AN INNOVATIVE APPROACH TO DETECT GLITCHES IN HARDWARE IMPLEMENTATIONS ON FPGAS

by

Kinjalben Shah
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

[illegible]	Dr. Jens-Peter Kaps, Thesis Director
KGaj	Dr. Kris Gaj, Committee Member
Alok Berry	Dr. Alok Berry, Committee Member
[illegible]	Dr. Andre Manitius, Chairman, Department of Electrical and Computer Engineering
[illegible] for KSB	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: 1/17/13	Spring Semester 2013 George Mason University Fairfax, VA

An Innovative Approach to Detect Glitches in Hardware Implementations on FPGAs

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science at George Mason University

By

Kinjalben Shah
Bachelor of Engineering
Gujarat University, 2006

Director: Jens-Peter Kaps, Professor
Department of Electrical and Computer Engineering

Spring Semester 2013
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my parents, for helping me start off with a good education from which all else springs...

Acknowledgments

I would like to take this opportunity to express my gratitude towards all who supported me during this thesis.

Dr. Jens-Peter Kaps, for believing in me, keeping faith in my work and for giving me an opportunity to work on this research project. I cannot thank him enough for taking his precious time out of his busy schedule to discuss the problems and ideas related to this research and guiding me towards the goal.

Cryptography Engineering Research Group (CERG) team members, for their suggestions and ideas. In particular, I would like to thank *Rajesh Velegalati* for his support in trubleshooting issues and clarifying my doubts in need.

Tejas Doshi, my supportive and patient husband, who gave me strength to persevere through what seemed like a totally overwhelming and never-ending task. I couldn't have done it without him.

And at last, I would like to thank my beloved *problems* for directing me to the goal of a better and a solid approach.

Table of Contents

List of Tables

List of Figures

Abstract

AN INNOVATIVE APPROACH TO DETECT GLITCHES IN HARDWARE IMPLEMENTATIONS ON FPGAS

Kinjalben Shah, MS

George Mason University, 2013

Thesis Director: Dr. Jens-Peter Kaps

Glitches are unproductive signal transitions due to unbalanced path delays at the inputs of a gate. Unlike asynchronous circuits, the functionality of synchronous circuits is not significantly affected by the presence of glitches. Despite of that, detection of glitches in synchronous designs is important because the number of transitions increases due to glitches and the dynamic power dissipation linearly depends on the number of transitions. Glitches contribute to the dynamic power which itself is a major portion of the total power consumed by the design. Consequently, in presence of glitches, the overall power dissipation of the design increases.

With the increased use of small battery powered devices, low power consumption has become a highly important concern for the designs. Many advantages of FPGAs like rapid prototyping, low cost and field re-programmability are offset by higher power consumption than ASICs. Therefore, detecting glitches that occur in FPGA designs is important in order to understand how much power is consumed by glitches.

The accuracy of the post place and route simulation mainly depends on the simulation model for a given family of FPGAs and does not take process variations into account. Therefore, simulation might not detect all the glitches in the design. This research provides a novel approach for glitch detection in hardware implementations on FPGAs. We designed a circuit that does not merely detect the presence of glitches or counts the number of spurious transitions that a glitch causes, our circuit also captures the glitch waveform and provides information about the position and the width of glitches. We also propose a methodology to increase the resolution of the captured waveform. From our proposed glitch detection method we can reliably reproduce the glitch waveform and can detect glitches with a width of as small as **324 ps** on Spartan-3E FPGA.

Chapter 1: Introduction

Field Programmable Gate Arrays (FPGAs) are highly desirable for the implementation of digital systems due to their flexibility and low time-to-market. FPGAs have many more advantages, such as rapid prototyping, field re-programmability, growing density and speed, simple design cycle and low cost for small volume, which makes them more viable alternative for many of the current *Application Specific Integrated Chip (ASIC)* applications. However, the advantages of FPGAs are offset by higher power consumption than ASICs. It is estimated that an FPGA design consumes overall 10 times more power than a functionally equivalent ASIC design [1].

1.1 Power in Digital Circuits

Power has become a significant design constraint in digital designs due to the demand of battery-powered devices in personal wireless communications and other portable digital applications. If advantages of FPGAs are to be fully exploited, the amount of power they consume must be reduced or carefully controlled. High power dissipation causes overheating, which degrades performance and reduces chip lifetime. To control their temperature levels, high power chips require specialized and expensive packaging and heat-sink arrangements. Reducing the power consumption of FPGAs leads to lower packaging and cooling costs as well as improved reliability.

For any *Complementary Metal-Oxide Semiconductor (CMOS)* circuit, total power consumption can be divided into two sources of power:

1. Static Power Consumption

2. Dynamic Power Consumption

1. **The Static Power Consumption** is the power that is dissipated when the device is powered up but in the idle state. The major component of static power is the leakage current that leaks either from source to drain or through the gate oxide, even if the transistor is off. The leakage current depends on temperature and process variation. The static power consumption is calculated by equation (1.1),

$$P_{static} = V_{DD} * I_{leakage} \tag{1.1}$$

   Where, V_{DD} is the Supply Voltage and $I_{leakage}$ is the Leakage Current.

2. **The Dynamic Power Consumption** is the power that is dissipated when a signal transition occurs at gate outputs. Sources of dynamic power consumption can be further divided in to two parts:

   (a) Short Circuit Power

   (b) Switching Power

   Short Circuit Power refers to the power dissipated when a direct current path exists from V_{DD} to GND. When a transition occurs at a gate output, for short amount of time direct current path exist between V_{DD} and GND. Short circuit power accounts for only about 10% of dynamic power [2], therefore the majority of dynamic power dissipation comes from switching power. *Switching Power* is given by equation (1.2),

$$P_{SW} = 0.5 * SA * C * V_{DD}^2 * f \tag{1.2}$$

   Where, SA is Switching Activity, C is the Effective Capacitance, V_{DD} is the Supply Voltage and f is the Operating Frequency.

Traditionally static power has been overshadowed by dynamic power consumption, but as transistor sizes continue to shrink, static power may overtake dynamic power consumption

Table 1.1: Power Consumption in a CMOS Inverter

Input Transitions	**Output**	$\mathbf{C_{load}}$
$0 \rightarrow 0$	1	No change
$0 \rightarrow 1$	$1 \rightarrow 0$	Discharge
$1 \rightarrow 0$	$0 \rightarrow 1$	Charge
$1 \rightarrow 1$	0	No change

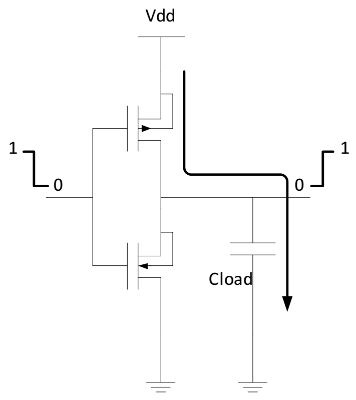


Figure 1.1: CMOS Charging

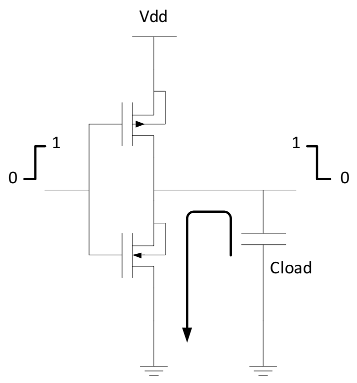


Figure 1.2: CMOS Discharge

[3] [4]. Despite the rising significance of static power in CMOS circuits, the majority of power dissipation in todays digital designs comes from dynamic power dissipation.

Signal transitions (logic level 0 - *low* to logic level 1 - *high* or vice versa) make up the *Switching Activity (SA)* of a circuit. Four types of transitions can occur on the gate output of binary logic. Input transitions and corresponding power consumption of a CMOS inverter is shown in Table 1.1.

A signal transition from low to high is a power consuming transition, because in this case the load capacitor is charged directly from the supply voltage and therefore power from supply is consumed by the circuit as shown in Fig. 1.1. On high to low transition the load capacitor is discharged and the energy s dissipated as shown in Fig. 1.2. Signal transitions from low to low and high to high do not consume any dynamic power.

1.2 Glitches

Signal transitions can be classified into two types:

- Functional transitions
- Spurious transitions

Functional transitions are the signal transitions necessary to perform the required logic function. The *spurious transitions* are glitches, which are the unproductive signal transitions due to unbalanced path delays at the inputs of a gate, causing the gate's output to transition briefly to an intermediate state.

Glitches can be further divided into two categories:

- Generated glitches
- Propagated glitches

If the input signals to a gate are skewed in time, a glitch may be *generated* and if a glitch arrives to an input of a gate and that input is sensitive at that moment, the glitch is *propagated* at the output of a gate. The number of propagated glitches is far bigger than the number of generated glitches [5].

We can see from the Fig. 1.3 that if input x2 arrives little bit later than x1 at the input of an XOR gate, a glitch is generated. This glitch is propagated depending on the gate. An inverter always propagates glitches. It can neither generate nor eliminate a glitch. On the other hand, an AND gate can either propagate or eliminate glitch propagation depending on the other input. If x3 at the input of an AND gate is always 0, any glitches will be eliminated, because 0 is the dominant input for an AND gate. With 1 at the x3 input of an AND gate, it may or may not propagate a glitch depending on the arrival time of x3. When x3 transits from 0 to 1 before the arrival of the glitched input of an AND gate, the glitch is propagated else it is eliminated.

Whether a transition is useful or useless depends on other possible transitions in the same clock cycle. Ideally, in a synchronous design, every net has either zero or one transition per

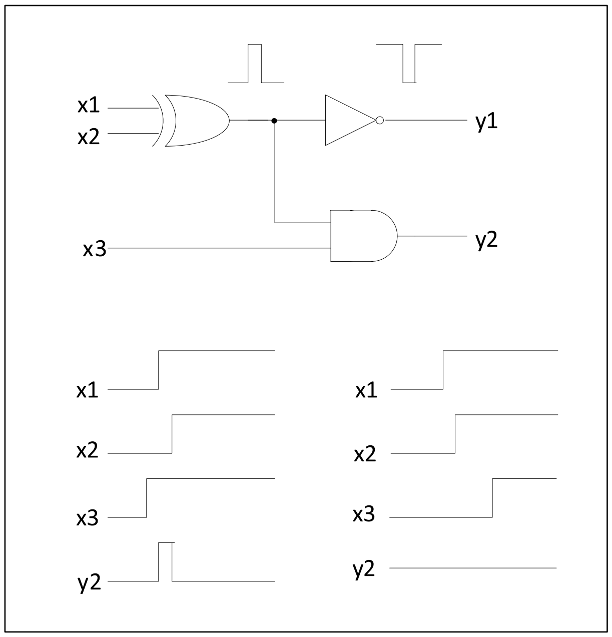


Figure 1.3: Generated and Propogated Glitches

clock cycle. Unfortunately, on any given net there is often more than one signal transition per clock cycle. If signal makes an odd number of transitions per clock period, then the final state of the signal is different than the original state. Only the final signal transition is considered to be useful and all other transitions are useless transitions. If signal makes even number of transitions per clock period, there is no change in the final state of the signal from the initial state. In this case, all transitions will be useless. This is shown in the Fig. 1.4.

Unlike asynchronous circuits, the functionality of synchronous circuits is not significantly affected by the presence of glitches. Despite of that, detection of glitches in synchronous designs is important as they have a significant effect on power consumption. As seen from equation (1.2) the dynamic power dissipation linearly depends on the switching activity of the signal and glitches increase the switching activity. Hence, the more glitches are in the circuit, the more dynamic power is dissipated.

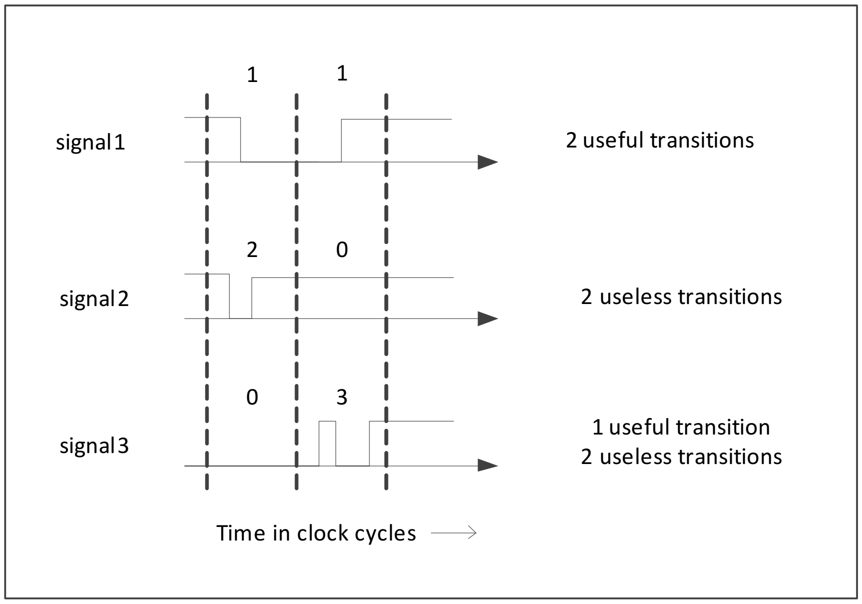


Figure 1.4: Useful and Useless Transitions

1.3 Effect of Glitches in FPGAs

The flexibility and re-programmability provided by FPGAs comes at the cost of higher power consumption. Based on the study in [6], glitch power (glitch related dynamic power) can be 60% of the dynamic power consumed in the FPGAs. The work in [8] and [9] estimates that glitch power in FPGAs comprises from 4% to 73% of total dynamic power, with an average of 22.6%. Dynamic power dissipation is a major factor of total power dissipation in FPGAs. A study that examined power dissipation in a commercial 90nm FPGA found that dynamic power accounted for 62% of total power [7].

Analysis of FPGA power consumption in [10] shows that power dissipation in FPGA devices is predominantly in programmable interconnect network. The authors used Xilinx XC4003A, fabricated in a 0.6 um, 2-layer metal process for the analysis. The study shows that the amount of power consumed by the interconnect can account for up to 65% of total dissipated power [10]. While glitches are not unique to FPGAs, the relatively high capacitive loading of the programmable interconnect places a much higher power cost on glitches in FPGAs.

Estimating the glitches that occur in FPGA designs is important in order to understand how much power is consumed by glitches.

1.4 Effect of Glitches in Secure Designs

Side-channel analysis exploits information leaked during the computation of a cryptographic algorithm. The most common technique is to analyze the power consumption of a cryptographic device using *Differential Power Analysis (DPA)* [11]. This side-channel attack exploits the correlation between the instantaneous power consumption of a device and the intermediate results of a cryptographic algorithm. Glitches dissipate extra information in terms of power, which may make side channel analysis easier.

To counteract DPA attacks two methods are widely used, Masking and Hiding. *Masking* randomizes the intermediate values. *Hiding* makes power consumption independent of data processed. In presence of glitches, these techniques are still vulnerable to DPA attacks. Research in [11] shows that the masking circuits are susceptible in presence of glitches. Dual-rail is a countermeasure based on hiding. The principle of this countermeasure is to generate a design equivalent and with opposite behavior of the target design such that every part of the circuit is perfectly balanced. This way the activity of the design remains constant and completely independent of the data processed. However, glitches can cause an imbalance in the power consumption in dual-rail circuits [14].

Glitches have shown to be a source of side-channel leakage [12], because the presence of a glitch depends on the specific input data pattern on the circuit. Further, the arrival time of signals at gate inputs can cause small data-dependent variations on the switching time of gates [13]. This variation shows up in the power-consumption pattern and can be exploited in power analysis attacks.

