, C, D, $S1$ and $S0$ are each clamped to logic 0, while output Y is also left floating. It is found that as the select signals are held low, the output Y follows the input A and hence visits both logic 0 and logic 1-state with an equal probability. The histogram plotted has been obtained by averaging over 1000 time samples.

'00', A input is allowed to float while the other three inputs B, C, D are clamped to logic '0'. When all the other intermediate p-bits are allowed to fluctuate in a correlated manner, only the two possible valid states of the multiplexer defined as (0000000) and (1000001) corresponding to decimal equivalents of '0' and '65' respectively are visited with an equal and high probability thus demonstrating a good implementation of the Boolean function.

3.3 Different input NAND Gate

An attempt has been made to generalize the Ising Hamiltonian for a three-input NAND gate by closely following the Ising Hamiltonians of the available basic (NOT/AND/OR) gates according to Eqs. (2.3) (2.5) (2.7). Thus, a three-input NAND gate consisting of three inputs and an output void of any internal p-bits represented using 4 p-bits m_1, m_2, m_3, and m_4 such that $m_4 = \neg(m_1 \wedge m_2 \wedge m_3)$ is simulated for the Ising Hamiltonian given as follows:

$$H_{3-INPUT-NAND} = 4-(m_1+m_2+m_3+2m_4)-(-m_1m_2-m_2m_3-m_1m_3-2m_1m_4-2m_2m_4-2m_3m_4) \tag{3.10}$$

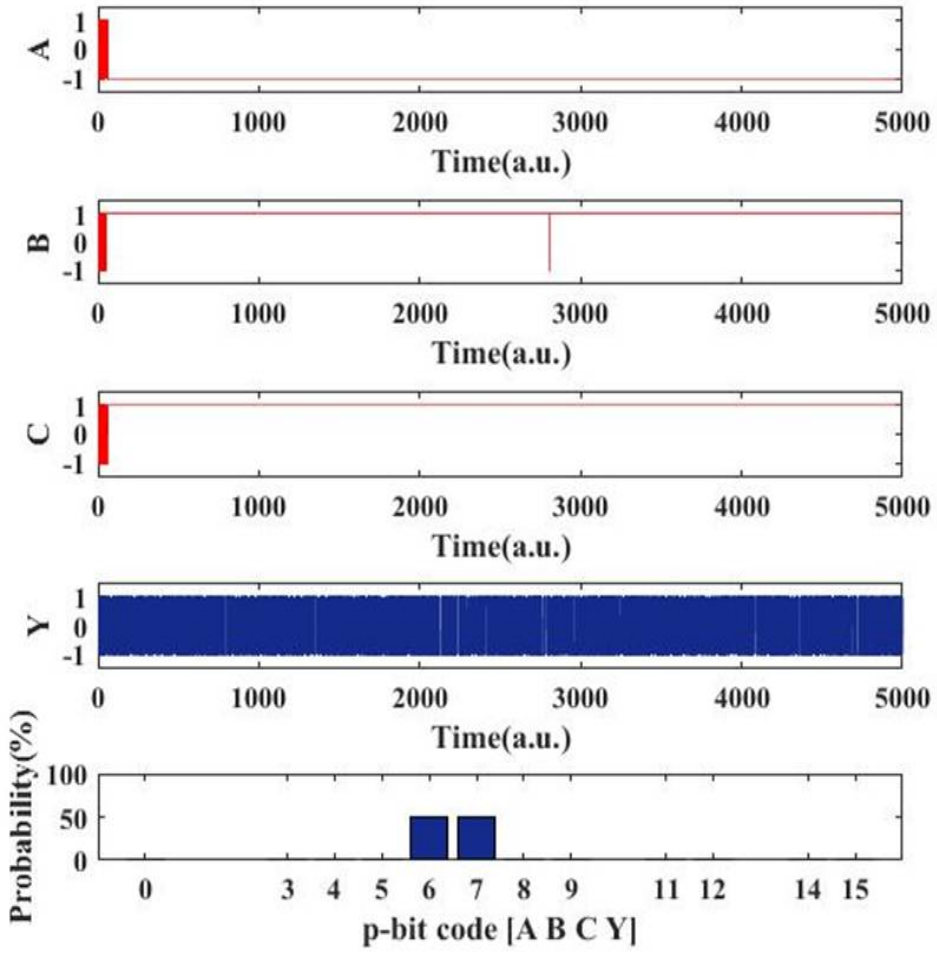


FIGURE 3.8: **3-input NAND gate without intermediate bits:** Simulation results of the three-input NAND gate where inputs A, B and C are clamped to logic 0, 1, 1 respectively while output Y is left floating. It is found that despite the given biasing of inputs, the output Y visits both logic 0 and logic 1-state with an equal probability thus failing at this particular input combination. The histogram plotted has been obtained by averaging over 5000 time samples.

On comparing Eq. (3.10) with Eq. (2.1), the h and J-matrices for a 3-input NAND gate are defined by:

$$h_{NAND3} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \end{bmatrix} \qquad J_{NAND3} = \begin{bmatrix} 0 & -1 & -1 & -2 \\ -1 & 0 & -1 & -2 \\ -1 & -1 & 0 & -2 \\ -2 & -2 & -2 & 0 \end{bmatrix} \tag{3.11}$$

The implementation of above matrices in PSL upon simulation yield results as shown in Fig. 3.8 and demonstrate correct functionality of a three-input NAND gate only for certain combination of inputs while it fails at other combination of input specifically when the three inputs are clamped to values of '0','1' and '1', respectively. Thus, due to the absence of concrete Ising Hamiltonians for three or higher order NAND gates, different input NAND gates have been realized using a basic two-input NAND gate as the fundamental block by cascading them in a series fashion to achieve the desired input NAND gate. However, it is important to point out that this work in no way concludes that realizing three-input NAND gate using only 4 p-bits is not possible.

3.3.1 Three-input NAND Gate

In PSL, a three-input NAND gate can be implemented with the help of 6 p-bits as shown in Fig. 3.9(a). P-bits m_1, m_2 and m_3 are connected to the three inputs A, B and C respectively; p-bit m_6 is connected to the output Y and the remaining *two* p-bits are the internal p-bits of the system under realization. As the PSL circuit for a three-input NAND gate constitutes of *six* p-bits, the h matrix will be of dimensions 6×1 while the J matrix would be of the dimensions 6×6. Following the same methodology used to assign weights for the two-input XOR gate, and according to Eq. (2.1), h and J matrices for a three-input NAND gate can be defined as follows:

$$h_{NAND3} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 4 \\ 3 \\ 2 \end{bmatrix} \qquad J_{NAND3} = \begin{bmatrix} 0 & -1 & 0 & -2 & 0 & 0 \\ -1 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -2 \\ -2 & -2 & 0 & 0 & -4 & 0 \\ 0 & 0 & -1 & -4 & 0 & -2 \\ 0 & 0 & -2 & 0 & -2 & 0 \end{bmatrix} \tag{3.12}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.9(a). The simulation results for the same shown in Fig. 3.9(a) demonstrate the correct functionality of the NAND gate as only the possible valid state corresponding to the chosen input value is visited with high probability when all the clamped and floated p-bits are allowed to fluctuate in a correlated manner.

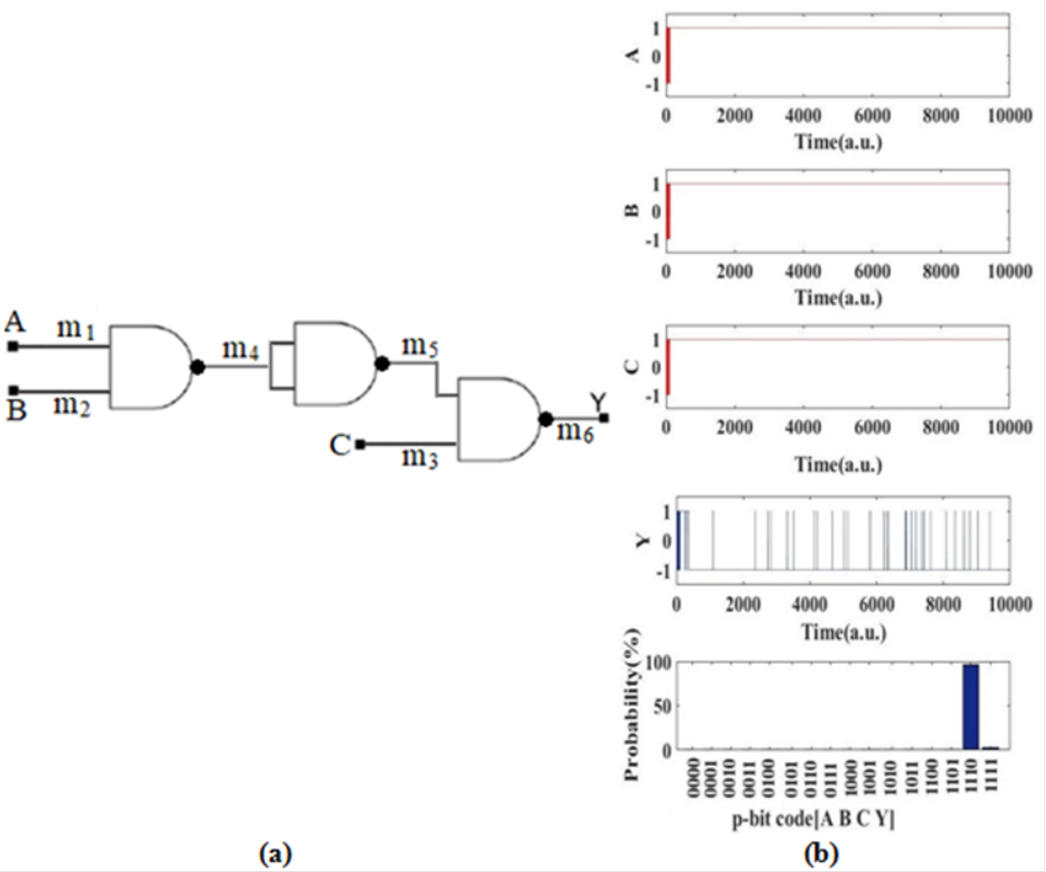


FIGURE 3.9: **3-input NAND Gate:** (a) Implementation of a three-input NAND gate using two-input NAND gates as building blocks. (b) Simulation results of the NAND gate where A, B and C are each clamped to logic 1, while Y is left floating. It is found that the output Y stays in logic 0-state with a high probability. The histogram plotted has been obtained by averaging over 10000 time samples.

3.3.2 Four-input NAND Gate

In PSL, a four-input NAND gate can be implemented with the help of 9 p-bits as shown in Fig. 3.10. P-bits m_1, m_2, m_3 and m_4 are connected to the four inputs A, B, C and D respectively; p-bit m_9 is connected to the output Y and the remaining $four$ p-bits are the internal p-bits of the system under realization. As the PSL circuit for a four-input NAND gate constitutes of *nine* p-bits, the h matrix will be of dimensions 9×1 while the J matrix would be of the dimensions 9×9. Following the same methodology used to assign weights for the two-input XOR gate, and according to Eq. (2.1), h and J matrices for a four-input NAND gate can be defined as follows:

$$h_{NAND4} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 4 \\ 3 \\ 4 \\ 3 \\ 2 \end{bmatrix} \qquad J_{NAND4} = \begin{bmatrix} 0 & -1 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 \\ -2 & -2 & 0 & 0 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -4 & 0 & -2 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & -2 & 0 & -4 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -4 & 0 & -2 \\ 0 & 0 & 0 & -2 & 0 & 0 & 0 & -2 & 0 \end{bmatrix} \tag{3.13}$$

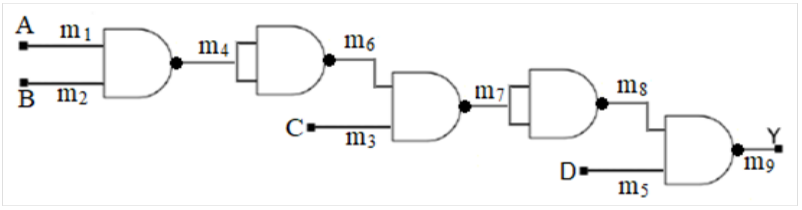


FIGURE 3.10: **4-input NAND Gate:** Implementation of a four-input NAND gate using two-input NAND gates as building blocks.

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.10. The simulation of the PSL circuit demonstrates the correct functionality of the four-input NAND gate as only the possible valid state corresponding to the chosen input value is visited with high probability when all the clamped and floated p-bits are allowed to fluctuate in a correlated manner.

3.3.3 Five-input NAND Gate

In PSL, a five-input NAND gate can be implemented with the help of 12 p-bits as shown in Fig. 3.11. P-bits m_1, m_2, m_3, m_4 and m_5 are connected to the five inputs A, B, C, D and E respectively; p-bit m_{12} is connected to the output Y and the remaining *six* p-bits are the internal p-bits of the system under realization. As the PSL circuit for a five-input NAND gate constitutes of *twelve* p-bits, the h matrix will be of dimensions 12×1 while the J matrix would be of the dimensions 12×12. Following the same methodology used to assign weights for the two-input XOR gate, and according to Eq. (2.1), h and J matrices for a five-input NAND gate can be defined as follows:

$$h_{NAND5} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 4 & 3 & 4 & 3 & 4 & 3 & 2 \end{bmatrix}^T$$

$$J_{NAND5} = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 \\ -2 & -2 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -4 & 0 & -2 \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \end{bmatrix} \quad (3.14)$$

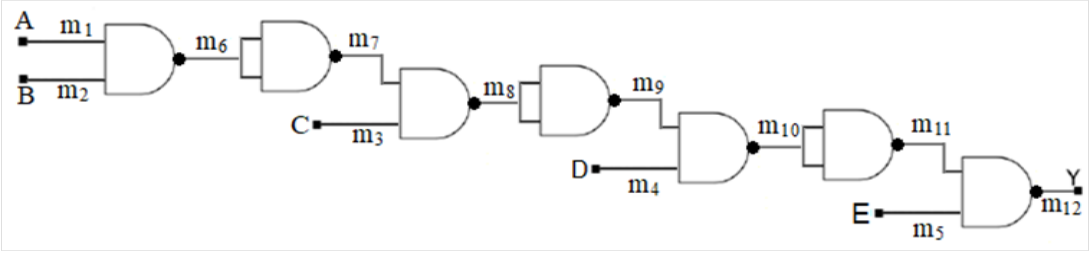


FIGURE 3.11: **5-input NAND Gate:** Implementation of a five-input NAND gate using two-input NAND gates as building blocks.

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.11. The simulation results for the same demonstrate the correct functionality of a five-input NAND gate as only the possible valid state corresponding to the chosen input value is visited with high probability when all the clamped and floated p-bits are allowed to fluctuate in a correlated manner.