1.5 Previous Work

Several techniques have been proposed to estimate and minimize glitches in order to reduce dynamic power.

1.5.1 Glitch Estimation

As glitches are related to switching activity, most of the research papers found to detect glitches were related to switching activity analysis.

The approach taken in [16] [17] for switching activity analysis is to apply random input vectors and perform functional (zero-delay) simulation and timing (routed-delay) simulation. Then authors perform power analysis and compute the glitch power as the difference in dynamic power between functional and timing simulations. [17] also proposed a method to predict routed switching activity at pre-layout stage. In this technique authors obtain delay information using depth of nodes instead of the information extracted from physical layout. The depth of a node is the length of the longest path from any primary input to the node.

In [18] a *Stochastic glitch estimation* method has been proposed, which estimates the probability of glitch occurrences at the output of a gate by computing the probability of an interval time (difference of arrival times of inputs at a gate) that is bigger than the delay of that gate. The methodology proposed in [19] presents a method to obtain glitch probability numbers early in the design cycle. In this paper authors find the probability of glitch at the output of gate based on the gate's pattern and propagation probabilities. The pattern probabilty is the pattern appearing at the inputs of gate, which causes a glitch at the output of that gate and the propagation probability is obtained as the number of pairs of paths out of all possible path pairs that have a delta delay that would cause a glitch. *Symbolic simulation* [20] can be used to estimate switching activity. This technique use a variable delay model for combinational logic, which computes the boolean conditions that cause glitches in the circuit. However, this model has a long run time.

In [21], the *Transition density* technique is proposed, which corresponds to average switching rates for gates in the circuit. This paper provides a technique to calculate the total circuit switching activity by means of propagation of transition densities and signal probabilities (the average value of the logic signals over all time) from input nodes to output nodes. The low-level activity estimation techniques in [22] propose the use of the boolean difference function to find the output transition densities of a logic component based on its input transition density. The techniques in [21] and [22] can be computationally expensive for large combinational circuits.

The method for modeling glitch activity inside arithmetic circuits using word-level statistics of signals is proposed in [23] using *Glitch Profile metric.* This metric is based on transition density [21] technique. The proposed method models the propagation of glitch activity in signals through the arithmatic components in circuits.

[24] provides a survey of switching activity techniques. Techniques can be mainly classified as Statistical techniques or Probabilistic techniques. *Statistical techniques* ([18], [16]) use simulation models and simulate the circuit for limited number of randomly generated input vectors. They can be accurate and can be built around existing simulation tools and libraries, but are strongly input pattern dependent. *Probabilistic techniques* ([21], [23], [22], [19], [17], [20]) estimate the switching activity using signal probability and transition densities. They are not as accurate and cannot be built around existing simulation tools and libraries but can be faster.

[25] introduces a glitch capture technology based on FPGA, which uses dual-jump-edge detection principle. In this principle, the logic level sampled at the beginning of the sampling period is considered as reference level. During the same sampling cycle, if there is a change in logic level from the reference level, a glitch is detected. It also provides a technique for the logic analyzer based glitch capture, but it is complex and expensive.

1.5.2 Glitch Reduction

The basic technique to avoid glitches is to align the inputs by balancing the delay paths. The glitch reduction techniques can be classified according to the level of the design process they are applied to. The levels can be classified as:

- Technology Level
- Implementation Level
- Register Transfer Level

- **Technology Level**

  GlitchMap [20] is an FPGA technology mapper that is glitch aware. In this technique glitches are minimized by favoring mapping solutions that balance LUT levels between different paths. The optimum mapping depth can be computed during cut enumeration by computing depth of every node and every cut. [26] proposes the technique for LUT based FPGAs at technology mapping level. This technique aims to keep high switching activity nets out of the FPGA routing and shows the activity conscious approach for logic replication. This technique shows the effect of logic replication on circuit structure and its consequences on power. The *Gate Freezing* [32] technique eliminates glitches by suppressing transitions until the freeze gate is enabled. This technique is less suitable for FPGAs since the applications implemented on FPGAs are not known until after fabrication.

- **Implementation Level**

  Glitchless [8] [9] proposes inserting programmable delay elements into configurable logic blocks (CLBs) of FPGAs. After a glitch-unaware routing stage, these delays are used to align signal arrival times. *GlitchReroute* [27] proposes an approach to reduce dynamic power in FPGAs by reducing glitches during routing. It finds alternative routes for early arriving signals so that signal arrival times at look-up tables are aligned. The authors developed an algorithm to find routes with target delays and then built a glitch-aware router. [16] presents a *Don't-care-based* synthesis technique for reducing glitch power in FPGAs. This technique takes advantages of don't-cares in the circuit by setting their values based on the circuit's simulated glitch behavior. [28]

proposes a technique to address power dissipation due to glitches post-routing, where the delays between LUTs are known. Their algorithm reduces glitches by inserting *negative edge triggered flip-flops (FFs)* at the output of LUTs of FPGAs that produce glitches. This prevents unnecessary logic transitions from propagating to the general routing network and subsequent LUTs. [29] presents an FPGA-targeted, glitch-aware, high-level binding algorithm. The binding algorithm provides the technique for data path allocation for power and area reduction. The binding algorithm proceeds in two parts of data path allocation: *register allocation*, binds all variables into a number of registers to minimize the total number of registers used and *operation assignment*, assigns operations to the distinct functional units within a same control step. In [30], pipelining and re-timing have been used to balance the path delays and reduce wire toggle rates. The *path balancing* [18] algorithm reduces glitches using the statistical gate sizing technique that considers the probability of glitch occurrence and calculates an optimal delay amount of sizing.

- **Register Transfer Level**

  [31] proposes several techniques that attempt to reduce glitching power consumption by minimizing propagation of glitches in the *RTL circuit.* The techniques include restructuring multiplexer networks (to enhance data correlations and eliminate glitchy control signals), clocking control signals, and inserting selective delays, in order to kill the propagation of glitches from control as well as data signals.

  These techniques may introduce area, delay and also power overhead due to extra logic inserted. Table 1.2 summarizes the area and delay overhead and also the reduction in glitch and dynamic power for the different glitch reduction techniques.

1.6 Motivation

Most of the techniques for glitch detection explained in the previous section estimate the number of spurious transitions using a simulation-based approach. The post place and route

Table 1.2: Comparision of Glitch Reduction Techniques

Technique	Area Overhead	Delay Overhead	Power Reduction	
			Glitch	Dynamic
Glitch Reroute [27]	Not affected	Negligible	27%	11%
Glitch Less [8] [9]	5.3%	0.2%	45%	18%
Gate Freezing [32]	2.8%	Not affected	14%	6.3%
GlitchMap [20]	-	-	-	18.7%
Don't-care based [16]	Not affected	Not affected	49%	12.5%
Negative edge triggered FFs [28]	0.7%	3.6%	-	7.25%
High-Level binding [29]	-9%	-	22%	19%
RTL Power Optimizor [31]	Negligible	Negligible	-	22.54%

simulation provides glitch detection, but it cannot take process variations of the FPGA into account. Moreover, its accuracy depends on the simulation model only, which might not expose all the glitches from the implemented circuit. Hence, to detect all glitches we need to measure the actual hardware implementation. These measurements can be done by fast and high quality oscilloscopes, which are very expensive. In this research we introduce a novel approach for glitch detection in hardware implementations on FPGAs. Furthermore, we have used a Spartan 3E FPGA board for our research, which is very inexpensive compared to the oscilloscope. Our research does not merely detect the presence of glitches or counts the number of spurious transitions that a glitch causes. It captures the glitch waveform, which provides the information about the position and the width of glitches in the design.

1.7 Thesis Organization

Chapter 2 gives detailed insight to the Xilinx Spartan-3E FPGA architecture. It will help in understanding different approaches to detect glitches discussed in this thesis. *Chapter 3* provides the analysis of how LUT input changes affect glitches within the FPGA. In *Chapter 4*, different simulation based glitch detection approaches are analyzed. *Chapter 5* shows how the best of the discussed techniques in chapter 4 are implemented on the FPGA board. Finally, the research is concluded in *Chapter 6*.

Chapter 2: Xilinx FPGA Internal Structure

Xilinx, Altera and Actel are the major vendors among the wide verities of FPGA manufacturers. We selected *Xilinx Spartan-3E FPGA* for our glitch detection circuits because of their low-cost and low-power qualities. The design of glitch detection circuits on FPGA depends on the effective use of architectural features provided in the targeted FPGAs. Therefore, it is important to understand the detailed architecture of Spartan-3E devices. This chapter describes the underlying structure of Xilinx Spartan-3E FPGAs and their features.

2.1 Spartan-3E FPGA Architectural Overview

The Spartan-3E family architecture has five basic functional elements [33]:

1. **Configurable Logic Blocks (CLBs):** CLBs can be programmed to perform a wide variety of logic functions as well as store data. They consist of RAM based look up tables to implement logic and storage elements that can be used as flip-flops or latches.

2. **Input Output Blocks (IOBs):** They control the flow of data between I/O pins and internal logic. Each IOB supports bidirectional data flow, 3-state operation, and numerous different signal standards. *Double Data-Rate (DDR)* registers are included.

3. **Block RAM (BRAM):** It provides data storage in the form of 18-kbit dual-port/single-port blocks of memory.

4. **Multiplier Blocks:** They accept two 18 bit binary numbers as input to calculate the product.

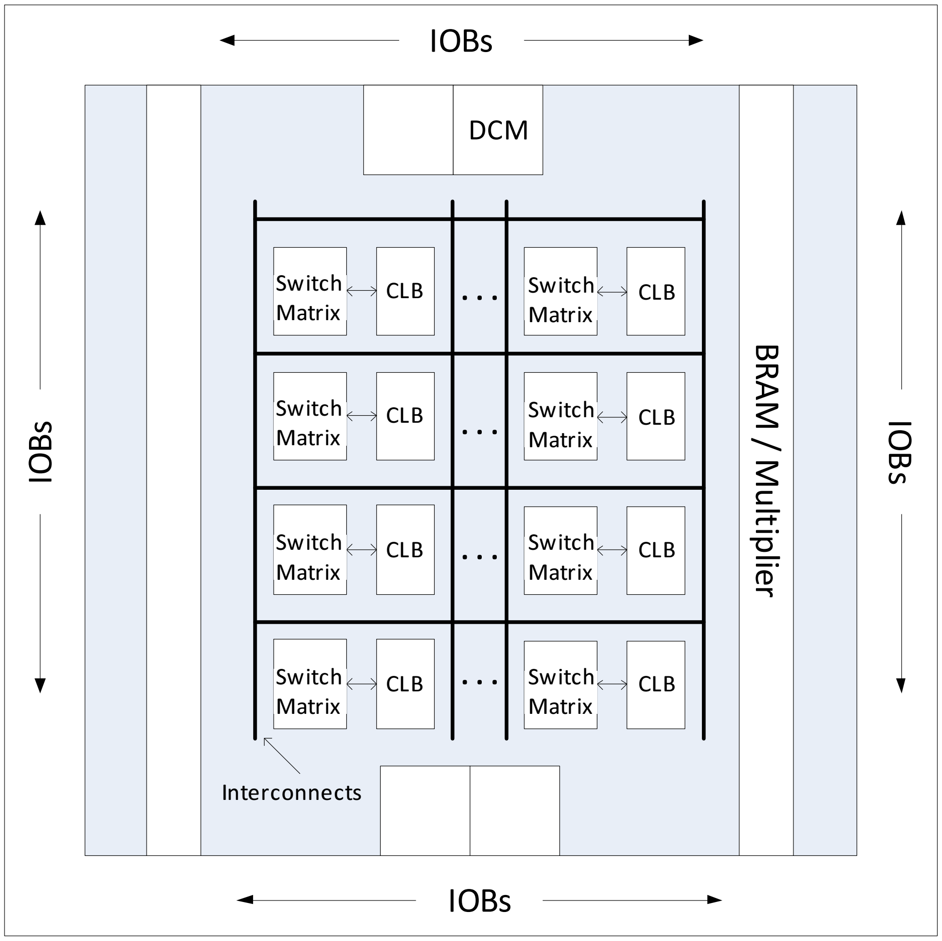


Figure 2.1: Spartan-3E FPGA Internal Architecture

5. **Digital Clock Manager (DCM):** It provides means for distributing, delaying, multiplying, dividing and phase shifting clock signals.

These elements are organized as shown in the Fig. 2.1.

A ring of IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM except for the XC3S100E, which has only one column. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device.

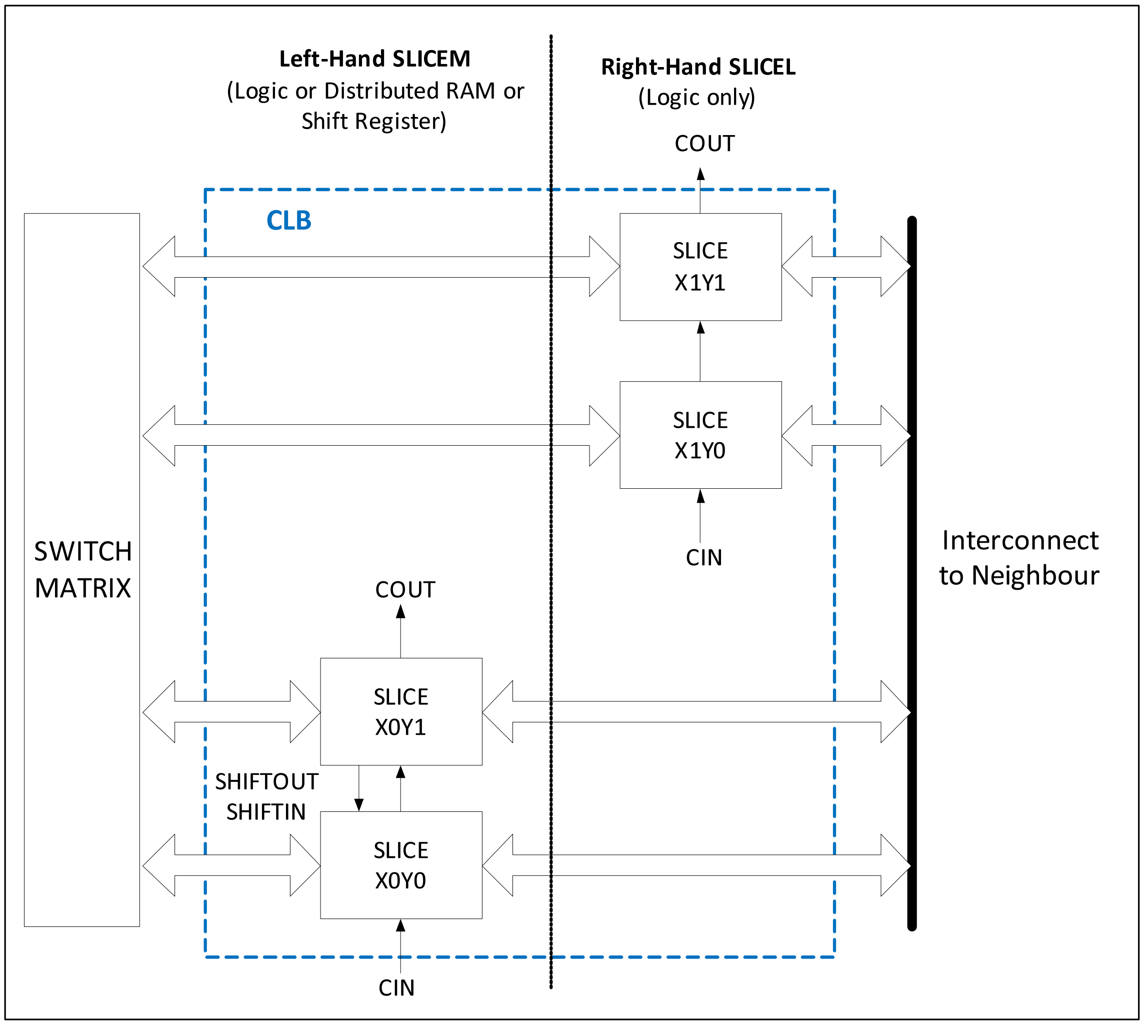


Figure 2.2: CLB Structure

The Spartan-3E family has network of routes that interconnect all five functional elements and transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the resources.