3.3.4 Six-input NAND Gate

In PSL, a six-input NAND gate can be implemented with the help of 15 p-bits as shown in Fig. 3.12. P-bits m_1, m_2, m_3, m_4, m_5 and m_6 are connected to the six inputs A, B, C, D, E and F respectively; p-bit m_{15} is connected to the output Y and the remaining *eight* p-bits are the internal p-bits of the system under realization. As the PSL circuit for a six-input NAND gate constitutes of $fifteen$ p-bits, the h matrix will be of dimensions 15×1 while the J matrix would be of the dimensions 15×15. Following the same methodology used to assign weights for the two-input XOR gate, and according to Eq. (2.1), h and J matrices for a six-input NAND gate can be defined

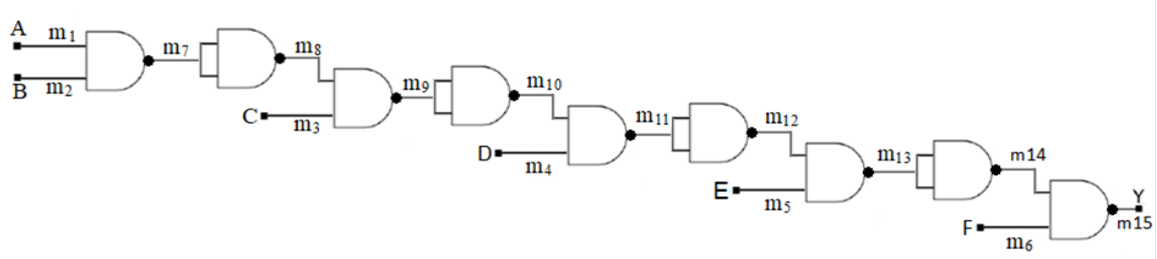


FIGURE 3.12: **6-input NAND Gate:** Implementation of a six-input NAND gate using two-input NAND gates as building blocks.

as follows:

$$h_{NAND6} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 2 \end{bmatrix}^T$$

$$\mathrm{J}_{NAND6} = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 \\ -2 & -2 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \end{bmatrix} \quad (3.15)$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.12. The simulation results for the same demonstrate the correct functionality of a six-input NAND gate as only the possible valid state corresponding to the chosen input value is visited with high probability when all the clamped and floated p-bits are allowed to fluctuate in a correlated manner.

3.3.5 Seven-input NAND Gate

In PSL, a seven-input NAND gate can be implemented with the help of 18 p-bits as shown in Fig. 3.13. P-bits m_1, m_2, m_3, m_4, m_5, m_6 and m_7 are connected to the seven

inputs A, B, C, D, E, F and G respectively; p-bit m_{18} is connected to the output Y and the remaining *ten* p-bits are the internal p-bits of the system under realization. As the PSL circuit for a seven-input NAND gate constitutes of *eighteen* p-bits, the h matrix will be of dimensions 18×1 while the J matrix would be of the dimensions 18×18. Following the same methodology used to assign weights for the two-input XOR gate, and according to Eq. (2.1), h and J matrices for a seven-input NAND gate can be defined as follows:

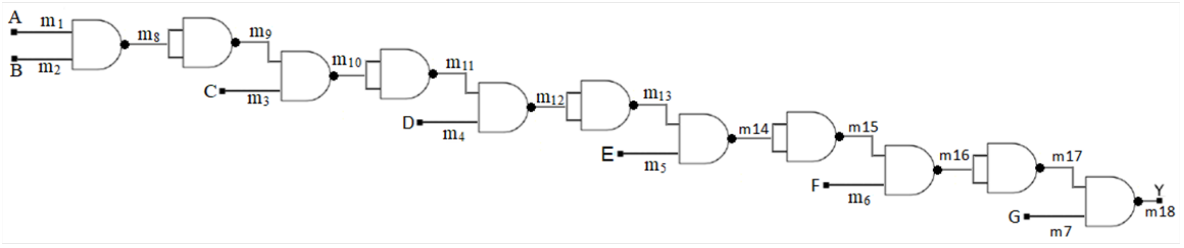


FIGURE 3.13: **7-input NAND Gate:** Implementation of a seven-input NAND gate using two-input NAND gates as building blocks.

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.13. The simulation results for the same demonstrate the correct functionality of a seven-input NAND gate as only the possible valid state corresponding to the chosen input value is visited with high probability when all the clamped and floated p-bits are allowed to fluctuate in a correlated manner.

$$h_{NAND7} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 2 \end{bmatrix}^T$$

$$\mathrm{J}_{NAND7} = \begin{bmatrix}
0 & -1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 \\
-2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 \\
0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0
\end{bmatrix} \tag{3.16}$$

3.3.6 Eight-input NAND Gate

In PSL, an eight-input NAND gate can be implemented with the help of 21 p-bits as shown in Fig. 3.14. P-bits m_1, m_2, m_3, m_4, m_5, m_6, m_7 and m_8 are connected to the eight inputs A, B, C, D, E, F, G and H respectively; p-bit m_{21} is connected to the output Y and the remaining *twelve* p-bits are the internal p-bits of the system under realization. As the PSL circuit for an eight-input NAND gate constitutes of $twenty - one$ p-bits, the h matrix will be of dimensions 21×1 while the J matrix would be of the dimensions 21×21. Following the same methodology used to assign weights for the two-input XOR gate, and according to Eq. (2.1), h and J matrices for an eight-input NAND gate can be defined as follows:

$$h_{NAND8} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 2 \end{bmatrix}^T$$

$$J_{NAND8} = \begin{bmatrix}
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 \\
-2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0
\end{bmatrix} \quad (3.17)$$

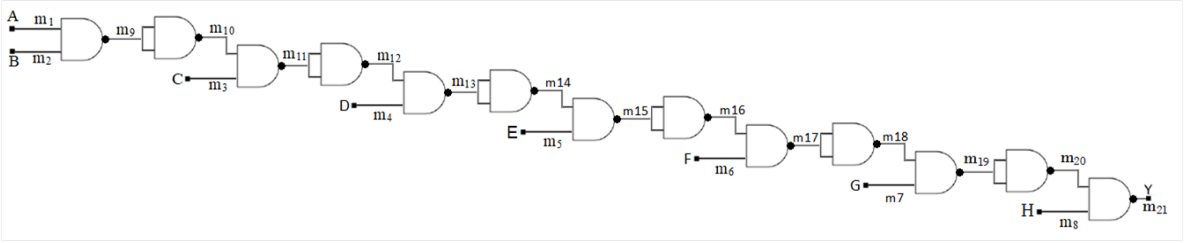
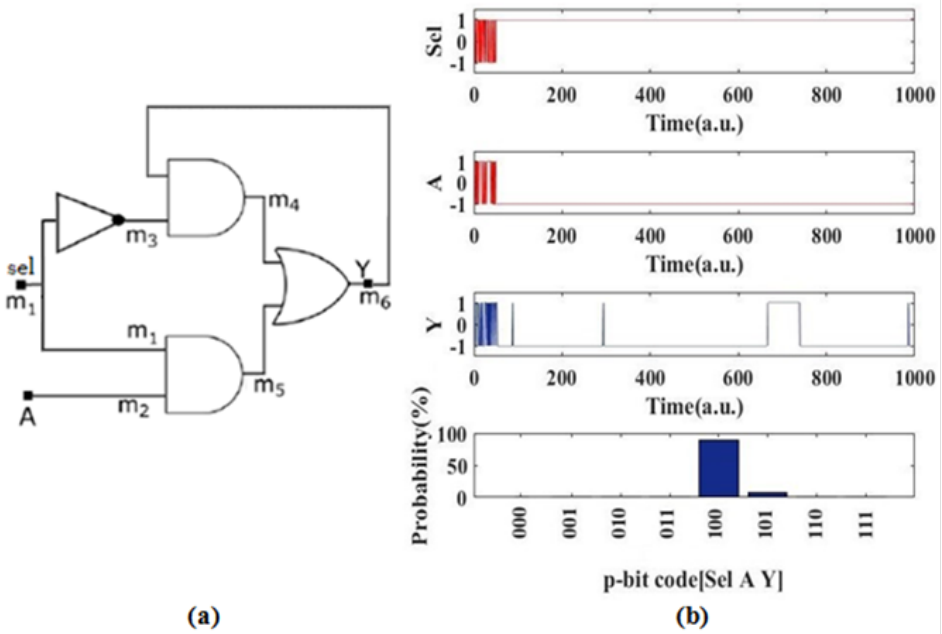


FIGURE 3.14: **8-input NAND Gate:** Implementation of an eight-input NAND gate using two-input NAND gates as building blocks.

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.14. The simulation results for the same demonstrate the correct functionality of an eight-input NAND gate as only the possible valid state corresponding to the chosen input value is visited with high probability when all the clamped and floated p-bits are allowed to fluctuate in a correlated manner.



3.4 Sequential Logic circuit

Through numerous examples discussed above, efficient PSL implementation of combinational Boolean logic has been established. In an attempt to further understand the behavior of sequential circuits implemented using PSL, the most fundamental sequential element, a latch, has been realized using a 2:1 MUX. A latch can be realized in terms of a 2:1 MUX as $Y = (A \wedge Sel) \vee (Y \wedge \overline{Sel})$, where A is the input, Sel is the level sensitive clock pulse and Y is the output of the latch. In PSL, the latch can be implemented with the help of 6 p-bits and their interconnection is shown in Fig. 3.15(a). P-bits m_1 and m_2 are connected to the two inputs Sel and A respectively; p-bit m_6 is connected to the output and the remaining *three* p-bits are the internal p-bits to the system under realization. Initially the clock is set to high so the latch is transparent and the output Y follows the input signal A as shown in Fig. 3.15(b). As soon as the Sel signal or the clock goes low, the latching action takes place and despite any transitions (from a low to high value) at the input, the output still retains the previous state as shown in Fig. 3.15(c). Thus, if Sel signal is set to a value of '0, the latch retains the previous state else if Sel signal is set to a value of 1, the output follows the input. As the PSL circuit for a latch constitutes of six p-bits, the h matrix will be of dimensions 6×1 while the J matrix would be of the dimensions 6×6. According to Eq. (2.1), h and J matrices for a latch can be defined as follows:

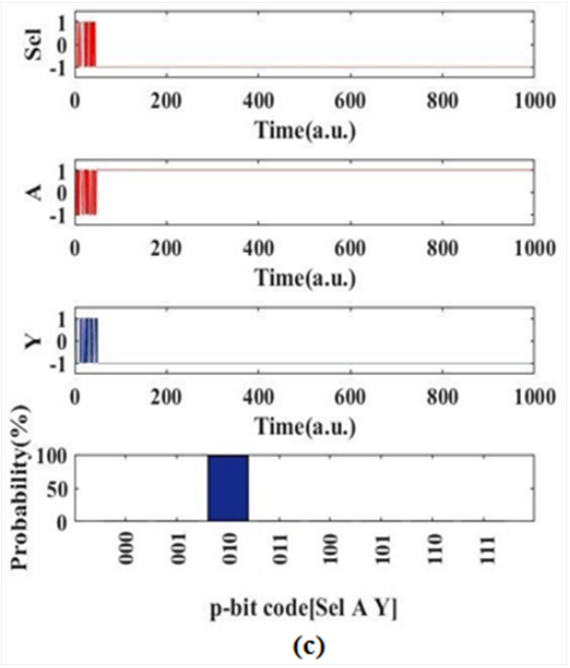


FIGURE 3.15: **Latch using 2:1 MUX:** (a) Implementation of a latch using 2:1 MUX in PSL. (b) Simulation results of the latch where A is clamped to logic 0 initially, Sel is set to high while Y is left floating. It is found that the output Y follows the input as the latch is transparent and thus, stays in logic 0-state with a high probability. The histogram plotted has been obtained by averaging over 1000 time samples. (c) As the simulation continues, A is now clamped to logic 1, Sel is set to low while Y is left floating. It is observed that the output Y does not follow the input as the latch is opaque now and thus, stays in logic 0-state retaining the previously stored value with a high probability. The histogram plotted has been obtained by averaging over 1000 time samples.

$$h_{LATCH} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -3 \\ -3 \\ 3 \end{bmatrix} \qquad J_{LATCH} = \begin{bmatrix} 0 & -1 & -1 & 0 & 2 & 0 \\ -1 & 0 & 0 & 0 & 2 & 0 \\ -1 & 0 & 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & 0 & -1 & 4 \\ 2 & 2 & 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & 4 & 2 & 0 \end{bmatrix} \tag{3.18}$$

The implementation of above matrices in PSL is governed by the interconnection between the p-bits as shown in Fig. 3.15(a). The simulation results for the same shown in Fig. 3.15(b)(c) demonstrate the functionality of a latch as when Sel is set to a high value, the output follows the input else if Sel is set to a low value, the latch becomes opaque and holds the previous state with a high probability as expected in an ideal latch behavior. However, it is important to point out that the operation of a sequential circuit depends on the timing of various signals and their inter-relationships. In the above simulation, the temporal relationship of various signals has not be treated rigorously and therefore

more work needs to be done to establish the correctness of sequential logic circuit realized using PSL. In rest of this thesis, only combinational circuits are considered.

Chapter 4

Quantifying Delay in PSL circuits

The propagation delay of digital circuits is defined as the difference in time (calculated at 50% of input-output transition) between the application of the input and the transition of the output. It measures the speed of logic circuits. The propagation delay can be further classified as the falling propagation delay (t_{PHL}) or the rising propagation delay (t_{PLH}) depending on whether the output switches from a high-to-low value or a low-to-high value, respectively. This definition can be extended to model the delay in PSL circuits as illustrated in this chapter. Moreover, similar to conventional logic circuits, single input switching (SIS) model can be considered in this work. An SIS model characterizes the delay of a timing arc of a multiple input gate such that only one of the inputs switches and causes a transition at the output.

4.1 Dynamics of a PSL circuit

Once an input is given to a PSL circuit, the p-bits fluctuate in a correlated manner to give the desired output. If the input is fixed, all other p-bits vary in a random but correlated fashion such that the correct output is obtained. Now if one of the input p-bits is changed, the intermediate p-bits start fluctuating again in a correlated manner. In the intermediate state, the p-bits could still have the wrong values thus giving a wrong output. As time progresses, the correlation between p-bits shifts them to their correct values thus, the probability of p-bits taking their correct values increases with increase in time and finally a steady state is reached.

P-bits are probabilistic in nature and vary with time as described above. They have certain probabilities of being either in '1' or '0'-logic state which is modulated by the interactions between them. To identify a point at which the probability of the value

of the p-bits has shifted from one particular value to the other, is a more difficult problem than in a conventional logic gate. To compute the probability and to quantify it, some kind of measurement over a number of time samples is required. The higher the number of samples taken, greater would be the accuracy in the computation to find the probability of a p-bit in being in a certain state (1/0). Ideally, for highest precision, infinite number of time samples should be taken. However, the number of time samples that can be taken to measure the probability of being in state 1/0 for a p-bit is limited by the time scale of the dynamics of a PSL system. For example, if the delay of the PSL circuit is in the range of $1ns$, the probability cannot be computed for 1μs. Based on this observation, two techniques to quantify the delay in a PSL circuit is proposed:

1. Modeling delay using a sliding window
2. Modeling delay as an intrinsic property of a circuit

These two techniques are described in the following paragraphs.

4.2 Modeling delay using a sliding window

To calculate the delay of a PSL circuit using the sliding window mechanism, a suitable window size (W) of width ranging from 200 to 5000 time samples is defined first. The window refers to a sliding window of a finite number of time samples taken at a time across which probability of occurrence of the correct output state is calculated.