2.2 Configurable Logic Block (CLB)

The CLBs constitute the main logic resource for implementing synchronous and combinational circuits. The CLBs are organized in arrays of rows and columns [Fig. 2.1]. Each

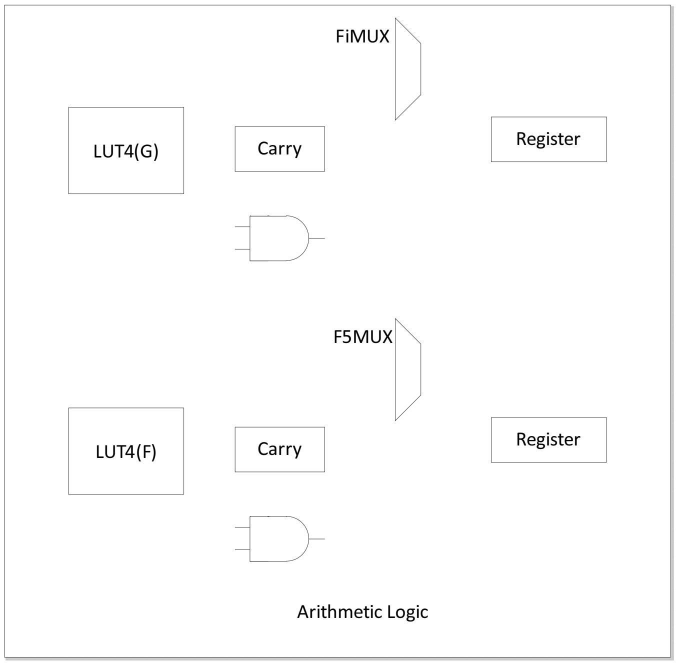


Figure 2.3: SLICEL

CLB contains four interconnected slices. The organization of slices is shown in Fig. 2.2. The pair on left side of the CLB is called *SLICEM (Memory Slice)* and it supports logic as well as memory functions. The other pair on the right side of the CLB is called *SLICEL (Logic Slice)* and it supports only logic. Each CLB is identical.

The location of a slice is determined according to its X and Y coordinates, starting in the bottom left corner of the die, which is shown in Fig. 2.2. The letter *X'* followed by a number identifies columns of slices, incrementing from the left side of the die to the right. The letter *Y'* followed by a number identifies the position of each slice in a pair as well as indicating the CLB row, incrementing from the bottom of the die. The SLICEM always has an even *X'* number and the SLICEL always has an odd *X'* number.

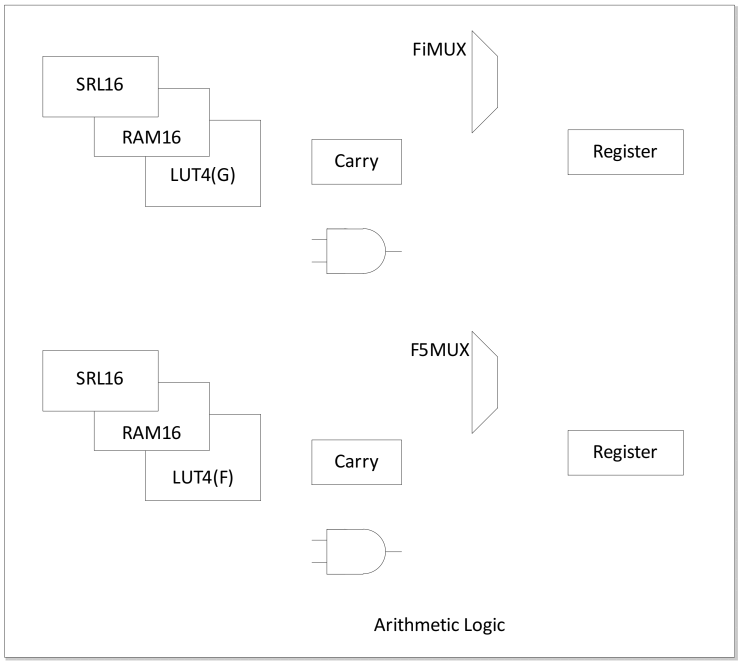


Figure 2.4: SLICEM

2.3 Slice

A slice includes two 4-inputs Look-Up Table (LUT) function generators and two storage elements, along with additional logic. The structures of SLICEL and SLICEM are shown in Fig. 2.3 and Fig. 2.4 respectively. Both slices have the following features which provide logic, arithmetic and ROM functions.

- Two 4-Input LUT Function Generators, F and G
- Two Storage Elements
- Carry and Arithmetic Logic
- Two Wide-Function Multiplexers, F5MUX and FiMUX

The two LUTs are used for implementation of logic functions. The two storage elements, which are programmable either as D-type flip-flops or as level-sensitive transparent latch,

Table 2.1: XOR Truth Table

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

provide a mean for synchronizing data to a clock signal. The carry logic is automatically used for most arithmetic functions in a design.

Figure 2.5 shows the detailed diagram of SLICEL. The LUTs located in the top and bottom portions of the slice are referred to as *"G-LUT"* and *"F-LUT"* respectively. The storage elements in the top and bottom portions of the slice are called *FFY* and *FFX*, respectively. Each slice has two multiplexers with *F5MUX* in the bottom portion of the slice and *FiMUX* in the top portion. Depending on the slice, the *FiMUX* takes on the name *F6MUX*, *F7MUX*, or *F8MUX*, according to its position in the multiplexer chain. The lower SLICEL and SLICEM both have an *F6MUX*. The upper SLICEM has an *F7MUX*, and the upper SLICEL has an *F8MUX*.

2.4 Look-Up Table (LUT)

The LUT is a RAM-based function generator and the main resource for implementing logic functions. Each of the two LUTs (F and G) in a slice has four logic inputs and a single output. Any four-variable boolean logic operation can be implemented in one LUT. Functions with more than four inputs are implemented by cascading LUTs or using wide function multiplexers.

The LUT can be described as a tree of multiplexers as shown in Fig. 2.6. The four inputs of the LUT serve as the select lines of the multiplexers. The inputs of multiplexers are stored in the look-up table depending on the logic configured. The output is obtained from the value stored in look-up table depending on the selection lines / LUT inputs.

Figure 2.7 shows a 2-input LUT implementation. The function implemented in the

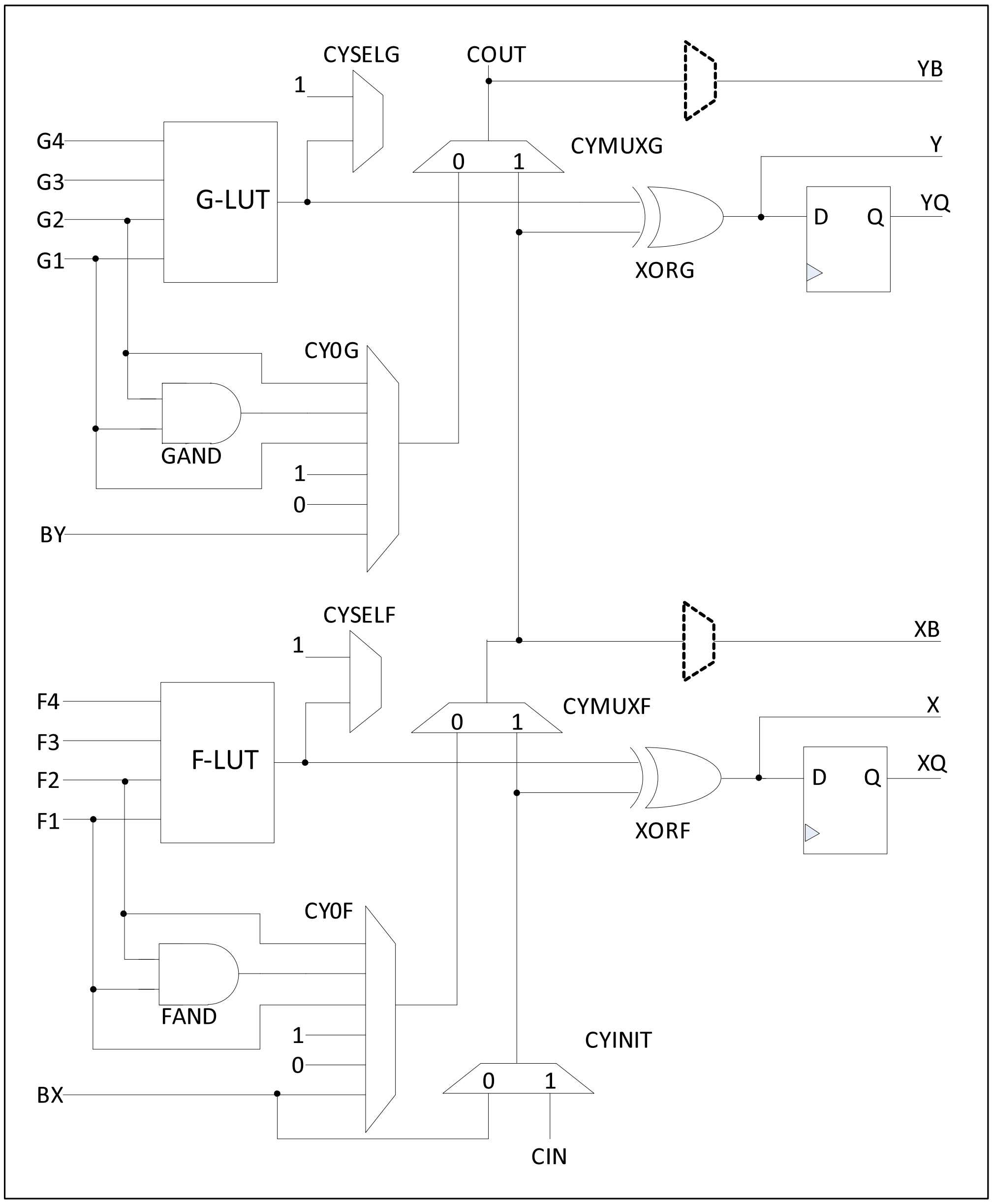


Figure 2.5: Carry and Arithmetic Logic

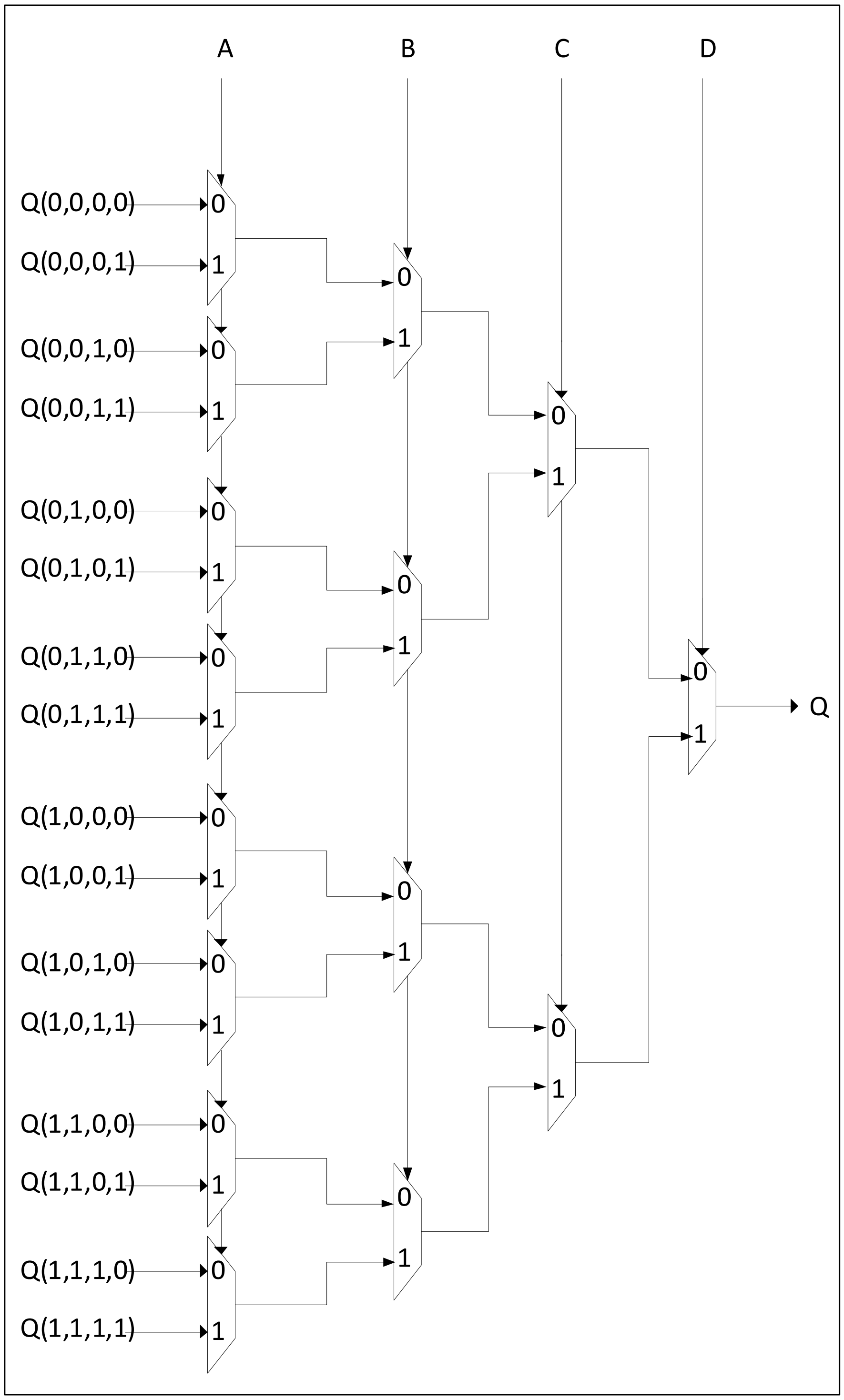


Figure 2.6: LUT Structure

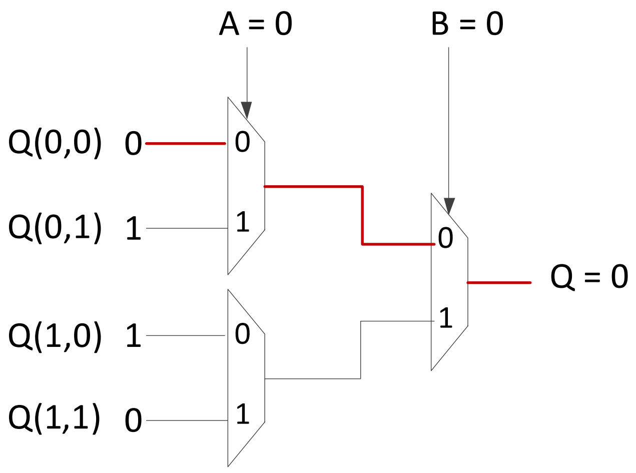


Figure 2.7: 2 Input LUT

LUT is an XOR function, given in Table 2.1. According to the logic implemented, the multiplexer's inputs are set and stored in the look-up table as shown in Fig. 2.7. When the inputs of the LUT (multiplexer selection lines) are 00, the output should be 0 according to the XOR logic. Therefore, the data value store at Q(0,0) must be 0 [Fig. 2.7].