At the beginning of the simulation, the output state for W number of time samples is initialized with all the possible output values with an equal probability. As the simulation progresses, the circuit is simulated for a pre-determined number of time samples ($t1$) for a certain input combination ($h_{fix-bias}$). At $t1$ time instant, one of the controlling input p-bit values in the $h_{fix-bias}$ matrix is switched and the circuit is further simulated for an infinite number of time samples. The transition of the input p-bit value at $t1$ time sample is chosen such that it leads to a rising or falling transition at the output p-bit. It is crucial to note that the gate may be asymmetrical in behavior. Therefore, the transition at different input p-bits (one at a time) can result in different delays. The delay depends on the distance (number of intermediate p-bits) of the input p-bit and the output p-bit of the gate.

The delay of the PSL circuit modeled using the sliding window mechanism can be described as below:

The simulation value (0/1) at the k^{th} time sample is denoted as Z_k.

The number of samples in the sliding window is taken as W.

The delay of the gate is denoted as τ (measured as number of samples).

The probability of output p-bit assuming logic '1' at any instant t can be given as follows:

$$P_t = \frac{\sum_{j=0}^{W-1} Z_{t-W+j}}{W} \tag{4.1}$$

The delay of the gate is defined as τ such that the probability P_t crosses 50% mark:

$$P_\tau < 50\% \tag{4.2}$$

In simulations and experiments done in this work, it is assumed that the circuit is in the steady state before $t1$. At $t1$ time instant, one of the input p-bits is switched to cause a transition at the output p-bit. A window size of W is taken and the probability of the output state to be in logic '1' is computed over W time samples. The window is then shifted by one time sample and the percentage of output state to be in logic '1' is computed again for W time samples and the process repeats for an infinite time samples. The computed probability percentage of the output state for each of those sliding windows is plotted against corresponding time instance and the number of time samples taken by the circuit after $t1$ to cross 50% of the initial output value thus defines the delay of the PSL circuit.

4.3 Results of delay computation using the sliding window approach

4.3.1 Comparison of delay of an inverter, two-input NAND gate and a two-input NAND gate synthesized using AND and NOT gates

The delay characteristics of a two-input NAND gate, a NAND gate implemented using basic gates of AND+NOT gate and an Inverter have been compared. Using the methodology described above, delay for the above circuits is calculated for different window sizes ranging from 100 to 1200 time samples in width and shown in Fig. 4.1 and table 4.1.

As observed from the Fig. 4.1, the delay of these circuits is a function of both the number of resources (p-bits) used to design the circuit as well as the window size chosen to calculate the delay of the PSL circuit. The inverter comprising of only 2 p-bits has the

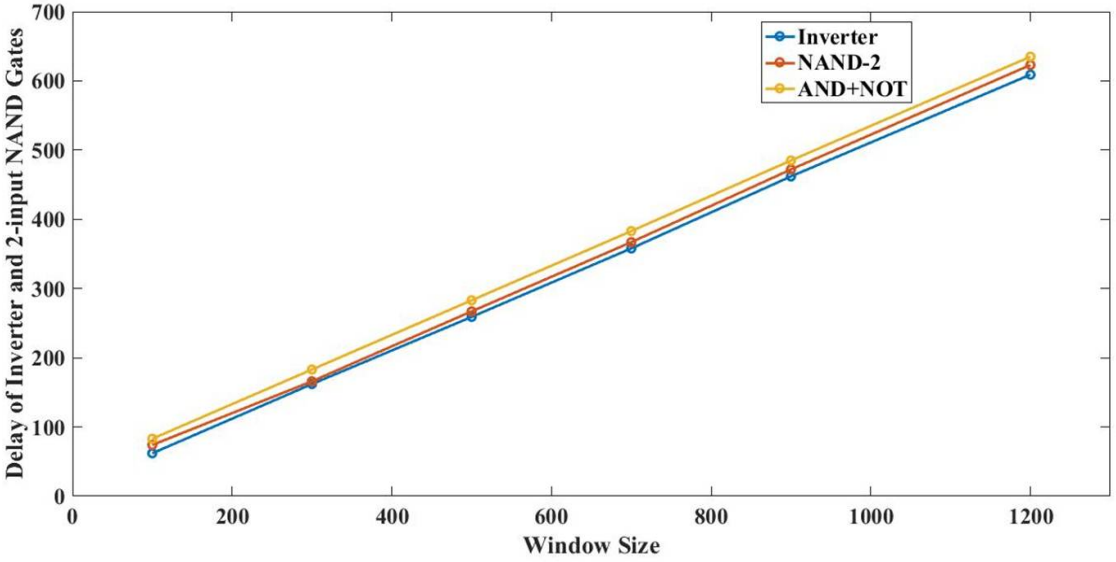


FIGURE 4.1: Delay of an inverter, two-input NAND gate and a two-input NAND gate synthesized using AND and NOT gates plotted as a function of the window sizes ranging from 100 to 1200 time samples in width.

$Window\ Size$	$Inverter$	$2-Input\ NAND$	$AND+NOT\ Gate$
100	62	74	83
300	162	166	183
500	259	267	283
700	358	367	383
900	462	472	485
1200	609	623	635

TABLE 4.1: Delays of different gates as function of different Window sizes

least delay among the three circuits while the AND+NOT gate implementation comprising of 4 p-bits has the maximum delay across all window sizes. The 2-input NAND gate comprising of 3 p-bits has lesser delay than the AND+NOT gate implementation as the number of p-bits used to synthesize the circuit are less. It should be noted that the previous output (output before the transition at the input p-bit) is also considered by the sliding window in the delay computation. Therefore, as the window size increases, the effect of the previous output remains for a longer duration. Therefore, the delay of all the circuits is observed to increase as the window size increases.

In general, when there are multiple possible implementation of a logic function, some trade-offs are involved. For example, in conventional CMOS logic, if the delay of an implementation is improved, then the area and the power dissipation degrade. Therefore, in PSL circuits also, it is important to assess if some trade-offs are involved in choosing an optimal implementation. Some of the metrics for trade-off in PSL can be number of resources (p-bits), delay and accuracy.

Next, the trade-offs involved in the two implementations of 2-input NAND gate is considered. Both the circuits are simulated for different time samples ranging from 100 to 10000 time samples and the accuracy metric of the circuits is observed for these different time samples. Since PSL realizes a Boolean function in a probabilistic manner, the quality of an implementation must be measured using some statistical technique. The accuracy of the circuit thus has been quantified in terms of the RMS error measure that indicates the deviation of the PSL circuit behavior from the expected ideal behavior and can be calculated for a n-input Boolean function as follows:

$$P_{error} = \sqrt{\frac{\sum_Q p_i^2}{Q}} \tag{4.3}$$

where $Q = 2^n$ are the number of possible input patterns and p_i is the probability of getting a wrong output for the i^{th} input pattern. The P_{error} would be zero for deterministic logic and it should be close to zero for a good PSL implementation.

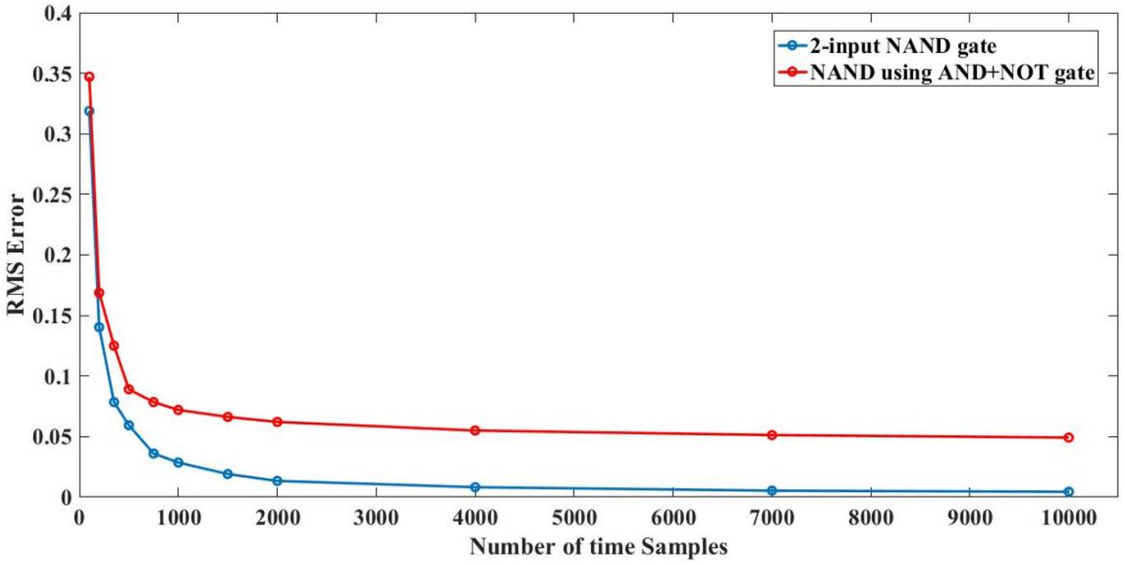


FIGURE 4.2: The RMS error plotted as a function of time samples for two different implementations of the two-input NAND gate.

In Fig. 4.2, the RMS error has been plotted as a function of time samples for the two different implementations. It is observed that the 2-input NAND gate implementation is more accurate than the basic gate (AND + NOT) gate implementation. This can be explained due to absence of any intermediate p-bits (which could settle to a wrong value) in the NAND implementation. Moreover, the RMS error also reduces as the time sample increases which is in conjunction with the expected behavior as the PSL circuits get more time to settle to the desired stable value with increase in the number of the time samples. However, it is interesting to note that the reduction in error metric is significant for smaller time samples but as the time samples increase drastically, the

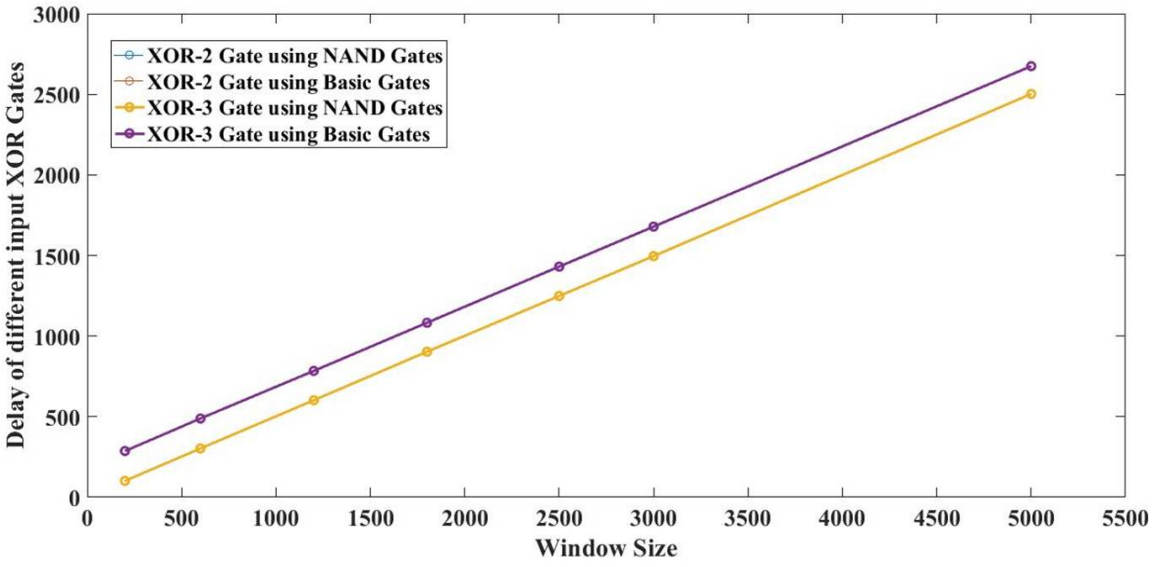
error converges to a constant value implying the steady state behavior of the respective implementations. Thus, it has been observed that the NAND gate implementation outperforms the AND+NOT gate implementation both in terms of delay and accuracy for the given functionality. Therefore, for this simple functionality, no trade-off is required in choosing an optimal solution.

4.3.2 Comparison of delay of different input XOR gates synthesized using basic or universal gates

The delay characteristics of a two-input XOR gate implemented using basic gates, two-input XOR gate implemented using NAND gates, three-input XOR gate implemented using basic gates and three-input XOR gate implemented using NAND gates have been compared. Using the methodology described above, delay for the above circuits is calculated for different window sizes (W) ranging from 200 to 5000 time samples in width where $t1$ is 10000 time samples wide.

$Window\ Size$	$XOR-2(using\ NAND)$	$XOR-2(using\ Basic)$	$XOR-3(using\ NAND)$	$XOR-3(using\ Basic)$
200	101	102	103	287
600	301	303	304	489
1200	601	601	602	784
1800	900	899	903	1083
2500	1250	1249	1251	1431
3000	1500	1501	1496	1679
5000	2500	2501	2502	2674

TABLE 4.2: Delays of different XOR gates as function of different Window sizes



As observed from Fig. 4.3, the delay of 2-input XOR gates is comparable to a 3-input XOR gate implemented using NAND gates while the 3-input XOR gate implemented using basic gates has the maximum delay due to the higher number of p-bits (13 p-bits)

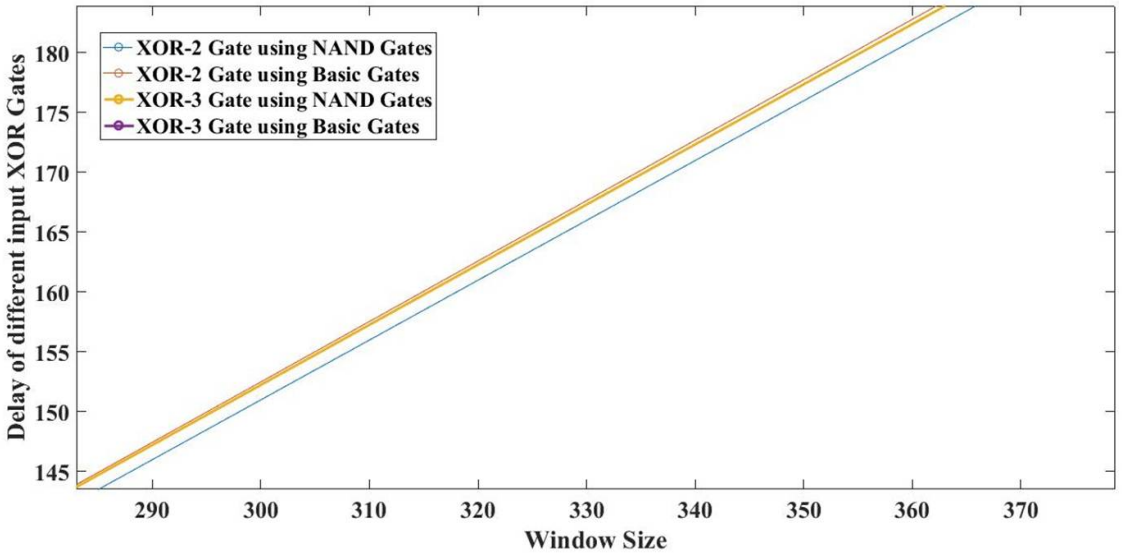


FIGURE 4.3: Delay of a two-input XOR gate implemented using basic gates, two-input XOR gate implemented using NAND gates, three-input XOR gate implemented using basic gates and three-input XOR gate implemented using NAND gates plotted as a function of the window sizes ranging from 200 to 5000 time samples in width. The zoomed in version shows the difference among the comparable delays of 2-input XOR gates and the 3-input XOR gate synthesized using NAND gates.

used to synthesize the circuit. From the above zoomed in version of the three seemingly comparable delays, it is observed that 3-input XOR gate synthesized using NAND gates (implemented using 11 p-bits) in fact has higher delay than the 2-input XOR gates which is attributed to a rather higher number of resources (p-bits) used in designing the circuit. Thus, as the number of p-bits in the design increase drastically, the delay of the logic gates is expected to increase generally implying a linear relationship between the delay and the number of resources (p-bits) used to design the circuit.