The LUTs of SLICEM can also be configured as:

- Two 16-bit shift registers, SRL16 or
- Two 16x1 distributed RAM blocks, RAM16
- **Shift Register (SRL16)**

  The SRL16 is an alternative mode for LUTs where they are used as 16-bit shift registers. Shift-in operations are synchronous with the clock, and the output length is dynamically selectable. A separate dedicated output allows the cascading of any number of 16-bit shift registers to create whatever size shift register is needed. Each CLB resource can be configured using four of the eight LUTs as a 64-bit shift register. It is also possible to combine shift registers across more than one CLB.

  As described earlier, an LUT can be described as 16:1 multiplexer with four inputs of LUT serving as the selection lines. With the SRL16 configuration, the fixed LUT

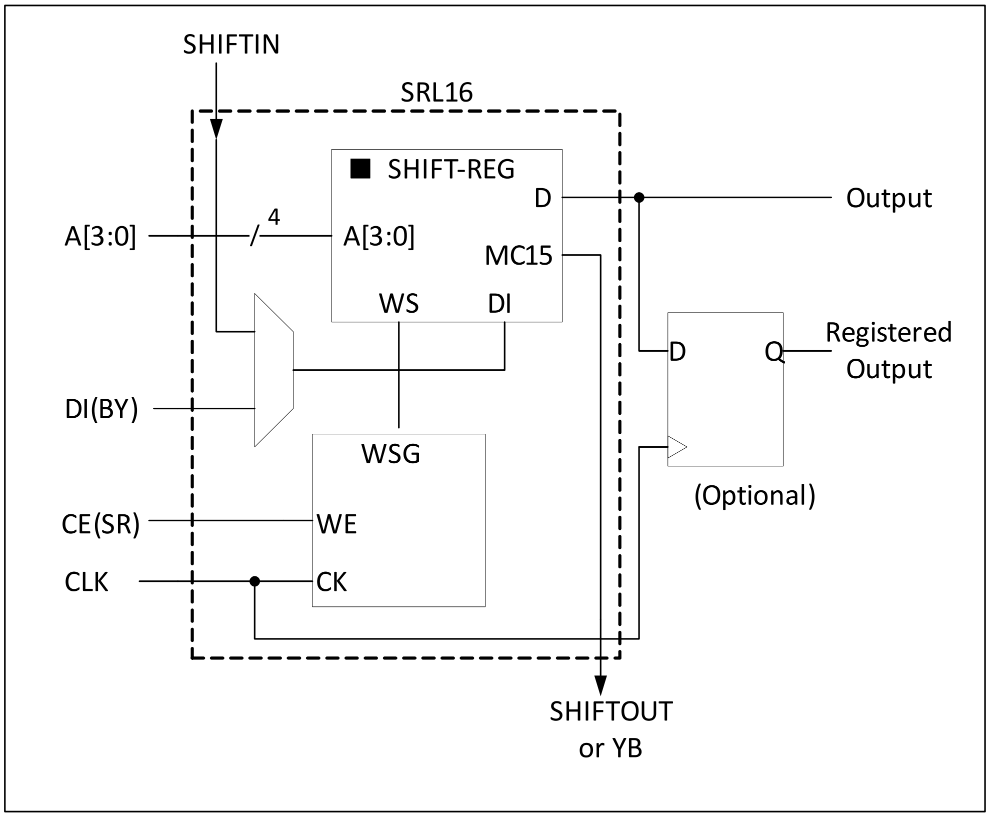


Figure 2.8: Shift Register (SRL16)

values are configured as an addressable shift register. Using the *DIFMUX*, the output *MC15* of *G-LUT* is cascaded to the input *DI* of *F-LUT*. To form larger shift register*SHIFTIN* and *SHIFTOUT* lines are used to cascade a SLICEM to the below SLICEM through *DIGMUX*.

Each shift register provides a shift output *MC15* for the last bit in each LUT, in addition to providing addressable access to any bit in the shift register through the normal *D* output [Fig. 2.8]. The address inputs *A[3:0]* are the LUT inputs *F[4:1]* or *G[4:1]*.

- **Distributed RAM (RAM16)**

  The SLICEM LUT can be configured as 16 x 1 bit synchronous RAM, called *Distributed RAM (DRAM)* or *RAM16*. Each bit can be addressed by four inputs of the LUTs. The LUTs can be cascaded for realizing deeper memories with minimal penalty on timing. Distributed RAM can be inferred or instantiated directly in design depending on the specific logic synthesis tool used.

2.5 Carry and Arithmetic Logic

The carry chain, together with various dedicated arithmetic logic gates, support fast and efficient implementations of addition operations. The carry logic is automatically used for most arithmetic functions in a design.

Figure 2.5 shows the entire carry logic and connections for one slice. The dashed lines indicate an additional multiplexer that is only found in the SLICEM half of the CLB. The carry and arithmetic logic consists of dedicated CLB resources. The carry path is very fast because it has dedicated connections within and between CLBs. These dedicated connections have almost zero delays. The carry output ($COUT$) of the bottom slice of one side of a CLB connects directly to the carry in (CIN) of the top slice. Each CLB has two sets of carry chains [Fig. 2.2], one using SLICEM and one using SLICEL. The carry chain cascades from the bottom to the top of each column of CLB slices.

The carry chain is controlled by five multiplexers: *CYINIT*, *CY0F*, and *CYMUXF* in the bottom portion and *CY0G* and *CYMUXG* in the top portion. The dedicated arithmetic logic includes the exclusive-OR gates *XORF* and *XORG* (bottom and top portions of the slice, respectively) as well as the *AND* gates *FAND* and *GAND* (bottom and top portions, respectively). The carry chain can be initialized at any point from the BX (or BY) inputs.

2.6 Block RAM (BRAM)

Block RAMs are embedded memory elements, organized in columns on FPGAs [Fig. 2.1]. They can be used for storing large amount of data in place of logic resources to save area of the FPGA. Each block RAM contains 18,432 bits of fast static RAM. Block RAMs can be configured by two types:

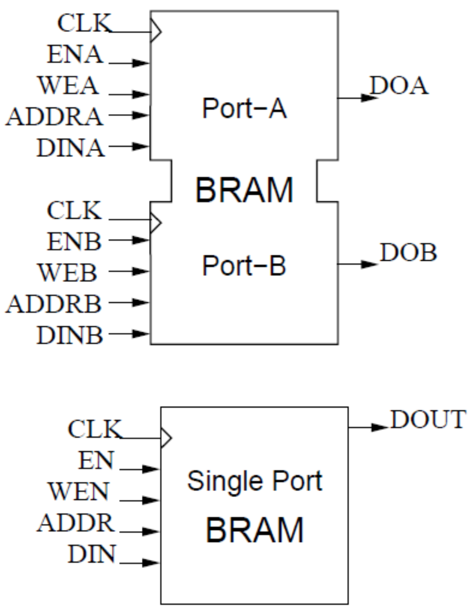


Figure 2.9: Block RAM (BRAM)

- **Single-port BRAM:** It has one set of address, control, and clock lines with maximum width of 64 bits for data lines [Fig. 2.9].

- **Dual-port BRAMS:** It has two identical data ports called *A* and *B* [Fig. 2.9], which can access the common block RAM. Each port has its own dedicated set of address, control, and clock lines with maximum width of 32 bits for data lines for synchronous read and write operations. Each port of the BRAM can be configured independently to select a number of different possible widths for the data input and data output signals.

  BRAM can be set to work in two modes:

  - **Read First Write Next mode:** It reads the old data from the memory location being addressed in that particular clock cycle before updating that memory location with the new data available at the inputs.

  - **Write First Read Next mode:** In this mode of operation the memory location being addressed gets updated with the new data at its input, and this new data

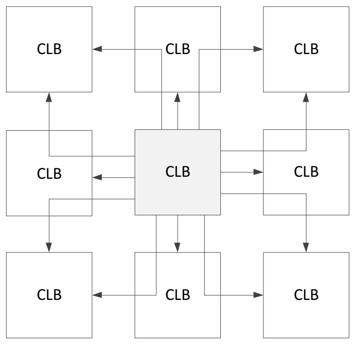


Figure 2.10: Direct Connections

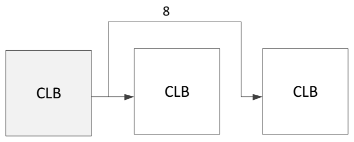


Figure 2.11: Double Lines Interconnect

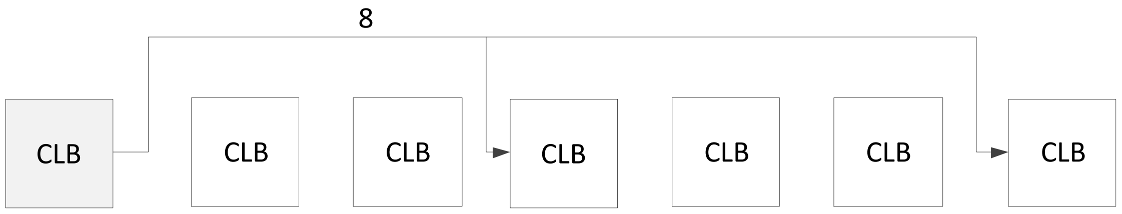


Figure 2.12: Hex Lines Interconnect

is then read out. In this mode the previous stored value thus gets lost.

Special interconnect surrounding the block RAM provides efficient signal distribution for address and data. So, multiple block RAMs can be cascaded to create wider or deeper memories.

2.7 Switch Matrix and Interconnects

Interconnect, also called routing, is the programmable network, which provides connectivity between functional elements within FPGAs such as IOBs, CLBs, DCMs and Block RAMs. Each functional element is connected to switch matrix, which connects to different kind of interconnects across the device. Fig. 2.1 shows the CLB and switch matrix connection.

The routing inside an FPGA fabric is categorized according to the distance of the

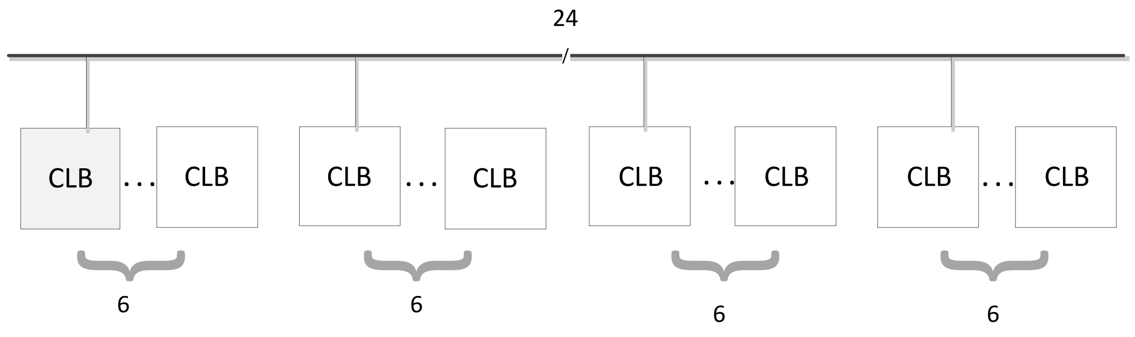


Figure 2.13: Long Lines Interconnect

output wire or line. There are four kinds of interconnects available in Xilinx FPGAs: *Long Lines* [Fig. 2.13], *Hex Lines* [Fig. 2.12], *Double Lines* [Fig. 2.11] and *Direct Lines* [Fig. 2.10]. Long lines are bidirectional lines that distribute signals across the device. Hex lines are connected to every third CLBs, double lines connect every other CLBs and direct lines connect to adjacent CLBs horizontally, vertically and diagonally. Besides these interconnects, local interconnects, called fast interconnects are available between slices of the CLB.

The Xilinx FPGA Editor tool can be used to view the interconnect of a blank device or to view the interconnect used in an implemented design [Fig. 2.14].

2.8 Editing FPGA Resources

The Xilinx ISE provides tool sets, which automatic routes the design logic on FPGA fabric during the implementation phase. To make changes in the design after implementation stage FPGA resources needs to be changed manually. There are two methods to edit FPGA resources:

1. FPGA Editor
2. Xilinx Design Language (XDL)

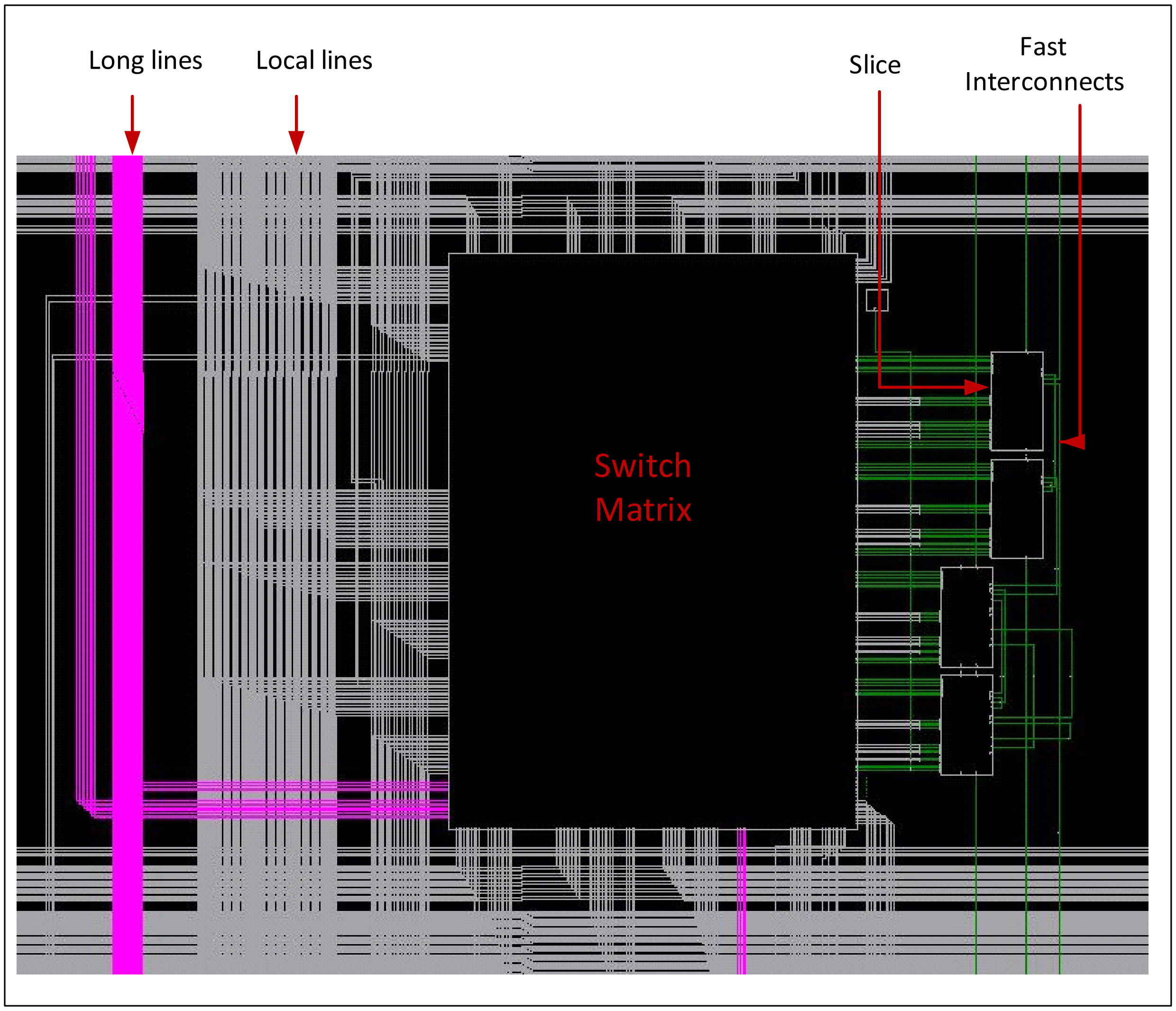
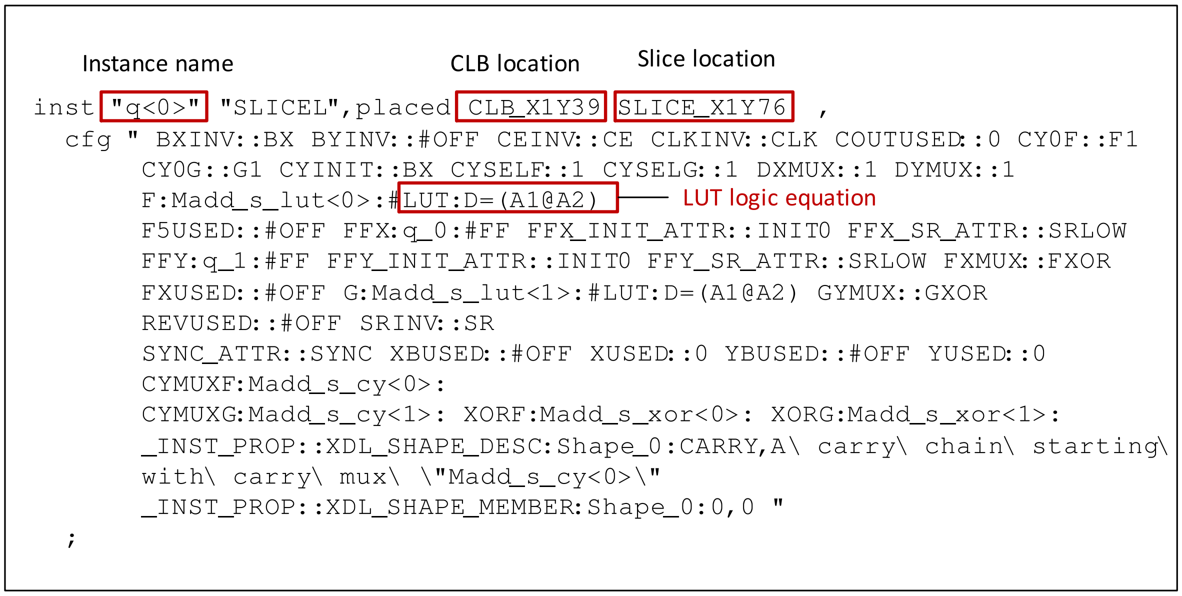