4.3.3 Comparison of delay of different multiplexers synthesized using NAND gates

The delay characteristics of a $2:1$ MUX and a $4:1$ MUX synthesized using NAND gates have been compared. Using the methodology described above, delay for the above circuits is calculated for different window sizes (W) ranging from 200 to 5000 time samples in width where $t1$ is 10000 time samples wide.

As observed from Fig. 4.4, the delay of $4:1$ MUX (implemented using 25 p-bits) is much larger than the $2:1$ MUX (implemented using 7 p-bits) for all the selected window sizes due to a rather higher number of resources (p-bits) used in designing the $4:1$ multiplexer PSL circuit and thus, falling in conjunction with the earlier theory of a higher number of p-bits in the design correspond to a larger delay.

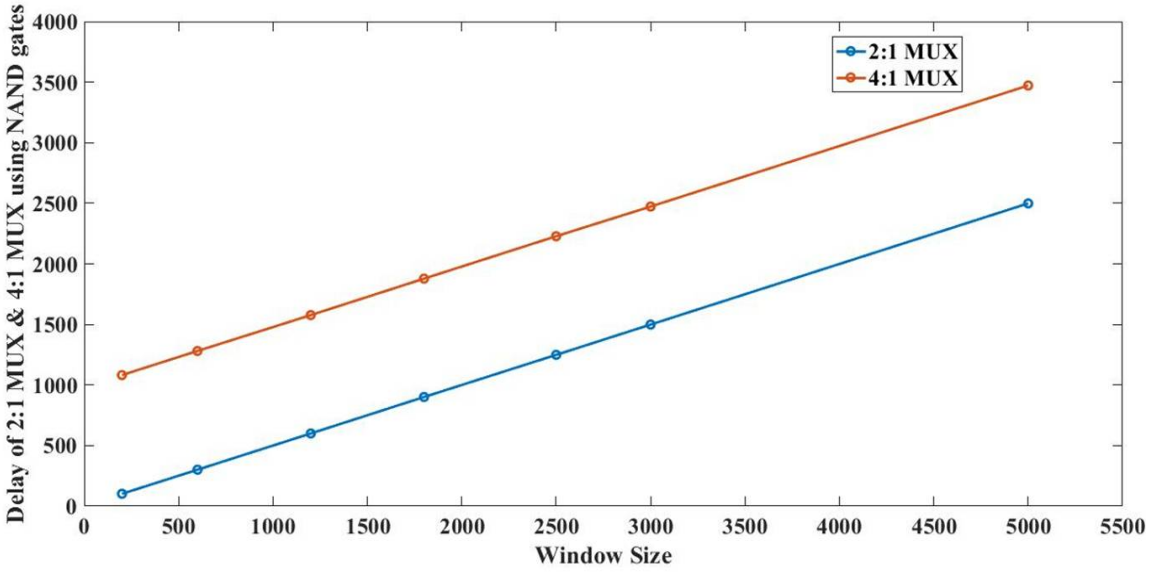


FIGURE 4.4: Delay of a 2 : 1 MUX and 4 : 1 MUX synthesized using NAND gates plotted as a function of the window sizes ranging from 200 to 5000 time samples in width. As evident from the graph, 4 : 1 MUX has a much larger delay than the 2 : 1 MUX for all window sizes matching the expected behaviour.

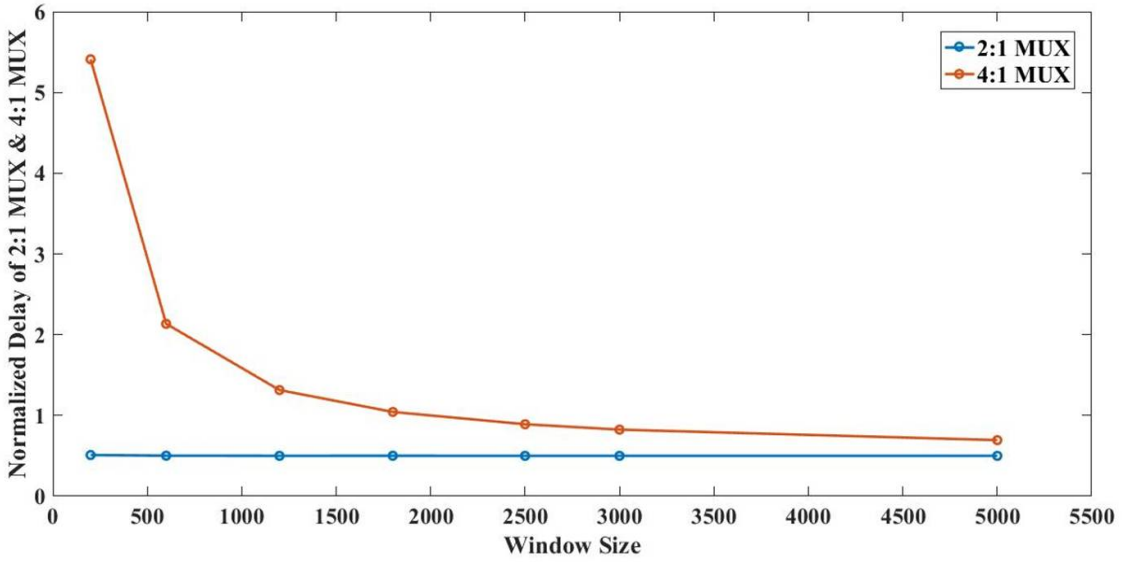


FIGURE 4.5: Normalized Delay of a 2 : 1 MUX and 4 : 1 MUX synthesized using NAND gates plotted as a function of the window sizes ranging from 200 to 5000 time samples in width. The circuits have a relatively higher delay for smaller window sizes.

The normalized delay of the two multiplexer circuits plotted in Fig. 4.5, is calculated by dividing the corresponding delay of each circuit for a particular window with its window size. As observed, for smaller window sizes, a relatively higher value of the normalized delay corresponds to the initial transient behavior of the circuit where as the relatively constant delay values for larger window sizes corresponds to the steady state behavior of the circuit under observation. Thus, keeping the window size big could be desirable to closely follow the steady state behavior of the circuit.

$Window\ Size$	$2:1\ MUX$	$4:1\ MUX$
200	102	1082
600	301	1281
1200	601	1577
1800	901	1878
2500	1249	2228
3000	1499	2473
5000	2499	3472

TABLE 4.3: Delays of 2:1 MUX and 4:1 MUX as function of different Window sizes

4.3.4 Comparison of fall delay of different input NAND gates

The fall propagation delay (t_{PHL}) characteristics of a $two-$, $three-$, $four-$, $five-$, $six-$, $seven-$, and an *eight*-input NAND gates implemented using *two*-input NAND gate as the fundamental building block have been compared. In order to replicate a falling transition at the output p-bit (that is the output transitions from a high-to-low value), one of the input p-bits is made to switch from a low-to-high value while all the other input p-bits are kept at a high value (non-controlling value). Using the methodology described above to incorporate the maximum delay experienced by a PSL circuit, the first input p-bit of the $h_{fix-bias}$ vector is switched and the delay for different input NAND gates is calculated for window sizes (W) ranging from 200 to 5000 time samples in width where $t1$ is 10000 time samples wide.

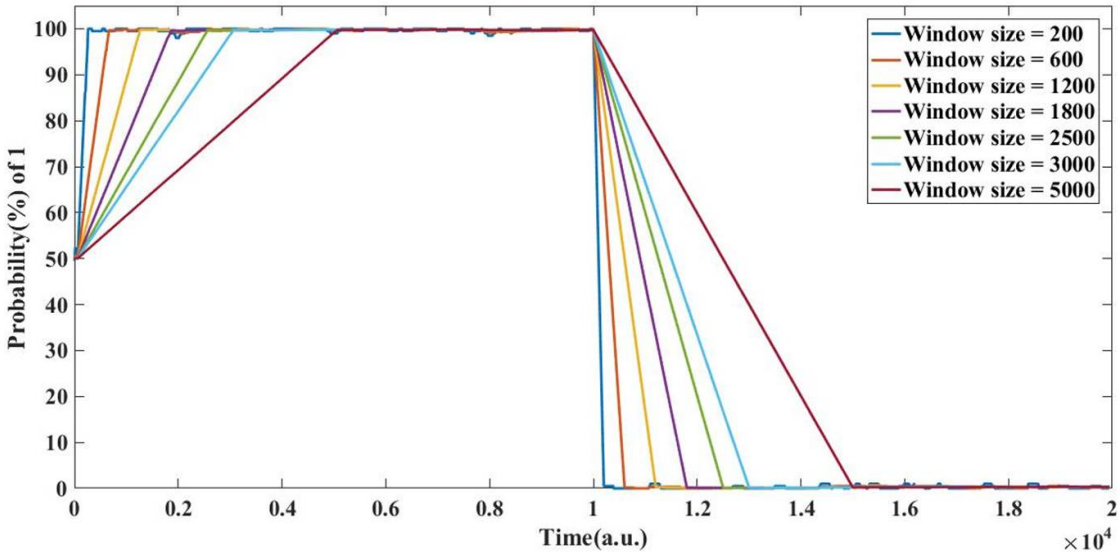


FIGURE 4.6: **2-input NAND gate:** The fall delay of a two-input NAND gate plotted as a function of time samples for different window sizes ranging from 200 to 5000 time samples in width.

As observed from Fig. 4.7, the 8-input NAND gate (implemented using 21 p-bits) has the maximum delay while the 2-input NAND gate (implemented using 3 p-bits) has the

Window Size	$NAND-2$	$NAND-3$	$NAND-4$	$NAND-5$	$NAND-6$	$NAND-7$	$NAND-8$
200	100	634	790	1004	1474	1593	1790
600	300	834	990	1204	1671	1792	1989
1200	600	1133	1290	1504	1973	2084	2288
1800	902	1435	1590	1806	2271	2382	2586
2500	1256	1785	1940	2156	2621	2732	2935
3000	1506	2035	2189	2405	2872	2979	3184
5000	2507	3035	3190	3405	3873	3980	4183

TABLE 4.4: Fall Delay of different input NAND Gates as function of different Window sizes

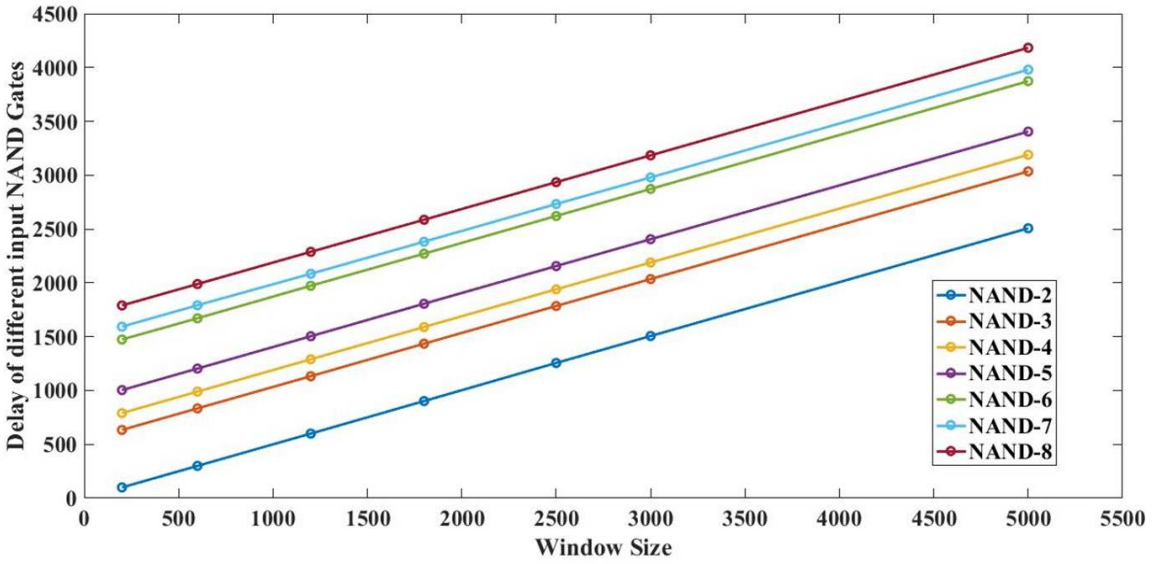


FIGURE 4.7: **Fall Delay of different input NAND gates:** The fall delay of different input NAND gates plotted as a function of window sizes ranging from 200 to 5000 time samples in width. The fall propagation delay is observed to increase with an increase in the number of inputs.

minimum delay which falls in line with the expected behavior. Moreover, the fall delay increases with the number of inputs from a 2-input NAND gate to an 8-input NAND gate for all the selected window sizes due to increase in the number of resources (p-bits) used in designing the PSL circuits.

4.3.5 Comparison of rise delay of different input NAND gates

The rise propagation delay (t_{PLH}) characteristics of a $two-, three-, four-, five-, six-$, $seven-$, and an *eight*-input NAND gates implemented using *two*-input NAND gate as the fundamental building block have been compared. In order to replicate a rising transition at the output p-bit (that is the output transitions from a low-to-high value), one of the input p-bits is made to switch from a high-to-low value while all the other input p-bits are kept at a high value (non-controlling value). Using the methodology described above to incorporate the maximum delay experienced by a PSL circuit, the first input p-bit of the $h_{fix-bias}$ vector is switched and the delay for different input NAND gates is

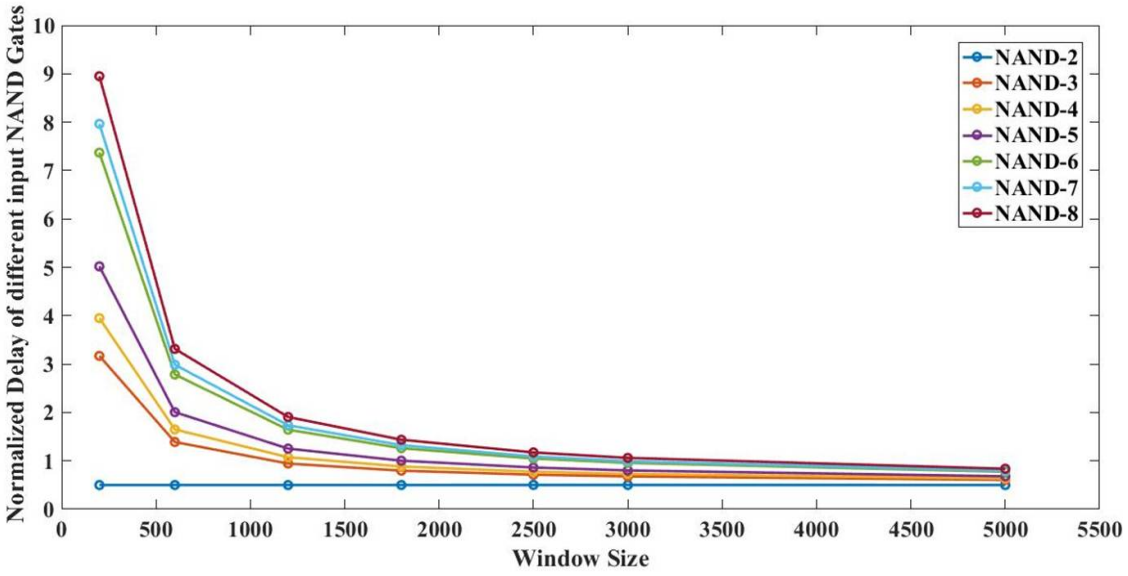


FIGURE 4.8: The Normalized fall delay of different input NAND gates plotted as a function of window sizes ranging from 200 to 5000 time samples in width. The circuits have a relatively higher delay for smaller window sizes.

calculated for window sizes (W) ranging from 200 to 5000 time samples in width where $t1$ is 10000 time samples wide.

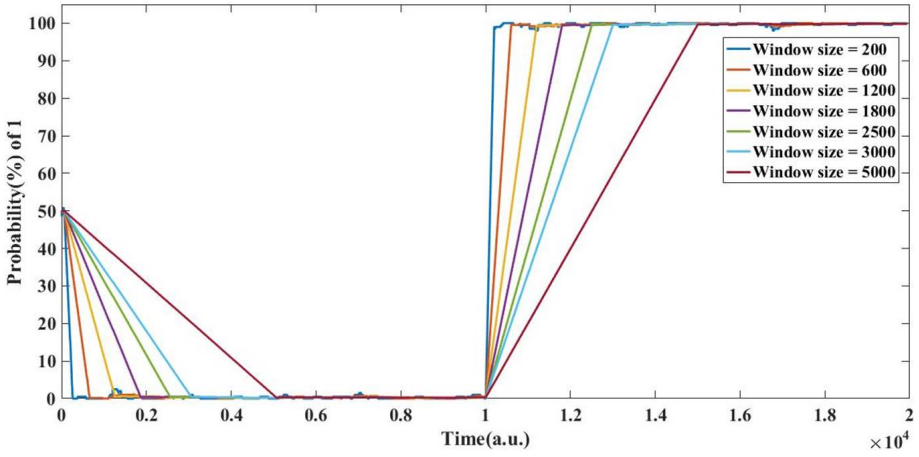


FIGURE 4.9: **2-input NAND gate:** The rise delay of a two-input NAND gate plotted as a function of time samples for different window sizes ranging from 200 to 5000 time samples in width.

It is observed from Fig. 4.10, that the rise delay remains relatively constant for all selected window sizes on contrary to the fall delay, for different input NAND gates. It does not depend on the complexity of the design and is a constant value for a particular window irrespective of the number of resources (p-bits) used in the PSL circuit. This rather unexpected behavior of the NAND gates to give a constant rise delay and an increasing fall delay with an increase in the number of inputs has been explored in greater depth in the following chapter.

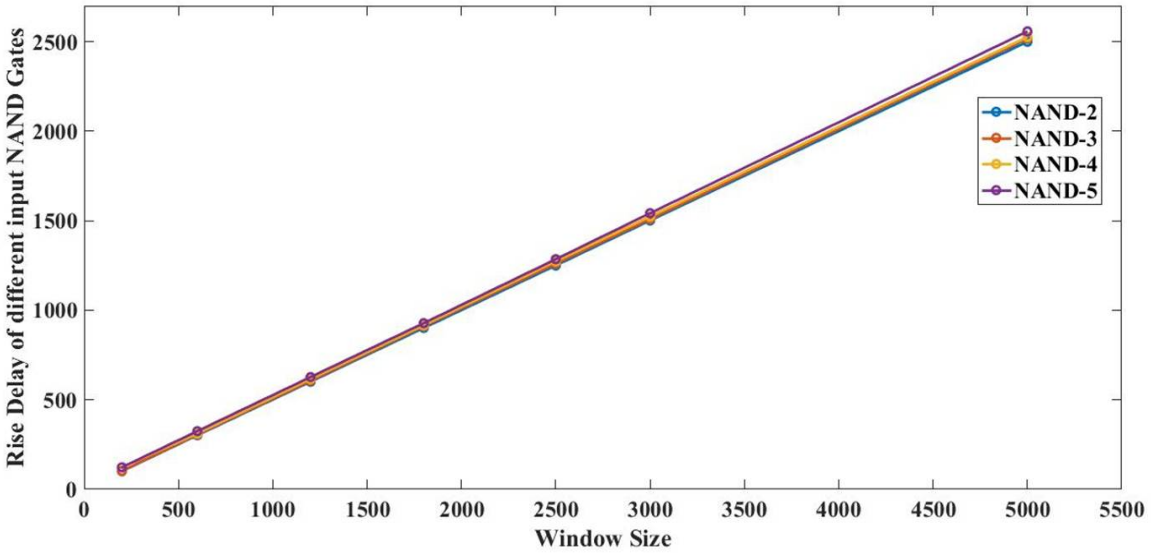


FIGURE 4.10: **Rise Delay of different input NAND gates:** The rise delay of different input NAND gates plotted as a function of window sizes ranging from 200 to 5000 time samples in width. The rise propagation delay remains constant with an increase in the number of inputs.

As observed from the above examples, the delay of a PSL circuit modeled using the sliding window mechanism is a function of both, the intrinsic characteristic of the circuit and the window size in consideration; as the window size increases the delay also increases. The dependence of delay as an artifact of the computational process can be explained with the help of a conventional digital circuit such as an inverter. In order to characterize the delay of an inverter, a capacitor is attached to the output of the inverter as shown in Fig. 4.11. As the load of the capacitor increases, the delay increases. The

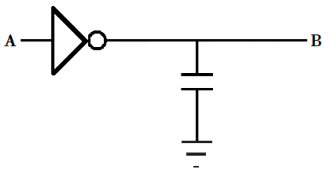


FIGURE 4.11: Characterization of delay in a conventional logic gate (Inverter) with a capacitor at the output.

observed behavior of increase in delay of a PSL circuit as the window sizes increases is analogous to this behavior. Intuitively, with an increase in the number of time samples, the past effect of the output is retained in the same way as a capacitor stores the previous value of the charge/voltage. Therefore, the output retaining the previous value and governing delay is similar to storing charge in the conventional circuit. Thus, a new way to characterize delay irrespective of a capacitor or the window size has been explored in the following section.

4.4 Modeling delay as the intrinsic characteristic of the circuit

Due to the shortcomings of the previous approach to characterize delay of a PSL circuit, a new way to compute delay as a function of only the intrinsic property of the PSL circuit is defined. In order to do so, as the simulation begins, the circuit is simulated for a predetermined number of time samples ($t1$) for a certain input combination ($h_{fix-bias}$). At $t1$ time instant, one of the controlling input p-bit values in the $h_{fix-bias}$ matrix is switched and the circuit is further simulated for an infinite number of time samples. The transition of the input p-bit value at $t1$ time sample is chosen such that it leads to a rising or falling transition at the output p-bit. The delay of the PSL circuit can be thus described as follows:

The simulation value (0/1) at the k^{th} time sample is denoted as Z_k.

The delay of the gate is denoted as τ (measured as number of samples).

The probability of output p-bit assuming logic '1' at any instant t can be given as follows:

$$P_t = \frac{\sum_{j=1}^{t} Z_j}{t} \tag{4.4}$$

The delay of the gate is defined as τ such that the probability P_t crosses 50% mark:

$$P_\tau < 50\% \tag{4.5}$$

Once the $h_{fix-bias}$ vector has been updated to reflect a transition at the output, the circuit is further simulated for an infinite number of time samples. In order to quantify delay as an intrinsic characteristic of the circuit under consideration, probability percentage of output state to be in logic-1 is computed for different number of time samples (T) starting from $t1$ time instant, taking one time sample at a time, increasing it to two samples and further increasing T until probability percentage crosses 50%. The computed percentage of the output state for each of those non-overlapping windows till $t1$ time instant and for each of those T different time intervals is plotted against corresponding time instance and the number of time samples taken by the circuit after $t1$ to cross 50% of the initial output value thus defines the delay of the PSL circuit.

This approach to calculate the delay of PSL circuits gets rid of any artifacts of the computational process and gives delay as an intrinsic characteristic of the circuit under consideration.

4.4.1 Revised comparison of fall delay of different input NAND gates

The fall propagation delay (t_{PHL}) characteristics of a $two-, three-, four-, five-, six-$, $seven-$, and an $eight$-input NAND gates implemented using two-input NAND gate as the fundamental building block have been compared. In order to replicate a falling transition at the output p-bit (that is the output transitions from a high-to-low value), one of the input p-bits is made to switch from a low-to-high value while all the other input p-bits are kept at a high value (non-controlling value). Using the methodology described above in order to incorporate the maximum delay experienced by a PSL circuit, the first input p-bit of the $h_{fix-bias}$ vector is switched and the delay for different input NAND gates is calculated for a non-overlapping window (W) of 100 time samples in width where $t1$ is 100000 time samples wide.

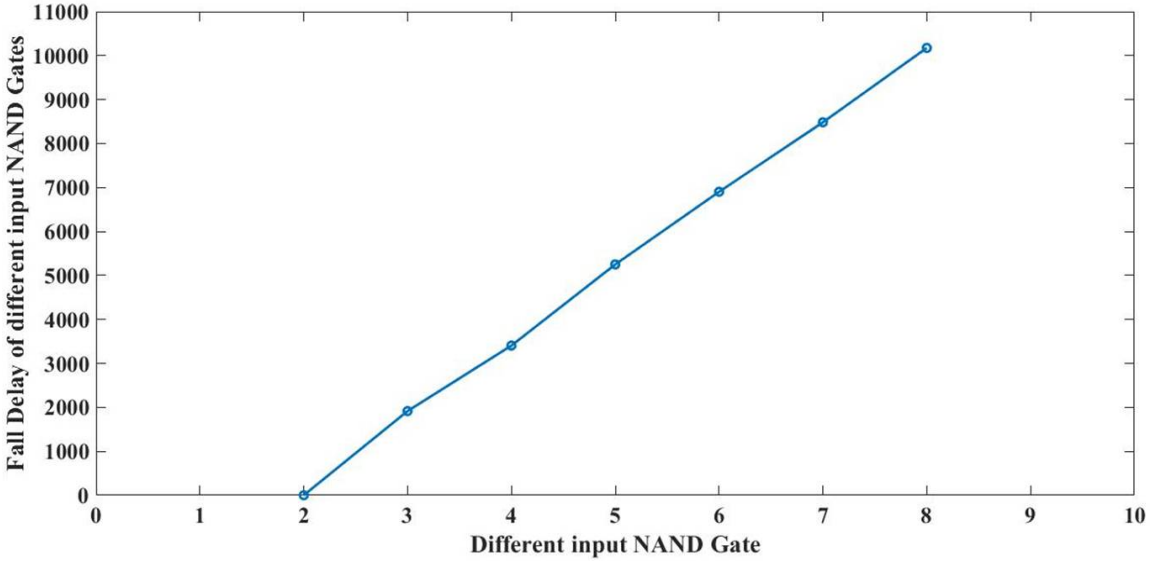


FIGURE 4.12: **Fall Delay:** The intrinsic fall delay of different input NAND gates plotted as a function of number of inputs of the gate. The fall propagation delay is observed to increase linearly with an increase in the number of inputs.

As observed from Fig. 4.12, the 8-input NAND gate (implemented using 21 p-bits) has the maximum delay while the 2-input NAND gate (implemented using 3 p-bits) has the minimum delay which falls in line with the expected behavior. Moreover, the fall delay increases approximately linearly with the number of inputs as we move from a 2-input NAND gate to an 8-input NAND gate due to increase in the number of resources (p-bits) used in designing the PSL circuits.

4.4.2 Revised comparison of rise delay of different input NAND gates

The rise propagation delay (t_{PLH}) characteristics of a $two-, three-, four-, five-, six-$, $seven-$, and an *eight*-input NAND gates implemented using *two*-input NAND gate as the fundamental building block have been compared. In order to replicate a rising transition at the output p-bit (that is the output transitions from a low-to-high value), one of the input p-bits is made to switch from a high-to-low value while all the other input p-bits are kept at a high value (non-controlling value). Using the methodology described above to incorporate the maximum delay experienced by a PSL circuit, the first input p-bit of the $h_{fix-bias}$ vector is switched and the delay for different input NAND gates is calculated for a non-overlapping window (W) of 100 time samples in width where $t1$ is 100000 time samples wide.

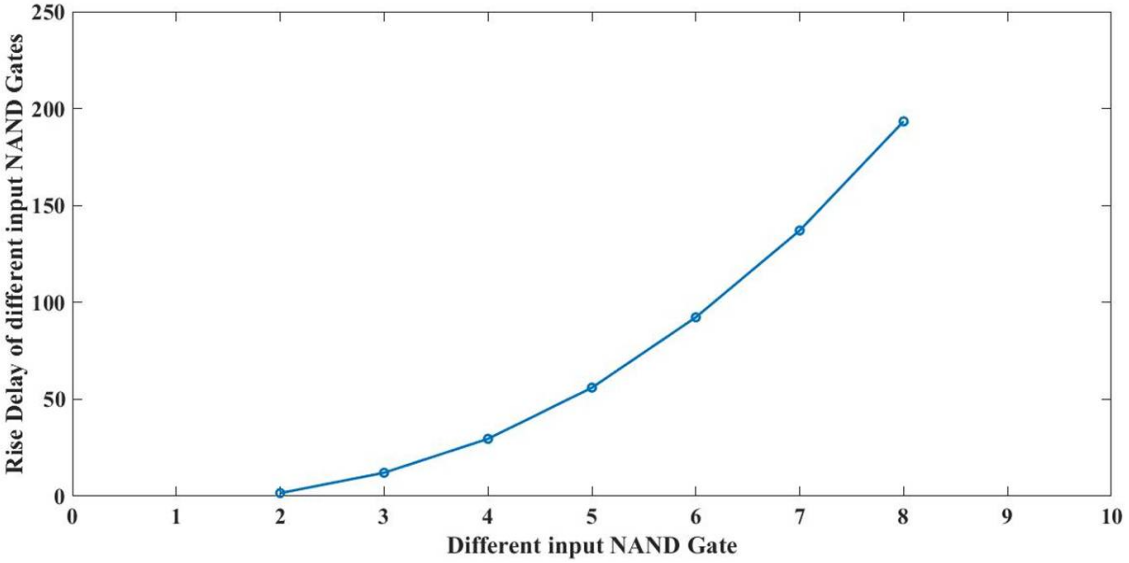


FIGURE 4.13: **Rise Delay:** The intrinsic rise delay of different input NAND gates plotted as a function of number of inputs of the gate. The rise propagation delay is observed to increase quadratically with an increase in the number of inputs.

Its is observed from Fig. 4.13, that the intrinsic rise delay of PSL circuits calculated using this approach increases approximately quadratically with the number of inputs as we move from a 2-input NAND gate to an 8-input NAND. This is in contrast with the constant rise delay observed using the sliding window approach. The different rate of increase in fall and rise delays as observed in Fig. 4.14 has been explored in greater depth in the following chapter.