Figure 2.14: Switch Matrix and Routing



```
inst "q<0>" "SLICEL",placed CLB_X1Y39 SLICE_X1Y76  ,
  cfg " BXINV::BX BYINV::#OFF CEINV::CE CLKINV::CLK COUTUSED::0 CY0F::F1
       CY0G::G1 CYINIT::BX CYSELF::1 CYSELG::1 DXMUX::1 DYMUX::1
       F:Madd_s_lut<0>:#LUT:D=(A1@A2)
       F5USED::#OFF FFX:q_0:#FF FFX_INIT_ATTR::INIT0 FFX_SR_ATTR::SRLOW
       FFY:q_1:#FF FFY_INIT_ATTR::INIT0 FFY_SR_ATTR::SRLOW FXMUX::FXOR
       FXUSED::#OFF G:Madd_s_lut<1>:#LUT:D=(A1@A2) GYMUX::GXOR
       REVUSED::#OFF SRINV::SR
       SYNC_ATTR::SYNC XBUSED::#OFF XUSED::0 YBUSED::#OFF YUSED::0
       CYMUXF:Madd_s_cy<0>:
       CYMUXG:Madd_s_cy<1>: XORF:Madd_s_xor<0>: XORG:Madd_s_xor<1>:
       _INST_PROP::XDL_SHAPE_DESC:Shape_0:CARRY,A\ carry\ chain\ starting\
       with\ carry\ mux\ \"Madd_s_cy<0>\"
       _INST_PROP::XDL_SHAPE_MEMBER:Shape_0:0,0 "
  ;
```

Figure 2.15: Slice Internal Components Descriptions in XDL

FPGA Editor, which is a part of Xilinx ISE, is a graphical application for displaying and configuring Xilinx FPGAs. Using FPGA editor small design changes can be made, but for the extensive change in the design FPGA Editor is not efficient. *XDL* is used for detailed change in designs.

FPGA Editor and XDL require a Native Circuit Description (NCD) file. This file contains the logic of the design mapped to components (such as CLBs and IOBs). Xilinx ISE contains an XDL tool, which converts NCD to a human readable ASCII XDL file and vice versa. The commands for conversions are:

```
xdl -ncd2xdl design.ncd design.xdl  -- Converts NCD file to XDL file
xdl -xdl2ncd design.xdl design.ncd  -- Converts XDL file to NCD file
```

An example of an instance is shown in Fig. 2.15. From this, one can know the slice configuration and also can edit it according to the design specification. The slice component is represented as,

```
Component_name : : #Parameter_value
```

If a component is not used, its parameter_value is given as #OFF as shown in the Fig. 2.15

Chapter 3: Glitch Analysis within the LUT of FPGA

The *Look-Up Table (LUT)* is the main resource for logic implementations in the FPGAs. As a result of the differences in delays through the routing network and LUTs themselves, signals arriving at LUT inputs may transition at different times, leading to glitches at the output of LUTs. Glitches generated from LUTs depend on the logic configured inside the LUT and also on the inputs of the LUTs. This chapter provides insight to the experiments done to see how LUT inputs change affect glitches within the FPGA.

3.1 Basic Implementation

The Fig. 3.1 shows the basic implementation of the design used for glitch analysis. We implemented XOR logic within LUTs. All the inputs to the LUT are coming from *Flip-Fops (FFs)*. This will make sure that there are no glitches at the inputs. Slices within the CLB are connected via fast interconnects and not through the slow routing network. Therefore, in order to have minimum delay from output of FFs to the inputs of the LUT, all input FFs and the LUT are placed within the same CLB. This is done by the *AREA_GROUP* [34] design implementation constraints using *User Constraints File (UCF)*.

The delay from the output of input FFs to the inputs of the LUT are shown in Table 3.1. This delay information is obtained from the FPGA Editor after post-place and route stage. In order to analyze how LUT inputs change affect glitches within the FPGA, we experimented with 2, 3 and 4 inputs change as follows:

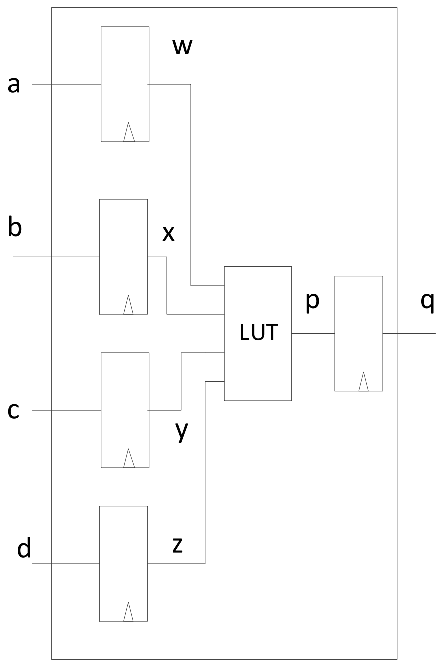


Figure 3.1: Glitch Analysis - Basic Implementation

Table 3.1: Register Output to Input of LUT Delay

Wire	**Delay (ns)**
w	0.514
x	0.383
y	0.346
z	0.554

Table 3.2: Glitch with 2 Inputs Change to Same Values

a b c d	q	Glitch Width (ps)	Wire Delay Difference of Changing Inputs (ps)
0 0 0 0	0	18	131
1 1 0 0	0		
0 0 0 0	0	16	168
1 0 1 0	0		
0 0 0 0	0	34	37
0 1 1 0	0		
0 0 0 0	0	72	208
0 0 1 1	0		
0 0 0 0	0	56	40
1 0 0 1	0		
0 0 0 0	0	38	171
0 1 0 1	0		

3.2 2 Inputs Change

In this experiment, only 2 inputs of the LUT transition from its previous values in each clock cycle, keeping the other two inputs constant. There are six possible combination of input signal changes. These can be further categorized based on the input signals transition to same value [Table 3.2] or different values [Table 3.3].

Consider the case where the inputs transition from 0000 $\rightarrow$ 1100. Ideally, the output p [Fig. 3.1] should remain constant at 0, because of the XOR logic implementaion within LUTs. However, varying arrival times on the inputs may cause an input transition sequence such as 0000 $\rightarrow$ 1000 $\rightarrow$ 1100, causing p to make a 0 $\rightarrow$ 1 $\rightarrow$ 0 transitions rather than remaining at 0. This generates a positive glitch at the output of LUT as shown in Fig. 3.2. As we can see with two input change, there can be maximum two transitions on the signal, hence a single glitch pulse can occur. The glitch is eliminated after the insertion of the FF at the output of LUT, so we cannot see a glitch on the output q in Fig. 3.1. Table 3.2 and 3.3 also shows the width of glitches observed and the difference of the input delays. As we can see from the table, the width of glitches

Table 3.3: Glitch with 2 Inputs Change to Different Values

a b c d	**q**	**Glitch Width (ps)**	**Wire Delay Difference of Changing Inputs (ps)**
0 1 0 0	1	18	131
1 0 0 0	1		
0 0 1 0	1	16	168
1 0 0 0	1		
0 0 1 0	1	34	37
0 1 0 0	1		
0 0 0 1	1	72	208
0 0 1 0	1		
0 0 0 1	1	56	40
1 0 0 0	1		
0 0 0 1	1	38	171
0 1 0 0	1		

and the difference in the input delays do not match. This leads to the conclusion that the LUT's internal delay may also affect the glitches.

3.3 3 Inputs Change

The 3 inputs change experiment is similar to the experiment done previously for 2 inputs change. Here, 3 inputs of the LUT change values from their previous values at every clock cycle. There are four possibilities to change 3 inputs values, keeping one input fixed. The truth table for this implementation is given by the Table 3.4.

Consider the case where the inputs transition from 0000 $\rightarrow$ 1110. Ideally, the output p should transit from 0 to 1 only once. However, varying arrival times on the inputs may cause an input transition sequence such as 0000 $\rightarrow$ 0100 $\rightarrow$ 1100 $\rightarrow$ 1110, causing p to make 0 $\rightarrow$ 1 $\rightarrow$ 0 $\rightarrow$ 1 transitions. This generates a glitch at the output of the LUT as shown in Fig. 3.3. For the 3 inputs change maximum of three transitions on the output signal can occur. These transitions constitute only a single glitch pulse and a useful transition as seen from Fig. 3.3. The glitch is eliminated after the insertion

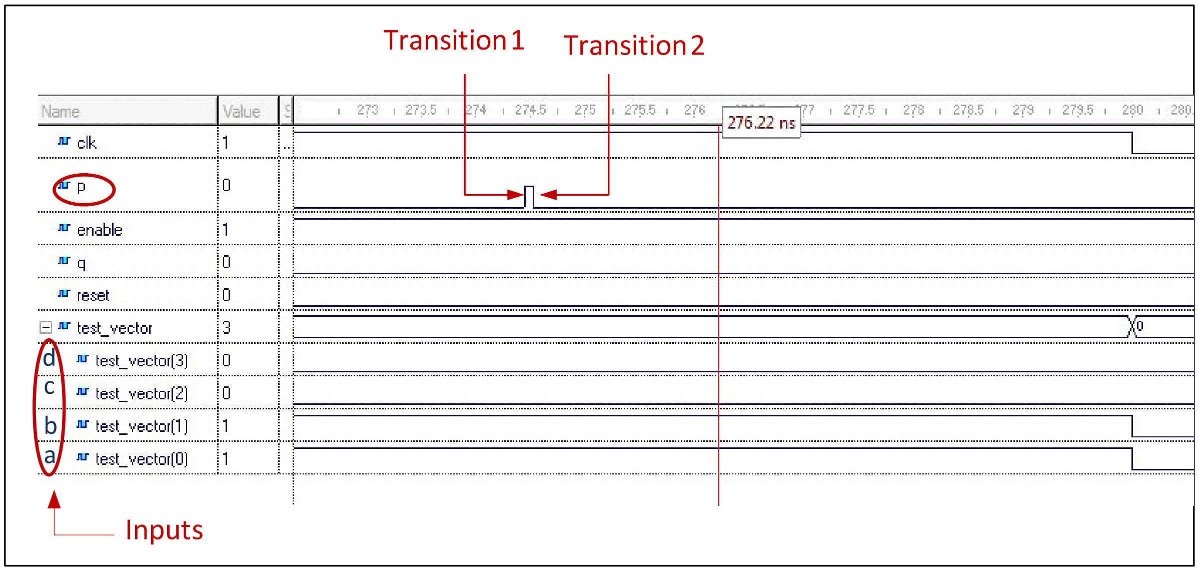


Figure 3.2: 2 Inputs Change - Waveform

Table 3.4: Glitch with 3 Inputs Changes

a b c d	q	Glitch width (ps)
0 0 0 0	0	18
1 1 1 0	1	
0 0 0 0	0	38
1 1 0 1	1	
0 0 0 0	0	56
1 0 1 1	1	
0 0 0 0	0	38
0 1 1 1	1	

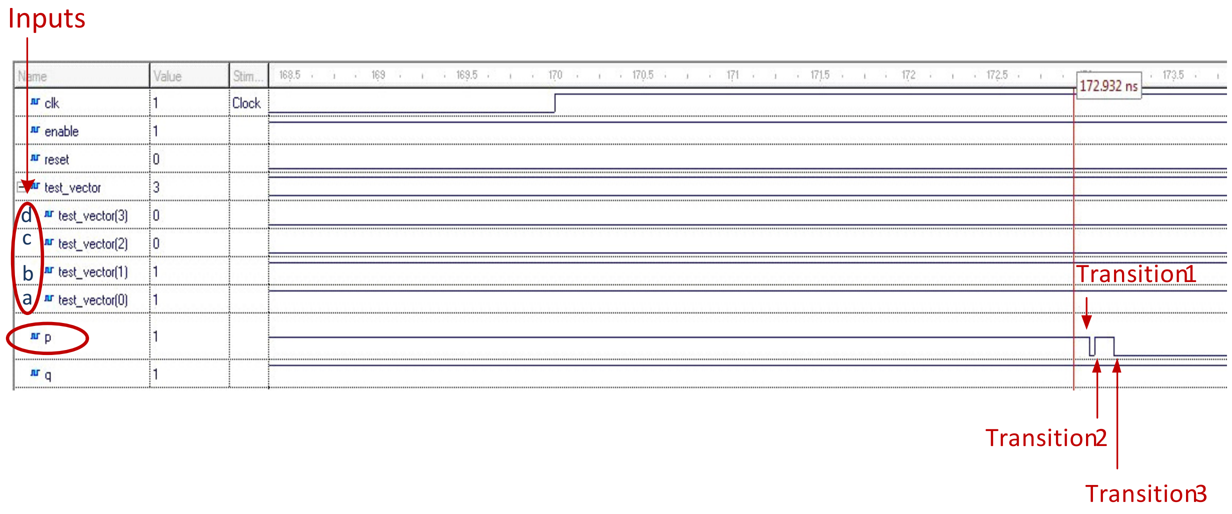


Figure 3.3: 3 Inputs Change - Waveform

of the FF at the output of the LUT. Hence, we cannot see glitch on the output q. Table 3.4 shows the width of glitches we observed.