	$NAND-2$	$NAND-3$	$NAND-4$	$NAND-5$	$NAND-6$	$NAND-7$	$NAND-8$
Fall Delay	1.8	1916.2	3405.2	5250.8	6903.4	8483.6	10174.6
Rise Delay	1.6	12.1	29.6	56	92.3	137.1	193.4

TABLE 4.5: Delay of different input NAND Gates as an intrinsic characteristic of the circuit.

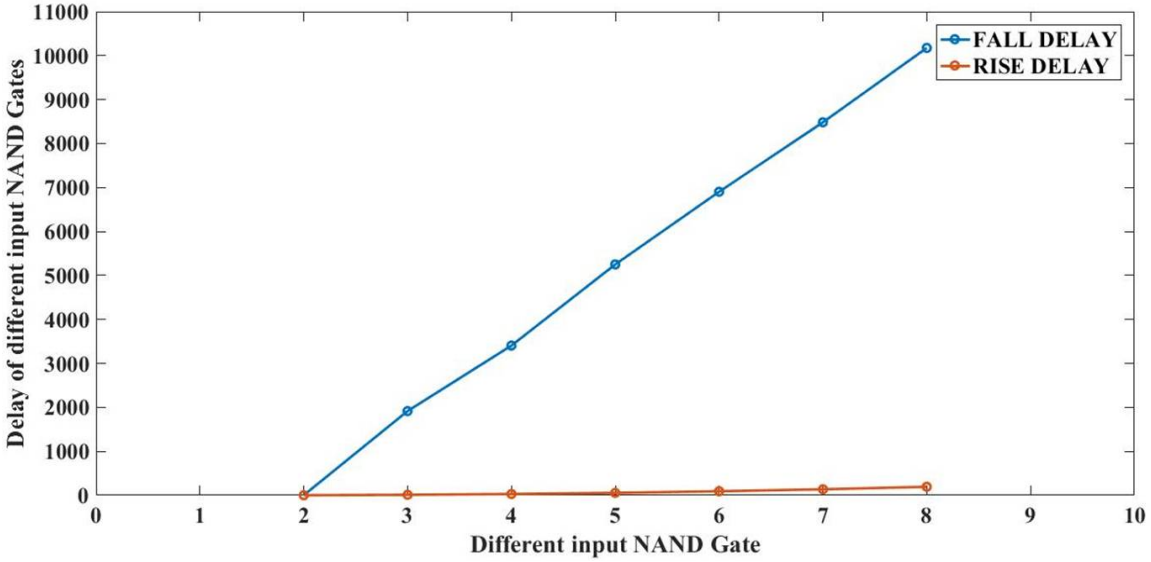


FIGURE 4.14: The intrinsic rise and fall delays of different input NAND gates plotted as a function of number of inputs of the gate. The rise propagation delay is observed to increase quadratically on contrary to the linear increase of fall delay with an increase in the number of inputs. Despite the linear increase, the fall delay is quite larger than the rise delay for respective input NAND gates.

Chapter 5

Modeling Delay in PSL circuits

Different rate of increase in fall and rise delays of logic gates implemented using PSL has been demonstrated in the previous chapter. The fall propagation delay increases linearly with increase in the number of inputs of a NAND gate. While the rise propagation delay increases quadratically with increase in the number of inputs of the NAND gate. The fall delay is greater than the rise delay in a NAND gate. Therefore, in this chapter the factors governing the delay are investigated and a model for the delay is proposed.

5.1 Fall Propagation Delay

The fall propagation delay of different input NAND gates has been observed to increase linearly with an increase in the number of inputs. To understand the behavior of the delay and factors that affect the delay, it is important to understand how modelling of p-bits is done in the simulation. Thus, a careful investigation in to the relative interaction among p-bits has been described using the set of equations used in this work to carry out the updation and simulation of p-bits. Consider the following set of equations used to update the p-bits of any given Boolean circuit:

$$sm(i,t) = sm(i,t) + (J(i,j) * m(j)) \tag{5.1}$$

where $sm(i,t)$ refers to an intermediate variable used to capture the strength of neighboring p-bits (m_j's) on the selected i^{th} p-bit calculated using the J matrix.

$$I(i,t) = I_0 * (h(i) + h_{fix}(i) + sm(i,t)) \tag{5.2}$$

where $I(i,t)$ is a variable used to store the interactions among p-bits as well as the influence of local and fixed bias on the p-bits. The parameters I_0, h and h_{fix} have been defined in equation 1.2.

$$m(i,t) = signum(r(i,t) + tanh(I(i,t))) \tag{5.3}$$

where $r(i,t)$ refers to a random number generated in the range of $(-1,+1)$. It is the signum of the sum of $r(i,t)$ and $tanh$ of $I(i,t)$ as per Eq. 5.3 that defines the updation of p-bits (m_i's).

Consider the case of a fall transition at the output of a three-input NAND gate implemented with PSL using 6 p-bits. The h_{fix} vector changes from $\begin{bmatrix}-6 & 6 & 6 & 0 & 0 & 0\end{bmatrix}^T$ to $\begin{bmatrix}6 & 6 & 6 & 0 & 0 & 0\end{bmatrix}^T$ after the first input is switched from a low-to-high value such that it reflects a high-to-low $(1-to-0)$ transition at the output and the corresponding values of variables $sm(i,t), I(i,t), m(i,t)$ have been recorded in table 5.1.

Time sample(t)	p-bit(i)	1	2	3	4	5	6
1	m(i,1)	-1	1	1	1	-1	1
	sm(i,1)	-3	-3	-1	0	-7	0
	I(i,1)	6	6	9	6	-6	3
2	m(i,2)	1	1	1	1	-1	1
	sm(i,2)	-3	-3	-1	0	-7	0
	I(i,2)	6	6	9	6	-6	3
3	m(i,3)	1	1	1	1	-1	1
	sm(i,3)	-3	-3	-1	0	-7	0
	I(i,3)	6	6	9	6	-6	3

TABLE 5.1: The order of updation of p-bits for a fall transition at the output.

The switching of the first input p-bit in the h_{fix} vector is reflected in $m(1,2)$ as it switches from -1 *to* 1, thus giving the corresponding values of $sm(1,2)$ and $I(1,2)$ as recorded in table 5.1. As observed from the above table, $sm(i,t)$ and $I(i,t)$ converge to constant values for different time samples. It is the constant value of $I(t)$ vector calculated using Eq. 5.2 that results in stable constant value of m_i's thus primarily defining the fall delay. As per Eq.5.3, only a random value greater than tanh(I(i,t)) in magnitude with an opposite sign generated the by the random number generator (RNM) can cause a sign reversal of the respective p-bit. As evident from $I(i,2)$ vector, as $I(6,2)$ has the lowest absolute value closest to zero, it would have the highest probability of switching the output p-bit m_6. In order for m_6 to switch, any random number generated in the range of $(-0.9951,-1)$ calculated by subtracting $tanh(3)$ from 1 could cause the output p-bit to switch. Likewise for m_1 or m_2 to switch, any random number generated should be in the range of $(-0.999988,-1)$ while it should be in the range of $(-0.99999998,-1)$ for m_3 to switch. Hence, if the desired random number is not generated to cause a

switching at the output p-bit, the PSL circuit gets stuck in a particular state. Thus, a higher fall delay is observed for a three-input NAND gate on comparison to a two-input NAND gate.

It is crucial to note that this behavior is not only a simulation artifact. It can even exist in the real systems, where p-bits continuously fluctuate between '0/1' logic state. The combination of p-bits in a particular way can lead to the PSL circuit getting stuck in a local minima. Thus, randomness of the p-bits or a particular random value generated in the specific range is needed to get the system out of the local minima. This highlights a very particular problem in PSL, which is not present in a conventional CMOS circuit.

Further it is observed that m_6 is always the first p-bit to flip and settle to its correct value among all the p-bits, provided if the random number is generated in that specific range of $(-0.9951, -1)$ to cause a transition at the output p-bit. Once m_6 settles to its correct value, the other intermediate p-bits start to converge to their correct value, one p-bit at a time, starting from the output p-bit towards the input p-bit until all p-bits settle to their correct value thus resulting in a higher delay. Due to increase in the number of intermediate p-bits with increase in number of inputs of a NAND gate; from 2 intermediate p-bits in a three-input NAND gate to 12 intermediate p-bits in an eight-input NAND gate; the fall delay of NAND gates increases with an increase in the number of inputs.

5.2 Rise Propagation Delay

The rise propagation delay of different input NAND gates has been observed to increase quadratically with an increase in the number of inputs. A careful investigation in to the relative interaction among p-bits reveals this difference in rise and fall delay of PSL circuits as follows. Consider the case of a rise transition at the output of a three-input NAND gate implemented with PSL using 6 p-bits. The h_{fix} vector changes from $\begin{bmatrix}6 & 6 & 6 & 0 & 0 & 0\end{bmatrix}^T$ to $\begin{bmatrix}-6 & 6 & 6 & 0 & 0 & 0\end{bmatrix}^T$ after the first input is switched from a high-to-low value such that it reflects a low-to-high (0-to-1) transition at the output and the corresponding values of variables $sm(i,t), I(i,t), m(i,t)$ have been recorded in table 5.2.

As $I(4,2)$ takes a value of '0', thus as per Eq. 5.3 the output p-bit m_6 is solely decided by the value generated by the random number generator. If the RNM generates a positive value, due to difference in sign as compared to its previous value, the output p-bit m_6 switches from -1 *to* 1 else if the RNM generates a negative value, m_6 retains the previous value thus giving rise to two cases:

Time sample(t)	p-bit(i)	1	2	3	4	5	6
1	m(i,1)	1	1	1	-1	1	-1
	sm(i,1)	1	1	1	-8	5	-4
	I(i,1)	-6	12	12	-6	12	-3
2	m(i,2)	-1	1	1	-1	1	-1
	sm(i,2)	1	3	1	-4	5	-4
	I(i,2)	-6	15	12	0	12	-3

TABLE 5.2: The order of updation of p-bits for a rise transition at the output.

$m(i,3)_a$	-1	1	1	-1	1	-1
$sm(i,3)_a$	1	3	1	-4	5	-4
$I(i,3)_a$	-6	15	12	0	12	-3

TABLE 5.3: (a) If RNM generates a negative value, which is the same case as $I(i,2)$.

$m(i,3)_b$	-1	1	1	1	1	-1
$sm(i,3)_b$	-3	-1	1	-4	-3	-4
$I(i,3)_b$	-12	9	12	0	0	-3

TABLE 5.4: (b) If RNM generates a positive value, it further gives rise to the following four cases:

$m(i,4)_c$	-1	1	1	-1	-1	-1
$sm(i,4)_c$	1	3	3	4	5	0
$I(i,4)_c$	-6	15	15	12	12	3

TABLE 5.5: (c) If RNM generates negative values for both m_4 and m_5 p-bits

$m(i,4)_d$	-1	1	1	-1	1	-1
$sm(i,4)_d$	1	3	1	-4	5	-4
$I(i,4)_d$	-6	15	12	0	12	-3

TABLE 5.6: (d) If RNM generates a negative value for m_4 and a positive value for m_5

$m(i,4)_e$	-1	1	1	1	-1	-1
$sm(i,4)_e$	-3	-1	3	4	-3	0
$I(i,4)_e$	-12	9	15	12	0	3

TABLE 5.7: (e) If RNM generates a positive value for m_4 and a negative value for m_5

$m(i,4)_f$	-1	1	1	1	1	-1
$sm(i,4)_f$	-3	-1	1	-4	-3	-4
$I(i,4)_f$	-12	9	12	0	0	-3

TABLE 5.8: (f) If RNM generates positive values for both m_4 and m_5 p-bits

Likewise each of the above state gives rise to other states and thus the system keeps on toggling among all possible states until all p-bits (m_i's) settle to their correct value and the correct state is found. It is interesting to note that it is the first intermediate p-bit that settles to its correct value right after the input p-bit. The order of updation

of p-bits continues from the input to the output p-bit on contrary to the reverse order of updating p-bits as observed during the falling transition at the output. Also, unlike during the fall propagation delay calculation, the PSL circuit never gets stuck in a particular state during rise delay calculation of different input NAND gates. As the PSL circuit never gets stuck in a particular state, the rise delay of different input NAND gates is observed to be much lesser than the fall delay of corresponding different input NAND gates. Furthermore, due to increase in the number of intermediate p-bits with increase in number of inputs of a NAND gate (from 2 intermediate p-bits in a three-input NAND gate to 12 intermediate p-bits in an eight-input NAND gate) the rise delay of NAND gates increases with an increase in the number of inputs as well.

5.3 Energy analysis of possible states

Neural networks with symmetric connections among the neurons determined by a weight matrix W_{ij} is analogous to classical statistical physics having symmetric natural connections among the interacting particles. As per Boltzmann law, the probability of a particular configuration α can be defined in terms of its associated energy E_α as follows [1]:

$$P_\alpha = \frac{1}{Z}\exp(-E_\alpha) \tag{5.4}$$

$$E_\alpha = -\{m\}_\alpha^T[W]\{m\}_\alpha \tag{5.5}$$

Where T denotes transpose and constant Z is chosen such that all P_α's add up to one. Taking inspiration from neural networks and extending the same to PSL circuits where the symmetric J_{ij} matrix defines the interaction strength between p-bits, the probability ($P\{m_i\}$) of each of the possible states m_i in terms of its associated energy ($E\{m_i\}$) is defined as follows:

$$P\{m_i\} = \frac{1}{Z}\exp(-E\{m_i\}) \tag{5.6}$$

$$E\{m_i\} = -I_0 * ((\frac{1}{2}\{m_i\}[J]\{m_i\}^T) + (\{m_i\}[h]^T) + (\{m_i\}[h_{fix}]^T)) \tag{5.7}$$

where Z is chosen such that all the probabilities $P\{m_i\}$'s add up to one. A two-input NAND gate containing *three* p-bits has a total of *eight* possible states when simulated using the h and J matrices defined as per Eq. 3.2. Fig. 5.1 depicts that only the possible valid states occupy the lowest energy level as compared to the invalid states thus further supporting the interconnection weights $[J_{ij}]$ taken to simulate the gate.

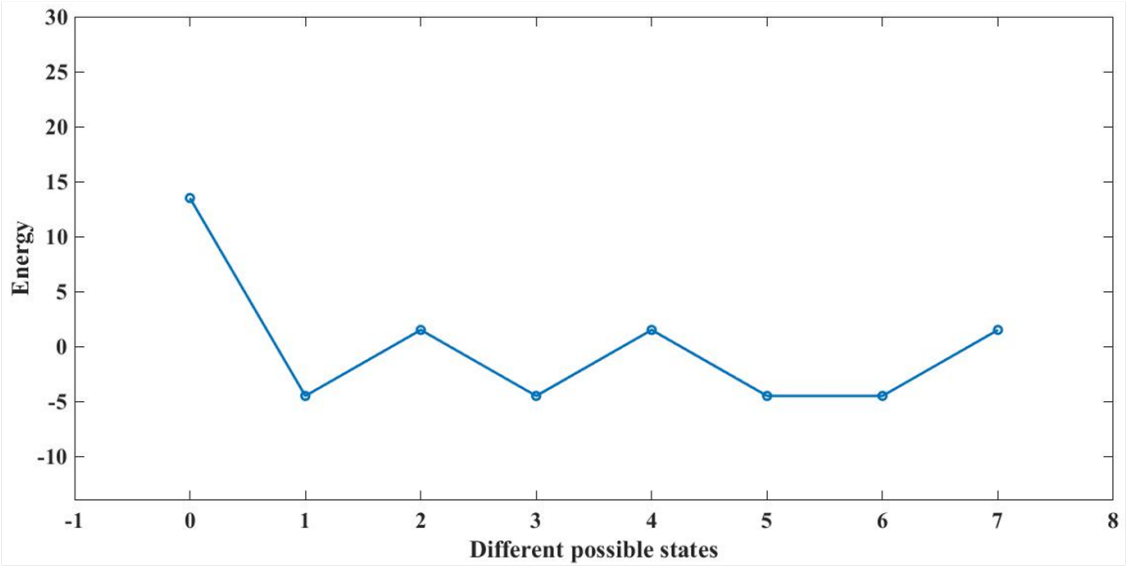


FIGURE 5.1: **Energy spectrum of NAND gate:** Energy spectrum of 2-input NAND gate as per Eq. (5.7) depicts that only logically correct states of the 2-input NAND gate occupy the lowest energy levels.

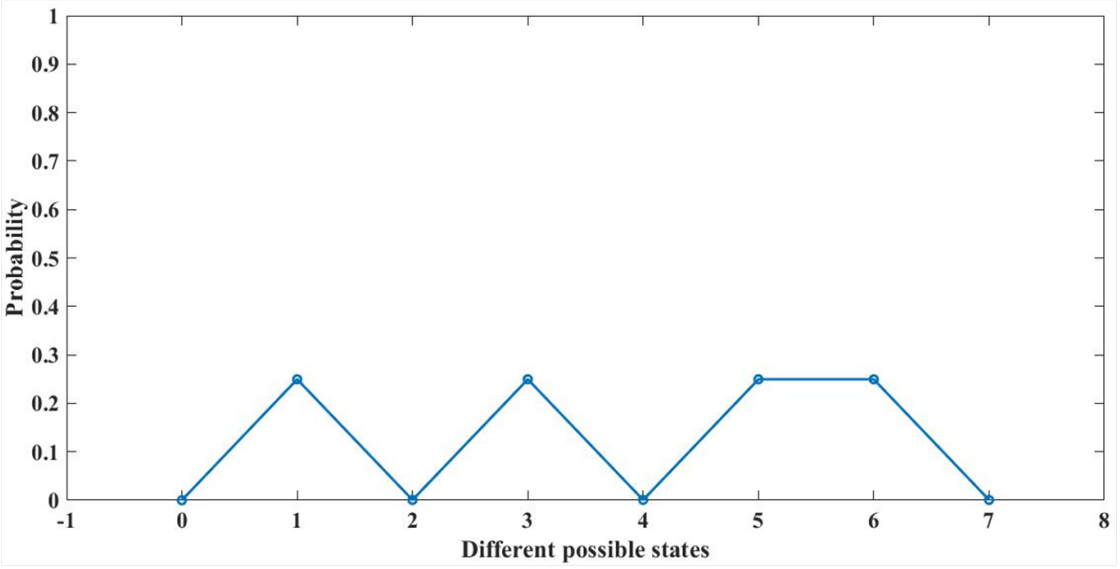


FIGURE 5.2: **Probability spectrum of NAND gate:** Probability spectrum of 2-input NAND gate as per Eq. (5.6) depicts that only logically correct states of the 2-input NAND gate have the highest probability of occurrence with the probability being equally divided among the four possible states.

5.4 Timing Model of Delay

The intrinsic rise propagation delay of different input NAND gates implemented using PSL is observed to increase approximately quadratically with increase in number of inputs from a 2-input NAND gate to an 8-input NAND as shown in Fig. 4.13. The quadratic equation

$$y = 4.6 * x^2 - 14 * x + 12 \tag{5.8}$$

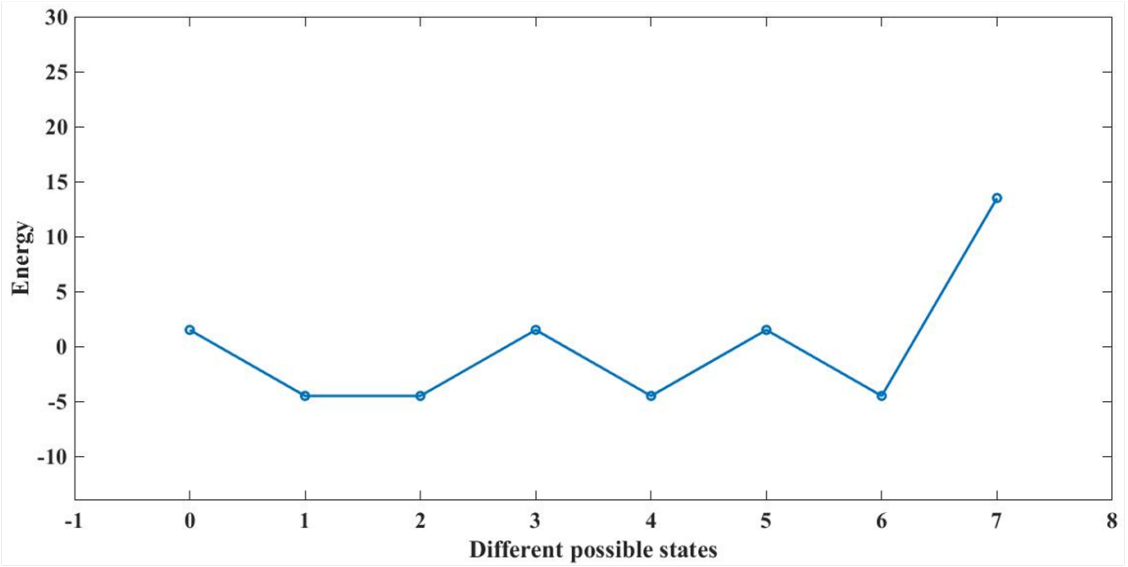


FIGURE 5.3: **Energy spectrum of NOR gate:** Energy spectrum of 2-input NOR gate as per Eq. (5.7) depicts that only logically correct states of the 2-input NOR gate occupy the lowest energy levels.

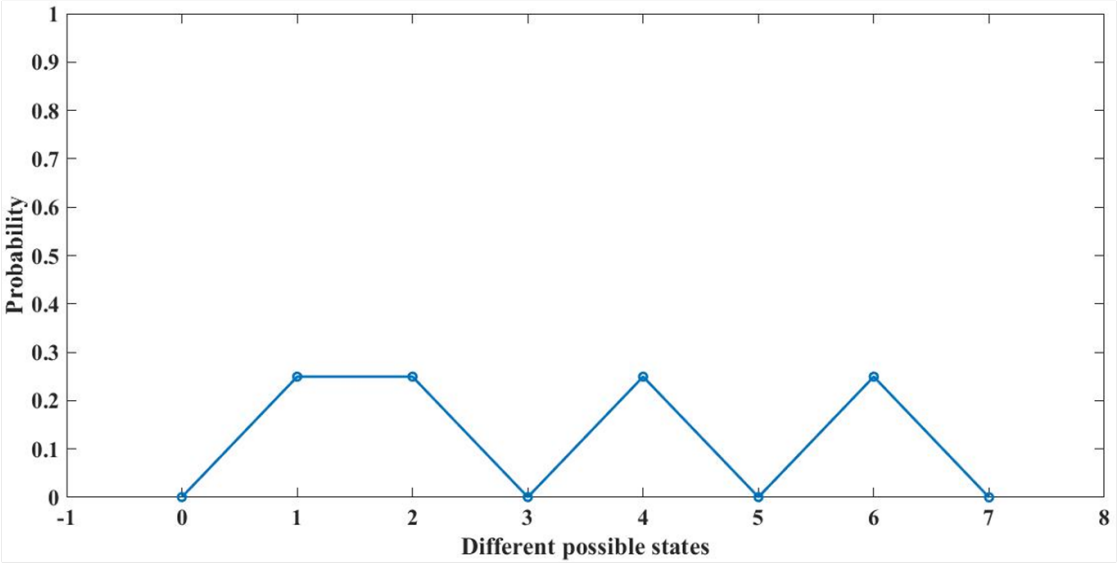


FIGURE 5.4: **Probability spectrum of NOR gate:** Probability spectrum of 2-input NOR gate as per Eq. (5.6) depicts that only logically correct states of the 2-input NOR gate have the highest probability of occurrence with the probability being equally divided among the four possible states.

closely fits the observed rise delay pattern as shown in Fig. 5.8. The value of these coefficients have been determined by the least square error method. This could be used in synthesis of circuits using the given equation to predict the rise delay.

However, the intrinsic fall propagation delay of different input NAND gates implemented using PSL is observed to increase linearly with increase in number of inputs as shown

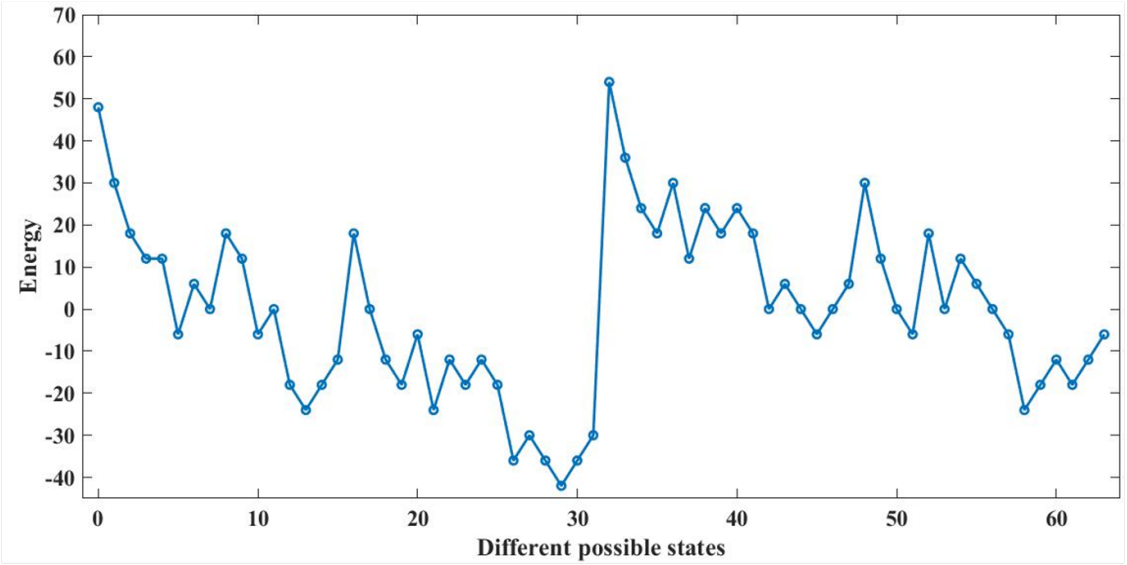


FIGURE 5.5: **Energy spectrum of NAND-3 gate:** Energy spectrum of 3-input NAND gate as per Eq. (5.7) for the three inputs clamped at logic 0, 1 and 1 respectively. It depicts that only one logically correct state of the 3-input NAND gate occupy the lowest energy level among all possible states.

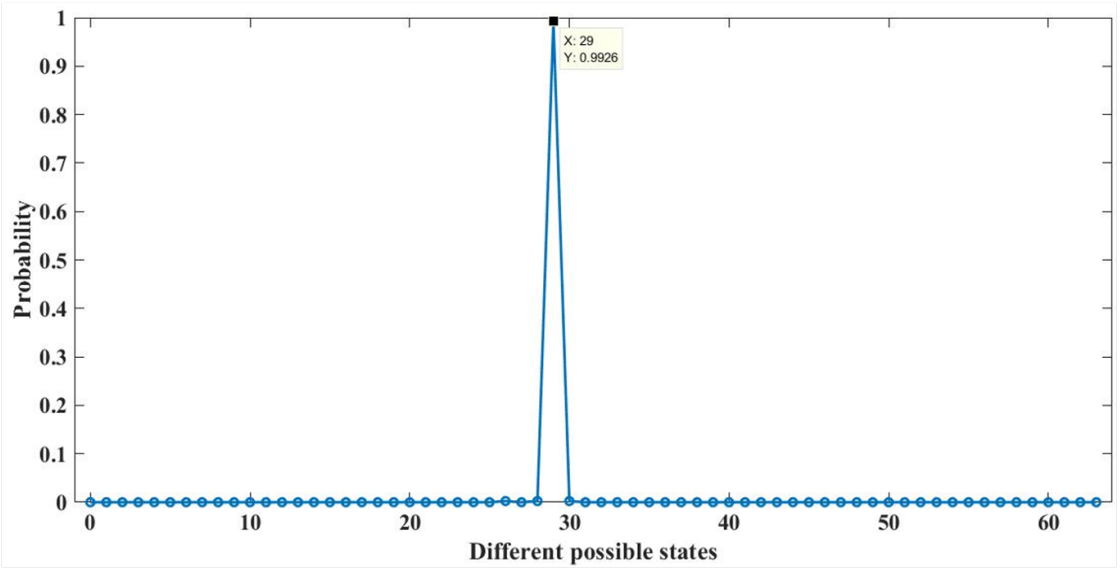


FIGURE 5.6: **Probability spectrum of NAND-3 gate:** Probability spectrum of 3-input NAND gate as per Eq. (5.6) for the three inputs clamped at logic 0, 1 and 1 respectively depicts that only one logically correct state of the 3-input NAND gate has the highest probability of occurrence among all possible states.

in Fig. 4.12. The linear equation

$$y = 1680 * x - 3260 \tag{5.9}$$

closely fits the observed fall delay pattern as shown in Fig. 5.7. The value of these coefficients have been determined by the least square error method. This could be used

in synthesis of circuits using the given equation to predict the fall delay. An analytical way to find these coefficients can be undertaken as a future work.

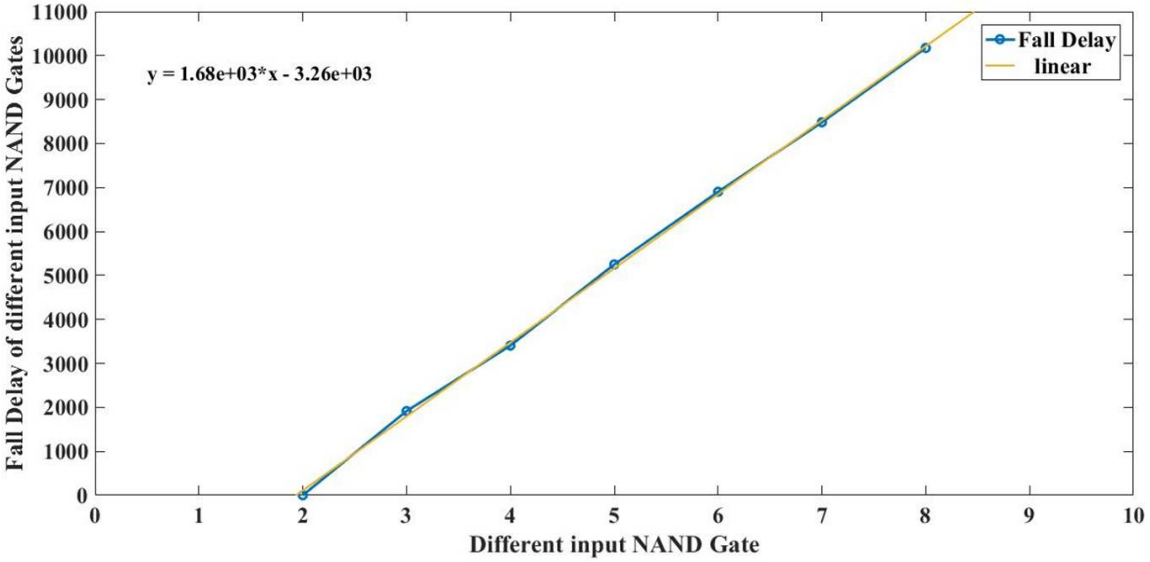


FIGURE 5.7: **Fall Delay:** Linear behaviour of fall propagation delay of different input NAND gates.