3.4 4 Inputs Change

In this experiment all inputs of the LUT transit from their previous values to new values in each clock cycle. In addition to above experiments, we enhanced the basic implementation in this experiment. For this, we took each input FF out from the CLB and placed it in nearby CLB one by one. This allowed us to observe the effects of delays on the glitches. The truth table for this implementation is given by the Table 3.5. This table also shows the placement of the input FFs placed outside the original CLB.

Consider the case where the inputs transition from 0000 $\rightarrow$ 1111. Ideally, the output p should remain constant at 0. However, p may go through several transitions like 0 $\rightarrow$ 1 $\rightarrow$ 0 $\rightarrow$ 1 $\rightarrow$ 0 because of input transition sequence such as 0000 $\rightarrow$ 0100 $\rightarrow$ 1100 $\rightarrow$ 1110 $\rightarrow$ 1111. This generates glitches at the output of LUT. For 4 inputs change,

Table 3.5: Glitches with 4 Inputs Change

a b c d	Output	Width of Glitches (ps)				
		All FFs in same CLB	D in different CLB (X6Y31)	C in different CLB (X6Y30)	B in different CLB (X10Y31)	A in different CLB (X10Y31)
0 0 0 0	0	29, 32	48, 159	48, 159	36, 117	48, 98
1 1 1 1	0					
0 0 0 1	1	29, 32	48, 159	48, 159	36, 117	48, 98
1 1 1 0	1					
0 0 1 0	1	29, 32	48, 159	48, 159	36, 117	48, 98
1 1 0 1	1					
0 0 1 1	0	29, 32	48, 159	48, 159	36, 117	48, 98
1 1 0 0	0					
0 1 0 0	1	29, 32	48, 159	48, 159	36, 117	48, 98
1 0 1 1	1					
0 1 0 1	0	29, 32	48, 159	48, 159	36, 117	48, 98
1 0 1 0	0					
0 1 1 0	0	29, 32	48, 159	48, 159	36, 117	48, 98
1 0 0 1	0					
0 1 1 1	1	29, 32	48, 159	48, 159	36, 117	48, 98
1 0 0 0	1					

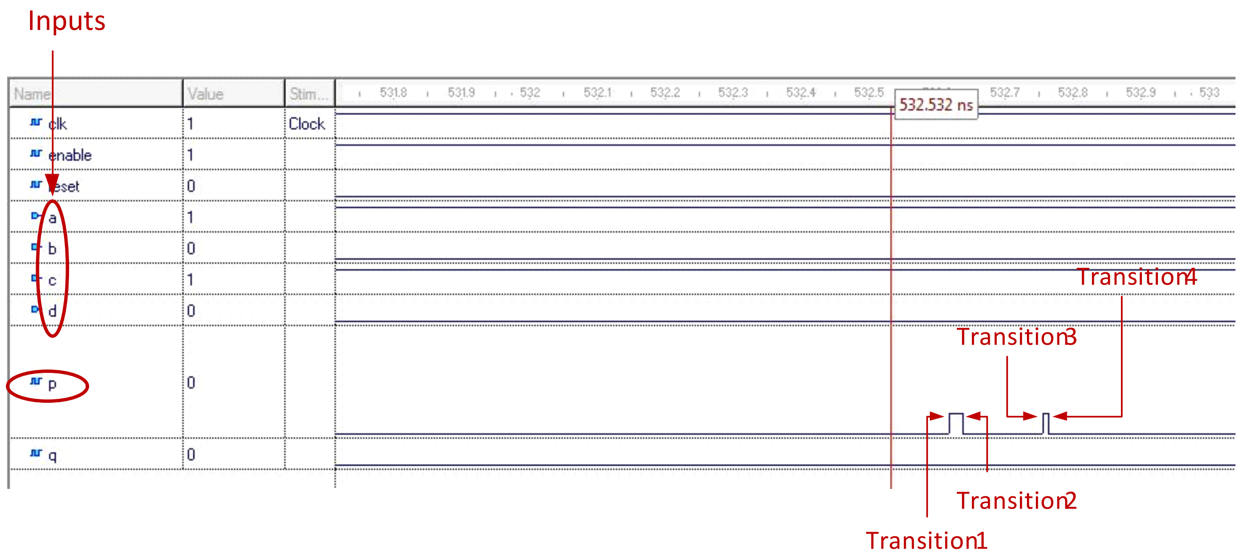


Figure 3.4: 4 Inputs Change - Waveform

Table 3.6: Comparision of Experiments

Experiments	**Minimum Width of Glitch (ps)**	**Maximum Width of Glitch (ps)**
2 Inputs Change	16	72
3 Inputs Change	18	56
4 Inputs Change	29	32

there can be maximum of four transitions on the signal that can occur. Therefore we can get maximum of two glitch pulses as shown in the Fig. 3.4. The glitch is eliminated after the insertion of the FF at the output of LUT, so we cannot see the glitches on q. Table 3.6 shows the width of the glitches observed.

3.5 Observation

Table 3.6 shows the minimum and the maximum width of glitches observed from the above experiments.

In addition to this, we also observed that when we place any single input FFs far from the LUT, the width of the glitches increased, up to 159 ps. This shows that the delays affect the width of glitches. The delay is due to the longer route connecting the FF to the LUT via the routing matrix. It is worth noting that we have looked at a single LUT only. Most combinational circuits have several LUTs and hence are even more prone to the glitches. For the research, these observations provided the specific minimum glitch width for deciding the approach to design a glitch detection circuit.

Chapter 4: Glitch Capture Circuit

4.1 Basic Idea

The main purpose of the glitch capture circuit is to reproduce the waveform of the glitch signal applied at the input. To accurately measure the glitch pulse, which has a very small width with respect to time, we need to design a glitch capture circuit that has a high enough resolution. The state of a logic signal can be captured by a flip-flop, which is typically driven by a clock. The maximum clock frequency of the Xilinx Spartan-3E FPGA is 572MHz (1.74ns), which is insufficient to capture a glitch. Hence, we adopted a method for the glitch capture circuit, where the input glitch signal is shifted by a tiny period of time for N times and sampled by the same clock. By sampling the glitch signal at N points at the same clock pulse, we obtained a sequence of *0's* and *1's* of length *N*. Integration of the sequence of *0's* and *1's* represents the sampled waveform of the glitched signal. The sampled waveform might not capture every transition of the glitch signal. However, by keeping the sampling delay short, we can increase the resolution of captured glitch waveform. This chapter provides the insight to our approach in designing the glitch detection circuit.

4.2 Glitch Capture Circuit Based on Wire Delay

4.2.1 Implementation Details

Figure 4.1 shows the implementation of a glitch capture circuit based on wire delay between sampling FFs. In this method, we used 32 sampling FFs. This 32-bit sampled data can be directly read out from the FPGA board using a USB interface.

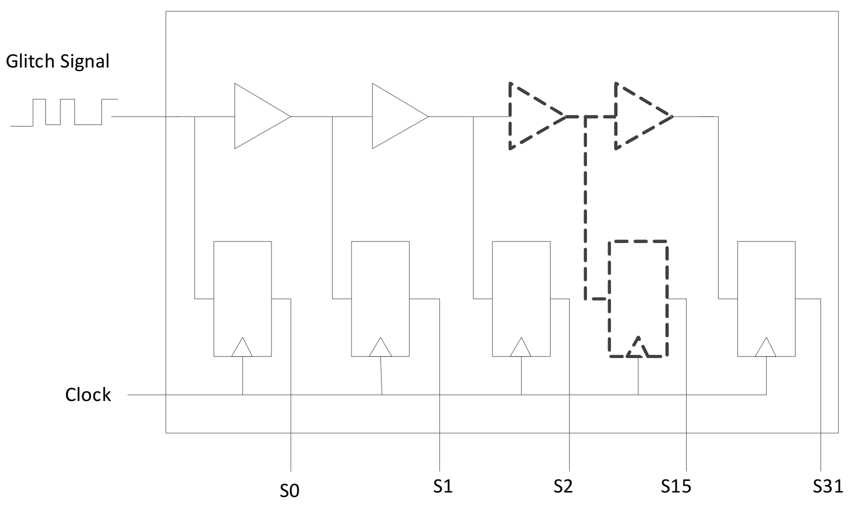


Figure 4.1: Wire Delay

For the accuracy and ease of reproduction of the sampling waveform, we used same delay between two consecutive samples. We utilized CLB's regular structure for this purpose and used only single slice from each CLB. We also used the same slice location within each CLB. To fix the slice location in the CLB, we made a hard macro using the FPGA editor. A hard macro is a placed-and-routed macro, which can be instantiated many times in the design. First we made a hard macro for a single CLB and then instantiated it 32 times in the design (the macro file name is the component name). The steps to make the hard macro are listed below:

1. Open the routed design (*.ncd*) in FPGA editor and save the design as a macro (*.nmc*).
2. Place FPGA editor in *edit mode* and remove all the IO pads (bonded to package pins and provide unidirectional or bidirectional interface between a package pin and the FPGAs internal logic.). It will un-place the pad components and un-route the signals connected to them.
3. Add External Macro pin to the site where previously routed pads were connected. These pins will be the ports of instantiated macro in the design.

4. Delete un-placed pads and un-routed nets.
5. To maintain the relative placement of the CLBs/Slices in the macro, select the CLB to use as the reference and create a *Relationally Placed Macro (RPM)*.

Then, we placed each instantiated hard macro in a chain, again to have equal delays between sampling FFs. The Xilinx's placement constraint LOC was used for this purpose. LOC defines where a design element can be placed within an FPGA. The FPGA editor can be used to find legal site names. We used an LOC constraint for each macro and specified it using UCF. The basic UCF syntax is:

```
INST instance_name LOC= location;
```

Where, location is a legal location for the part type.

For example,

```
INST "macro_loop[0]" LOC = SLICE_X66Y20;
```

To have minimum distance between sampling FFs for the accurate glitch capture, we did not place any logic between them for this design. The input glitch signal is applied directly to all the FFs, even though there are wire delays between them. The glitch signal applied to the input of this circuit was periodic signal with time period of 2ns. We performed post-place and route simulation using *Modelsim 10.1a*.

4.2.2 Observation

In the FPGA editor we observed irregular routing between two sampling FFs. Therefore, the delay between them was also different. We confirmed this from the post-place and route static timing report. We also observed that the sampled sequence of *0's* and *1's* obtained from simulation does not match with the glitch signal. Moreover, the drawback of this design is that the glitch signal is driving all the FFs, which creates a high fan-out problem.

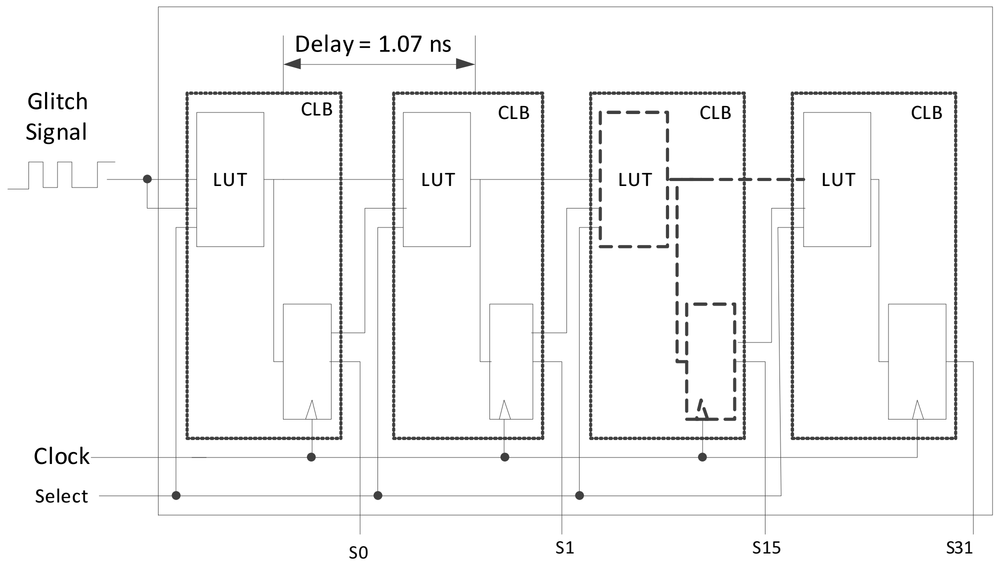


Figure 4.2: Glitch Capture Circuit Based on LUT Delay

4.3 Glitch Capture Circuit Based on LUT Delay

In this design, we tried to resolve the problems encountered in the previous design such as irregular routing between sampling FFs and high fan-out of the glitch signal.

4.3.1 Implementation Details

The implementation for this design is shown in Fig. 4.2. This design is very similar to the previous design except we introduced a LUT delay between the sampling FFs in addition to the wire delay. The glitch signal is applied only to the first LUT. The inputs of other LUTs are connected to the outputs of their previous LUTs. We used the same design criteria as the previous design such as 32 sampling FFs and use of single slice per CLB.

For this design we also implemented a hard macro, which includes the LUT delay and the sampling FFs as shown in Fig. 4.3. We call this macro a capture/read macro. A multiplexer is implemented in the LUT. When we want to use the design in

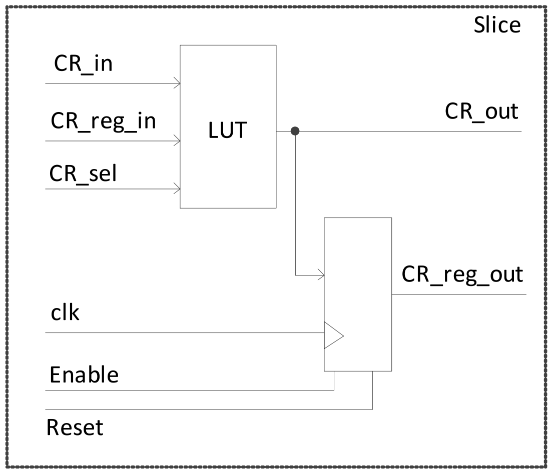


Figure 4.3: Hard Macro Implementation

capture mode (sampling mode), the *CR_in* input of LUT is selected by *CR_sel* select signal. When we want to use the design in read mode (shift mode), the *CR_reg_in* input is selected. Just like the previous design, we instantiated 32 hard macros in the design and placed them in a chain by applying LOC placement constraint in UCF file. The read mode is for future experiments only. We have used capture mode for our experiments.

From the post-place and route static timing analysis we found the delay between two consecutive CLBs (sampling delay) is **1.07ns**, which is a resolution of this design.

4.3.2 Automated Simulation and Output Analysis Process

First, to check the correctness of the design, we applied a clock signal as the glitch signal with time period of 4.2ns and 50% duty cycle at the input of this circuit. After performing post-place and route simulation, the output obtained is realized into waveform, which is shown in Fig. 4.5. The *X* in the output is due to setup time violation of the sampling FFs. For Spartan-3E FPGA, setup time (time from the setup of data at the *F* or *G* input to the active transition at the *CLK* input of the CLB) for (-4) speed grade is 0.53ns [33].