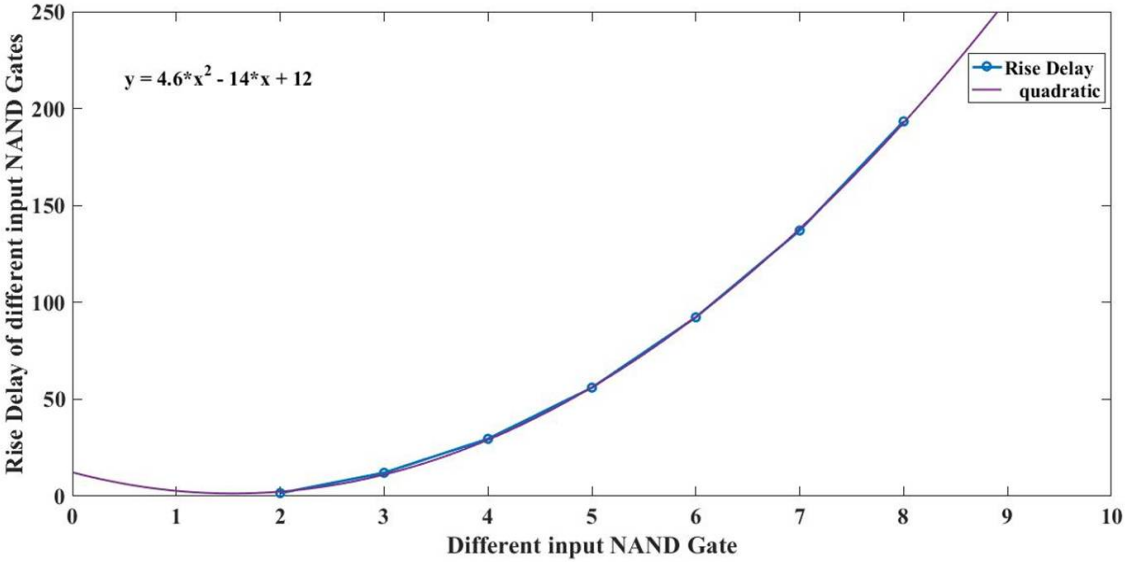


FIGURE 5.8: **Rise Delay:** Quadratic behaviour of rise propagation delay of different input NAND gates.

5.5 Possible Method to reduce the Fall Delay

One of the possible methods to reduce the delay is to get rid of the intermediate p-bits all together. Due to absence of concrete Ising Hamiltonian equations for bigger gates, the task of carefully selecting the h and J matrices becomes dauntingly difficult such that

the functionality of the gate is also maintained. Using the proposed Ising Hamiltonian of a three-input NAND gate void of any intermediate p-bits as per Eq. 3.10, and refining it further by taking the J matrix as half of the J matrix defined by Eq. 3.11 and keeping the h vector as same, the functionality is maintained and the delay is reduced simultaneously. The h and J-matrices for an improved 3-input NAND gate are defined as follows:

$$h_{NAND3} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \end{bmatrix} \qquad J_{NAND3} = \begin{bmatrix} 0 & -0.5 & -0.5 & -1 \\ -0.5 & 0 & -0.5 & -1 \\ -0.5 & -0.5 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \tag{5.10}$$

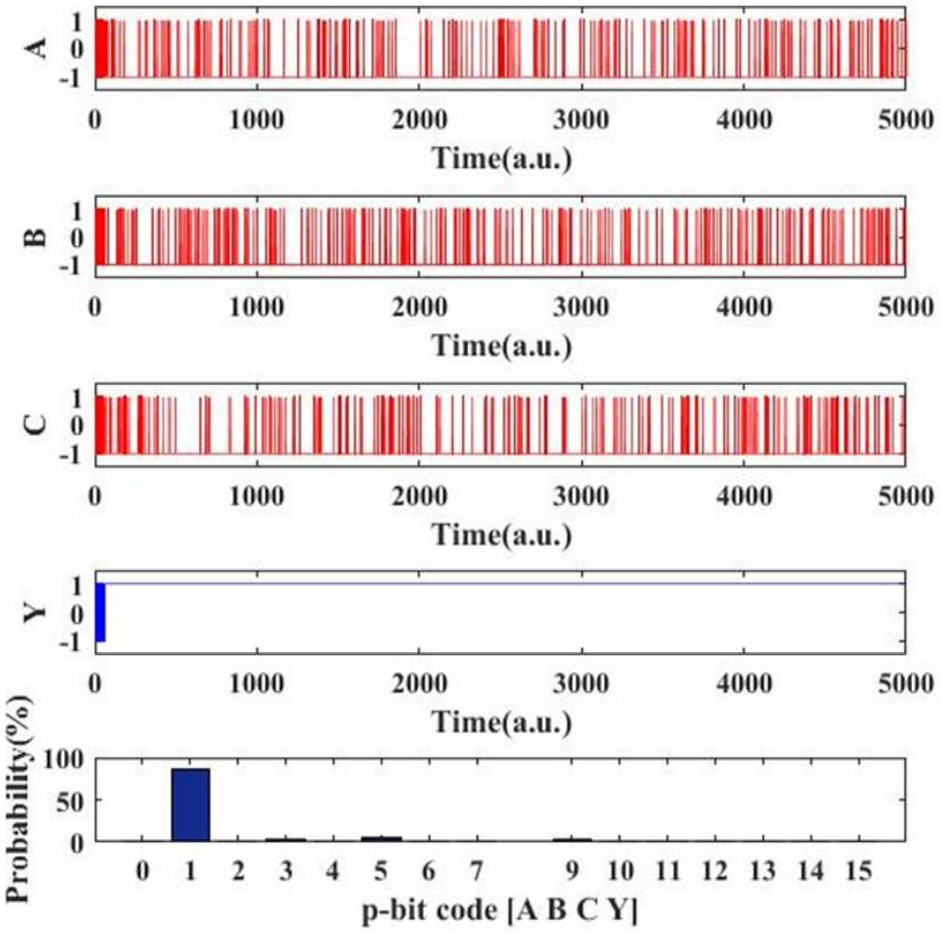


FIGURE 5.9: **3-input NAND gate without any intermediate p-bits:** Simulation results of the three-input NAND gate where inputs A, B and C are all clamped to logic 0 while output Y is left floating. It is found that despite the given biasing of inputs, the output Y stays in logic 1-state with a high probability. The histogram plotted has been obtained by averaging over 5000 time samples.

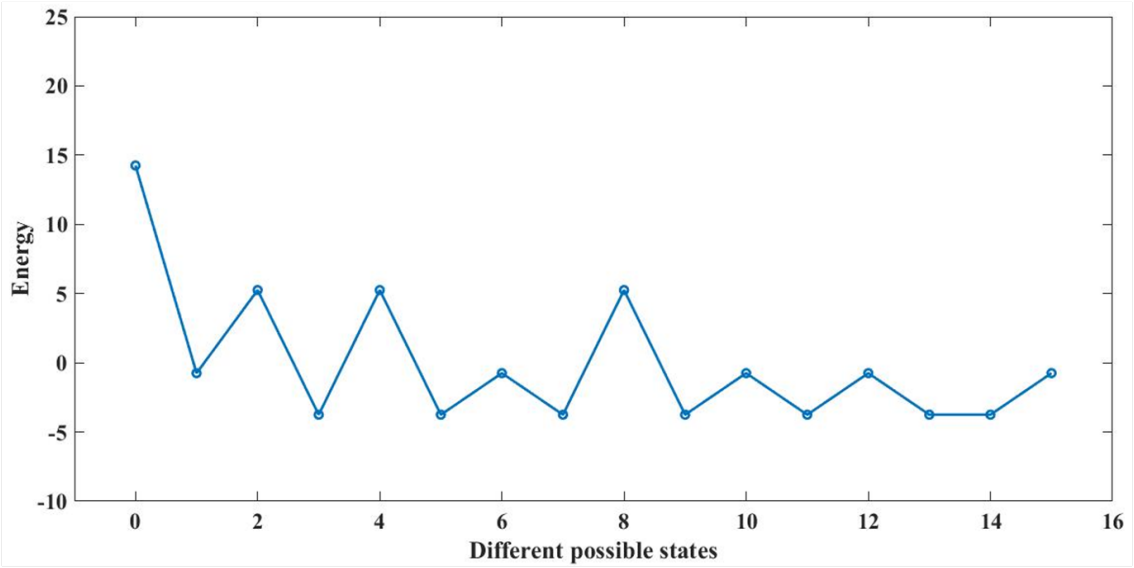


FIGURE 5.10: Energy spectrum of 3-input NAND gate as per Eq. (5.7) implemented without any intermediate p-bits in PSL depicts that all but one logically correct state of the 3-input NAND gate occupy the lowest energy levels.

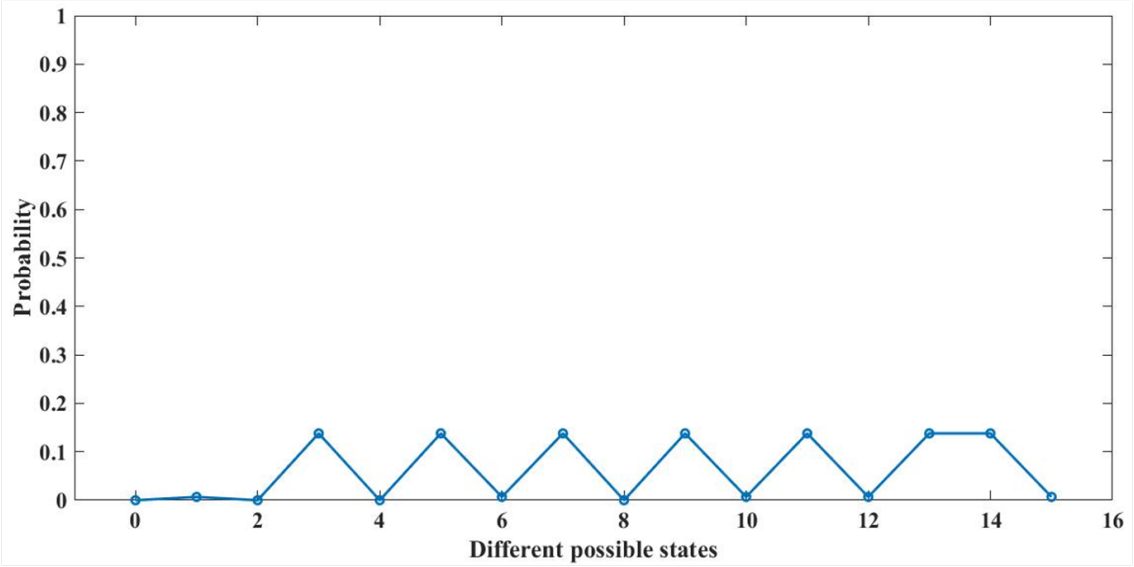


FIGURE 5.11: Probability spectrum of 3-input NAND gate as per Eq. (5.6) implemented without any intermediate p-bits in PSL depicts that all but one logically correct state of the 3-input NAND gate have the equal probability of occurrence with the probability being equally divided among the seven possible states.

This approach can be used to reduce the delay of a PSL circuit. In the forward mode of operation, it will always work correctly as shown in Fig. 5.9. But if the PSL circuit is free willed, then it could possibly give an inaccurate result and needs futher investigation.

	Fall Delay	*Rise Delay*
$NAND-3$	2.5	2.1

TABLE 5.9: Delay of 3-input NAND Gate implemented without any intermediate p-bits

Chapter 6

Conclusion

In this work, the timing analysis of Boolean functions implemented using PSL has been investigated. The synthesis of complex gates using basic and universal gates has been performed using PSL and trade-offs between different implementations have been carefully studied. An attempt to synthesize sequential circuits by implementing a latch in PSL has been explored too but needs further investigation.

This research primarily focused on the characterization of delay of different Boolean functions and different factors that govern it. A few methods to characterize the delay as a function of intrinsic property of the circuit have been proposed. A careful investigation of the mechanics of the PSL circuits reveals the probable reasons for different rise and fall propagation delays of different input NAND gates. An important observation found in this work is that, for certain transition of inputs in a PSL circuit, the circuit can be locked into a particular wrong or intermediate state for a long duration as per table 5.1. To take the circuit out of this intermediate state, the randomness of the p-bits is required. This is a distinct problem of PSL compared to conventional digital circuits.

Bibliography

[1] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, "Stochastic p-bits for invertible logic," *Phys. Rev. X*, vol. 7, p. 031014, Jul 2017.

[2] V. Q. Diep, B. Sutton, B. Behin-Aein, and S. Datta, "Spin switches for compact implementation of neuron and synapse," *Applied Physics Letters*, vol. 104, no. 22, p. 222405, 2014.

[3] B. Behin-Aein, V. Diep, and S. Datta, "A building block for hardware belief networks," *Scientific Reports*, vol. 6, p. 29893, Jul 2016.

[4] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, pp. 13–24, Dec 2012.

[5] A. Sengupta, Y. Shim, and K. Roy, "Proposal for an all-spin artificial neural network: Emulating neural and synaptic functionalities through domain wall motion in ferromagnets," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, pp. 1152–1160, 2016.

[6] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "Ising computer," *Hitachi Review*, vol. 65, 2016.

[7] J. D. Biamonte, "Nonperturbative k-body to two-body commuting conversion hamiltonians and embedding problem instances into ising spins," *Phys. Rev. A*, vol. 77, p. 052331, May 2008.

[8] L. Personnaz, I. Guyon, and G. Dreyfus, "Collective computational properties of neural networks: New learning mechanisms," *Phys. Rev. A*, vol. 34, pp. 4217–4228, Nov 1986.

[9] B. Sutton, K. Y. Camsari, B. Behin-Aein, and S. Datta, "Intrinsic optimization using stochastic nanomagnets," *Scientific Reports*, vol. 7, p. 44370, Mar 2017.

Publications based on this work

Ishan Bhatia and Sneh Saurabh, "Investigation of Timing of Logic Gates realized using Probabilistic Spin Logic (PSL)", to be submitted in *IEEE, Access*, 2019.

Brief Biography of the author

Ishan Bhatia was born in February, 1993. He did his schooling from Delhi Public School, Dwarka and completed his B.Tech from Bhagwan Parshuram Institute of Technology (BPIT), GGSIPU, Delhi in the field of Electronics and Communication Engineering. He is pursuing his M.Tech in ECE at Indraprastha Institute of Information Technology, Delhi with specialization in VLSI and Embedded System design. He is currently working as an RnD Engineer in Memory Designing at Synopsys, Noida. He can be reached at **ishan17092@iiitd.ac.in** or **ishanbhatia36@gmail.com**.