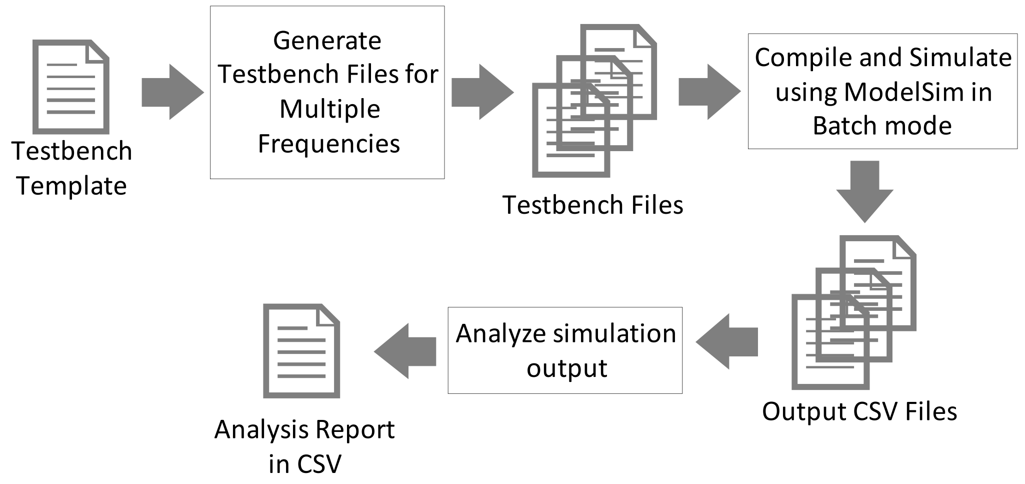
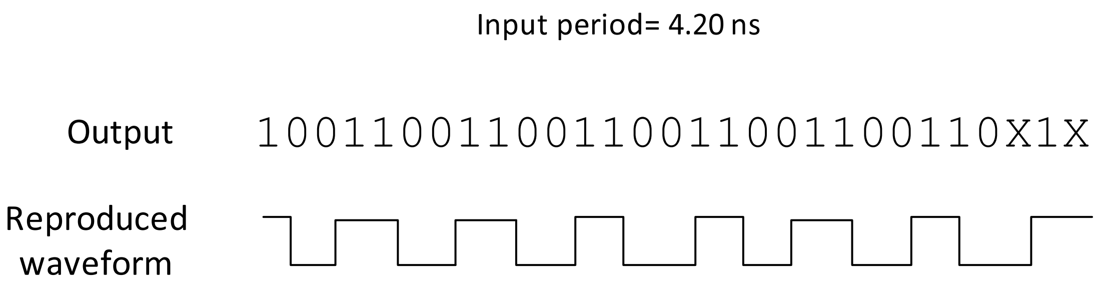

Testbench Template
Generate Testbench Files for Multiple Frequencies
Testbench Files
Compile and Simulate using ModelSim in Batch mode
Output CSV Files
Analyze simulation output
Analysis Report in CSV

Figure 4.4: Automated Simulation Process



Patterns	No. of Patterns	Digits/Pattern	No. of Digits
#00's + #11's	12	2	24
#0X's + #1X's	2	1.5	3
#1's	1	1	1
Total	15		29

Average no. of digits per pattern= 1.81

1.81 * 2 * 1.07ns(delay between two FFs) = 3.87ns

Figure 4.5: Reproduced Waveform Example Based on LUT Delay Design

From the Fig. 4.5, we can see that the shape of the reproduced sampled waveform at the output is a periodic waveform, similar to the clock signal applied as the glitch signal. To check the correctness of the waveform, we analyzed the obtained waveform to see weather the time period of it matches with the time period of the input. For this, first we assigned value *1* to digits 0 and 1 and *.5* to digit X. Then we separated all the different patterns from the output such as, single 0, single 1, 00, 11, 0X, 1X, etc (considering no pattern starts with X, only pattern breaks at X). After multiplying each pattern with its appropriate value, we get the total number of digits in the waveform. Dividing this number by total number of patterns, results the average number for digits per pattern. Multiplying 1.81 (the average number of digits per pattern) by 1.087ns (the delay between sampling FFs) and 2 (both low and high periods of one clock cycle), we get a time period 3.87ns for the obtained output as shown in Fig. 4.5. This is very close to the input time period of 4.2ns. The variation may be due to the assumption we made in assigning the value of 0.5 to X.

Once the manual analysis confirmed the correctness of design for one frequency, for the further testing and to find the limits of the design, we generated a Perl script for automatic simulation and output analysis for different frequency input signals. The process flow is shown in the Fig. 4.4. The script generates multiple testbenches for different input frequencies from a testbench template. Then each testbench is compiled and simulated one by one using ModelSim in batch mode. Output from each simulation is stored in *Comma-Separated Values* (CSV) format. In the analysis part, the script analyzes each CSV file just as explained in the manual analysis process and the final report for all the frequencies is stored in a new CSV file. A sample of the analysis report is shown in the Table 4.1. Figure 4.6 shows the graph for different input signal time periods versus the corresponding output signal time periods obtained from the analysis. From the accuracy column of Table 4.1 and Fig. 4.6, we can see that the output signal time periods follow closely to the input signal time periods. Any discrepancy in the results may be due to the assumption we took such as no

Table 4.1: Sample of the Analysis Report

Time period of an Input signal (ns)	**Sampling Output**	**Avg. # of 0's or 1's in a Pattern (Output width / No. of unique patterns)**	**Time period of an Output signals (ns)**	**Accuracy (in percentage)**
01.8	`101X0101X0101X0101X010X10101X0`	1.13 (27 / 24)	02.42	74.40
02.1	`10101010101010101010XXXXXX10101`	1.18 (29.5 / 25)	02.53	79.60
03.9	`11001X01100110X10011001X01100`	1.72 (27.5 / 16)	03.68	94.36
04.2	`100110011001100110011001100110X1X0`	1.81 (29 / 16)	03.87	92.15
05.1	`0011100110001100X110011X0011`	2.25 (27 / 12)	04.81	94.32
06.3	`00011100011100011100011100X11`	2.85 (28.5 / 10)	06.10	96.83
07.2	`111100011100001110001111000`	3.38 (27 / 8)	07.23	99.58
08.1	`00001111000011100001111000X`	3.79 (26.5 / 7)	08.11	99.87
09.6	`1111100001111X000011110000 0`	4.42 (26.5 / 6)	09.46	86.00
10.5	`111110000011111000001111X`	4.9 (24.5 / 9)	10.49	99.90
11.4	`1111100000111110000000000011111`	5.2 (26 / 5)	11.13	97.64
12.0	`0000001111100000011111000000`	5.6 (28 / 5)	11.98	99.83
13.5	`000000111111000000111111000000`	6.00 (30 / 5)	12.84	95.12
15.0	`000000011111110000000`	7.00 (21 / 3)	14.98	99.86
16.2	`0000000111111110000000`	7.33 (22 / 3)	15.69	96.80
17.4	`00000000111111110000000 0X`	8.17 (24.5 / 3)	17.48	99.54
18.9	`11111111100000000X`	8.75 (17.5 / 2)	18.73	99.10
19.2	`111111111000000000`	9.00 (18 / 2)	19.26	99.68

pattern starts with X and the value of X.

4.3.3 Observation

From all the experiments and analysis, we can say that this design is able to reproduce the input glitch signal at the output. Although, we also found that we are not able to reproduce the waveform for the input glitch signal time period less than 1.8ns (frequency higher than 555Mz). We should be able to get the desired output for the input signal time period less than 1.8ns because the hold time (Time from the active transition at the CLK input to the point where data is last held at the F or G input) is 0 ns and setup time is 0.53ns for the Spartan-3E FPGA with -4 speed grade [33],

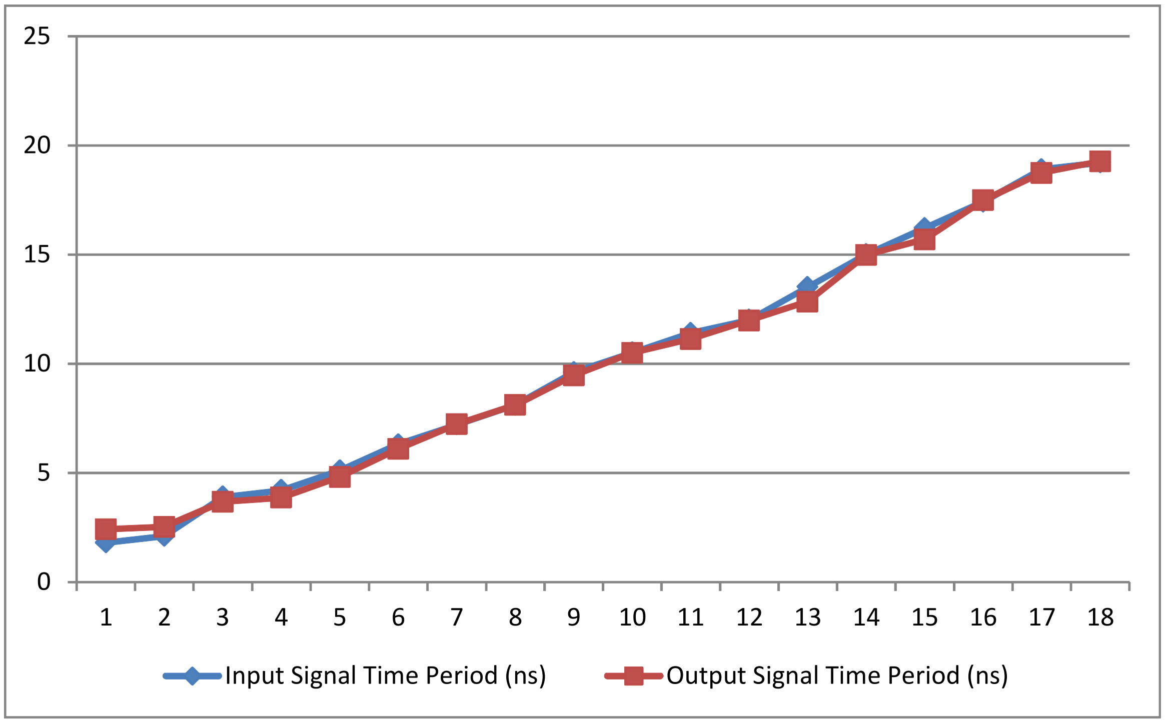


Figure 4.6: Simulation Results Chart

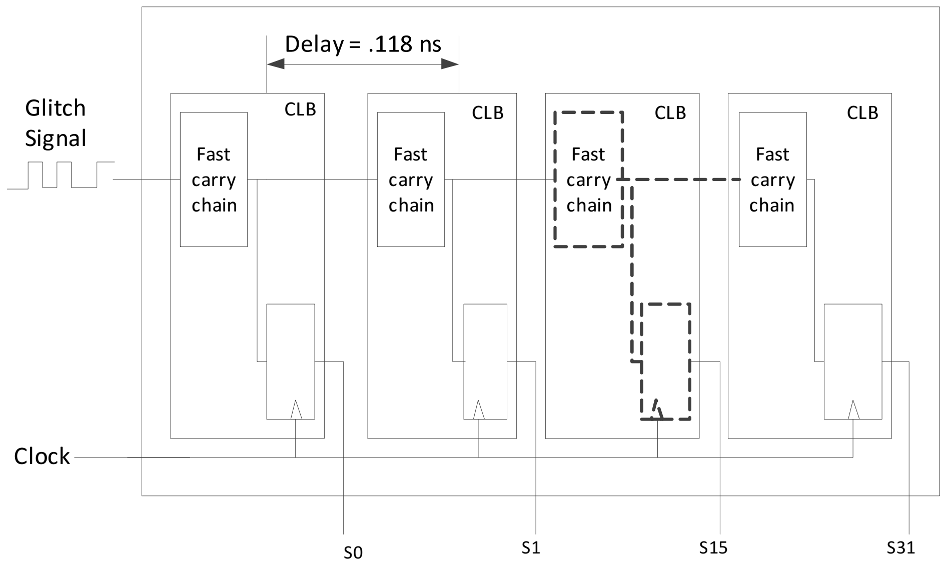


Figure 4.7: Glitch Capture Circuit Based on Carry Chain Delay

which is less than the circuit resolution of 1.07ns, which is also less than 1.8ns. We assume that this might be due to a limitation of the simulation tools.

4.4 Glitch Capture Circuit Based on Fast Carry Chain Delay

In previous design, the sampling delay was 1.07ns and we could not find expected results for the input signal period less than 1.8ns. We thought if we reduce the sampling delay, we might reproduce the waveform for the input signal period less than 1.8ns. The fast carry chains of the FPGA are very fast as they use dedicated routing resources. We used them as the sampling delay element for this experiment.

4.4.1 Implementation Details

Fig. 4.7 shows the design of glitch capture circuit based on fast carry chain delay. The design criteria such as 32 sampling FFs and use of single slice per CLB are the same as the previous LUT based delay design. We created a fast carry chain hard macro for this design and instantiated 32 times.

The fast carry chain hard macro is implemented using FPGA editor and XDL. The macro design is shown in Fig. 4.8. For the first macro only, *CYINIT* is selected to *BX*, where the glitch input signal is applied. For the rest of the macros *CYINIT* is selected to *CIN* to pass the signal in the carry chain. Then we made *CYSELF* and *CYSELG* to select 1 and also the selected inputs are tied to 1. As a result, *CYMUXF* and *CYMUXG* can propagate *CIN* to *COUT*. We also implemented XOR logic in the LUT and applied inputs such that LUT output is always 0. So, *CIN* value is always passed through *XORF* and is sampled at *XQ*.

4.4.2 Observation

We observed that the sampling delay between two FFs for the fast carry chain design is **0.118ns**, which is the resolution of this design. The sampling delay of this design is very less compared to the other designs and the structure of this design is very regular. We confirmed the regular structure with FPGA editor after place and route. Although, when we analyzed the post-place and route simulation data, the reproduced output waveforms and their frequencies did not match with the input signals frequencies. We assume that this might be due to implementation error.

4.5 Glitch Capture Circuit Based on Variable Delay

From LUT delay design, we observed the correct reproduced waveform. We will call this design as LUT based glitch capture design now onwards. However, this design does not work for the input glitch signal with time period less than 1.8ns. Hence, to improve the resolution of the design we introduced a variable delay in front of the LUT based glitch capture design as shown in Fig. 4.9. We repeatedly applied the same glitch signal input to the design, but started the detection each time with a different

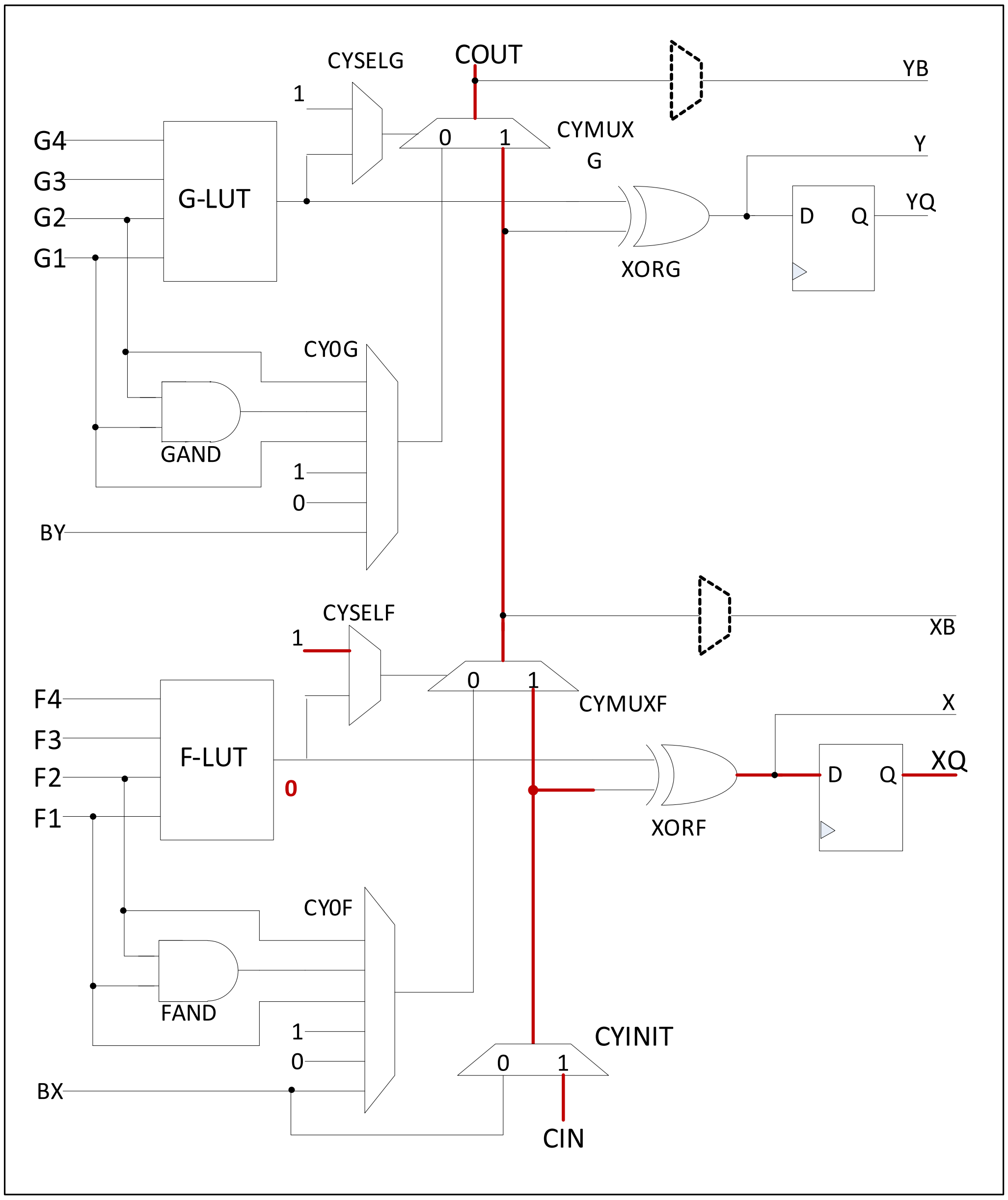


Figure 4.8: Glitch Capture Circuit Based on Fast Carry Chain Delay

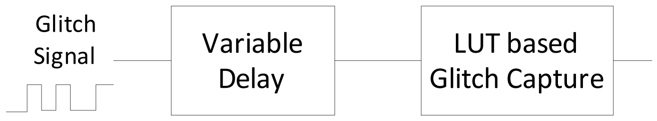


Figure 4.9: Glitch Capture Circuit Based on Variable Delay

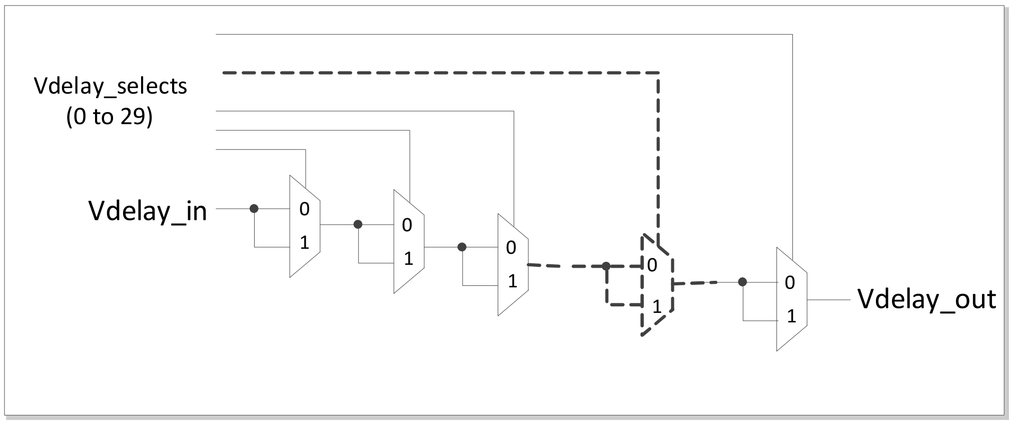


Figure 4.10: Variable Delay Implementation

delay selected from the variable delay design. By post-processing the sampled data, we can reproduce the output waveform of the circuit as shown in Fig. 4.11.

4.5.1 Implementation Details

When we implemented simple 2-1 multiplexer, we found different delays from input to output for both the inputs. When the 0^{th} input is selected the delay for input to output was **5.931ns** and for other selection the delay was **6.02ns**. We used this information to implement our variable delay design, which is shown in Fig. 4.10. We cascaded 30 multiplexers, placing only one multiplexer per CLB to have constant delays between them. The delay between two adjacent selections (000...000 $\rightarrow$ 000...001) is 0.046ns. The resolution is improved from LUT based glitch capture design by 95.7%

Note: The setup and hold time of FFs limit the detection of glitches which are larger than 0.53ns. However, the delay circuit allows to determine the start and length of such glitches with a resolution of 0.046ns.

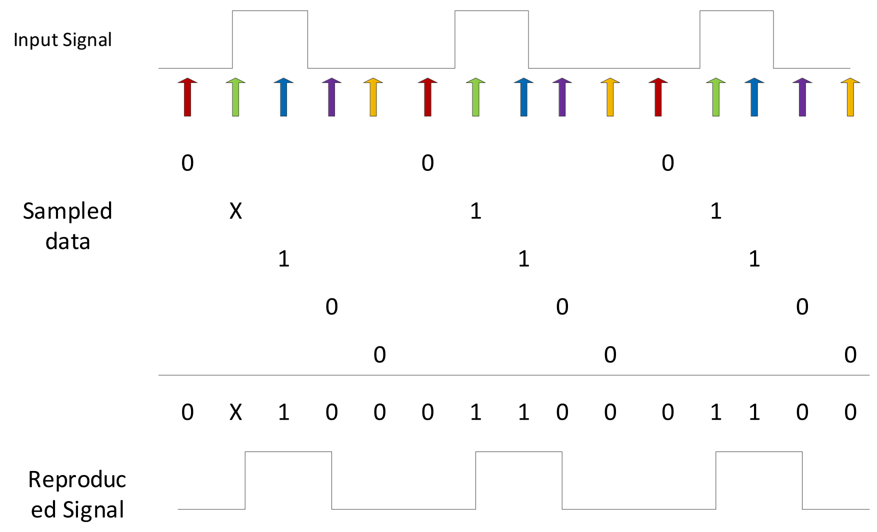
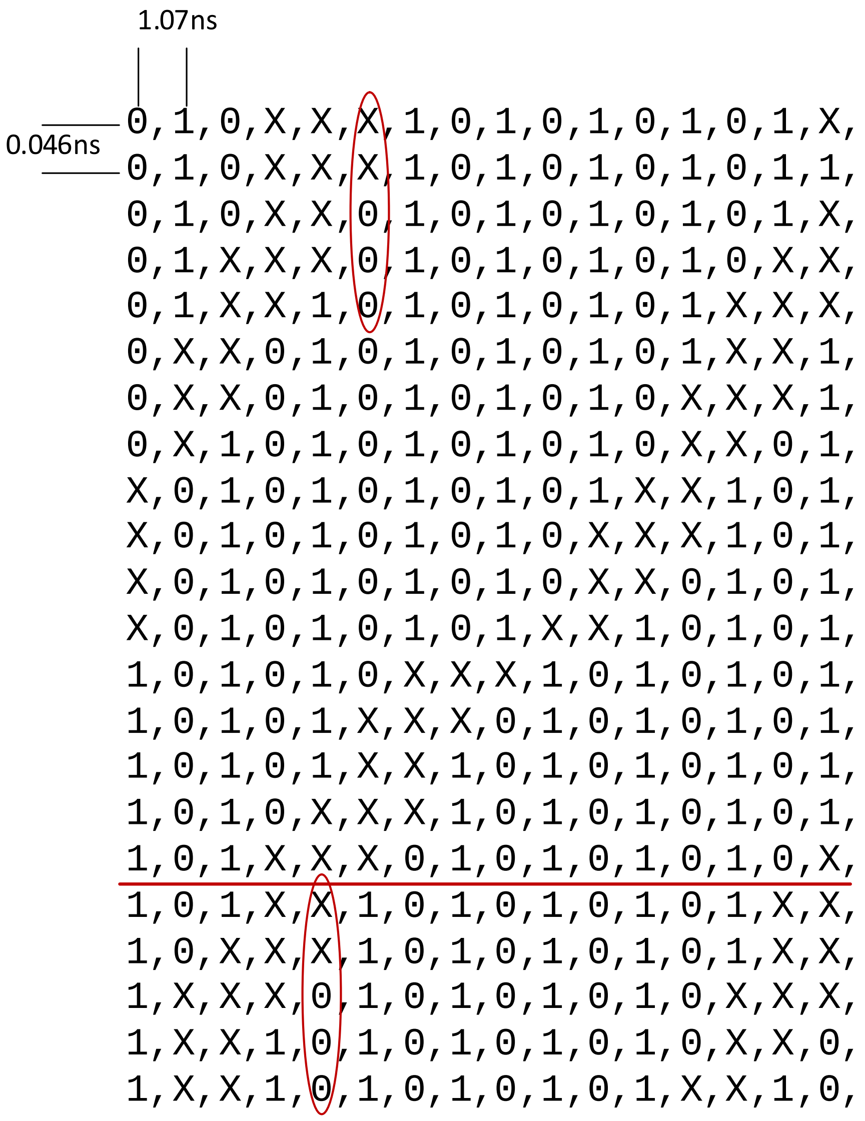


Figure 4.11: Example of Reproduced Waveform Based on Variable Delay

4.5.2 Simulation

The maximum delay of the variable delay design is **36.17ns** and the delay of the LUT based glitch capture design is **38.53 ns**. Considering a delay Δ of the input circuit, the sampling should be done before 75ns (Δ delays + 36.17ns +38.53ns). During the simulation, we applied different input signals with different frequencies to the circuit and obtained simulated output. A sample of the output obtained from the simulation by applying 2 ns periodic signal is shown in Fig. 4.12. Each row is the sampled data of the LUT based glitch capture circuit and each column is the delay introduced to the input signal from the variable delay circuit. We integrated the output data to reproduce the waveform. As we can see from the Fig.4.11, if we increase the input delay more than the sampling delay, overlapping will occur. In order to avoid the overlap and to reproduce the accurate waveform it is important to decide on the maximum delay from the variable delay design. For this, we try two methods.

– We analyzed the output and as we can see from the Fig. 4.12, the XX000 from column 5 overlaps with top of the next column. This is same for the other



```
0,1,0,X,X,X,1,0,1,0,1,0,1,0,1,X,
0,1,0,X,X,X,1,0,1,0,1,0,1,0,1,1,
0,1,0,X,X,0,1,0,1,0,1,0,1,0,1,X,
0,1,X,X,X,0,1,0,1,0,1,0,1,0,X,X,
0,1,X,X,1,0,1,0,1,0,1,0,1,X,X,X,
0,X,X,0,1,0,1,0,1,0,1,0,1,X,X,1,
0,X,X,0,1,0,1,0,1,0,1,0,X,X,X,1,
0,X,1,0,1,0,1,0,1,0,1,0,X,X,0,1,
X,0,1,0,1,0,1,0,1,0,1,X,X,1,0,1,
X,0,1,0,1,0,1,0,1,0,X,X,X,1,0,1,
X,0,1,0,1,0,1,0,1,0,X,X,0,1,0,1,
X,0,1,0,1,0,1,0,1,X,X,1,0,1,0,1,
1,0,1,0,1,0,X,X,X,1,0,1,0,1,0,1,
1,0,1,0,1,X,X,X,0,1,0,1,0,1,0,1,
1,0,1,0,1,X,X,1,0,1,0,1,0,1,0,1,
1,0,1,0,X,X,X,1,0,1,0,1,0,1,0,1,
1,0,1,X,X,X,0,1,0,1,0,1,0,1,0,X,
1,0,1,X,X,1,0,1,0,1,0,1,0,1,X,X,
1,0,X,X,X,1,0,1,0,1,0,1,0,1,X,X,
1,X,X,X,0,1,0,1,0,1,0,1,0,X,X,X,
1,X,X,1,0,1,0,1,0,1,0,1,0,X,X,0,
1,X,X,1,0,1,0,1,0,1,0,1,X,X,1,0,
```

Figure 4.12: Sample Output from Variable Delay Based Design

columns too. Therefore, we can chop of those repetitive rows from the output, which leaves 23 rows.

– We know the sampling delay of LUT based glitch capture design is 1.07ns and the resolution of Variable delay design is 0.046ns. In order to avoid overlap, we can have maximum 23 (1.07/0.046 = 23 approx.) rows.

4.6 Observation

With the glitch capture circuit based on variable delay we obtained the glitch detection resolution of 0.046ns, which indicates 95.7% increase in glitch detection resolution from the LUT based glitch capture circuit. For the experiments, we applied input glitch signal with different frequencies. Although, we are not able to reproduce the waveform for the input glitch signal frequency higher than 555Mz (1.8ns), which is consistant from the LUT based glitch capture circuit. We assume that this might be a limitation of the simulation tools and FPGA simulation models.

Chapter 5: Glitch Measurement

In previous chapter we demonstrated the design of the glitch capture circuit with a detection resolution of 0.046ns. However, we are not able to capture glitches smaller than 1.8ns width, which might be due to the limitation of the simulation tools. Therefore, we decided to implement the glitch detection circuit on an FPGA board. We used the Nexys II FPGA board, which has a XC3S500E device with -4 speed grade from the Xilinx Spartan-3E family. It runs at a clock frequency of 50MHz. For the simulation, we know the delay difference (0.046ns) of the variable delay circuit and the sampling delay (1.08ns) of the glitch capture circuit. In simulation we can obtain this information from the FPGA editor and from the post-place and route static timing analysis. This information is required to reproduce the input waveform. However, due to the process variation of the FPGAs, we can not have the accurate delay information. Therefore, we implemented a ring oscillator (RO) for on-board circuit calibration.

5.1 Ring Oscillator

5.1.1 Implementation Details

Fig. 5.1 shows the implementation of the ring oscillator. As we can see, it consists of one XOR gate and two Inverters connected in a loop. When *temp(0)* is 0 and *osc_en* is 1, *temp(2)* will be 1, *temp (1)* will be 0, *temp(0)* will be 1, which makes *temp(2)* 0. Hence, we get continuous fast pulses at *osc_clk_out*. Each gate is implemented in a separate slice and even in a separate CLBs to have pulses with 50% duty cycle.

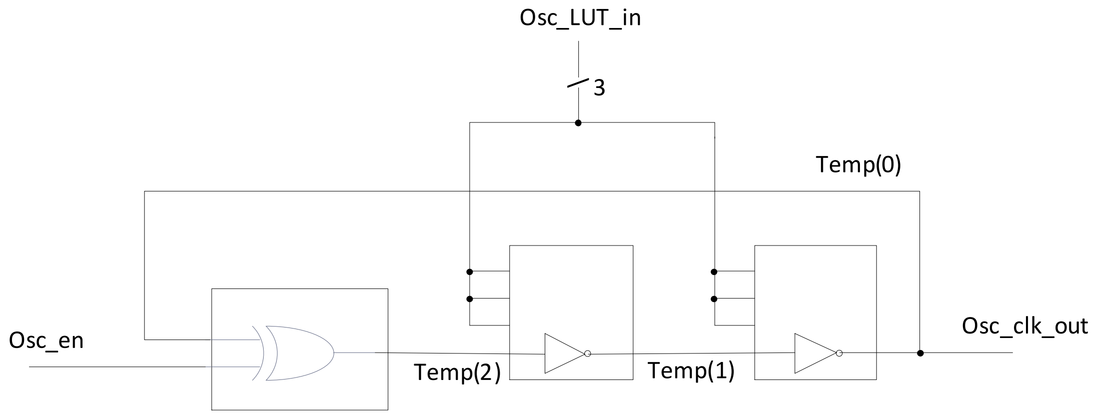


Figure 5.1: Ring Occilator Implementation

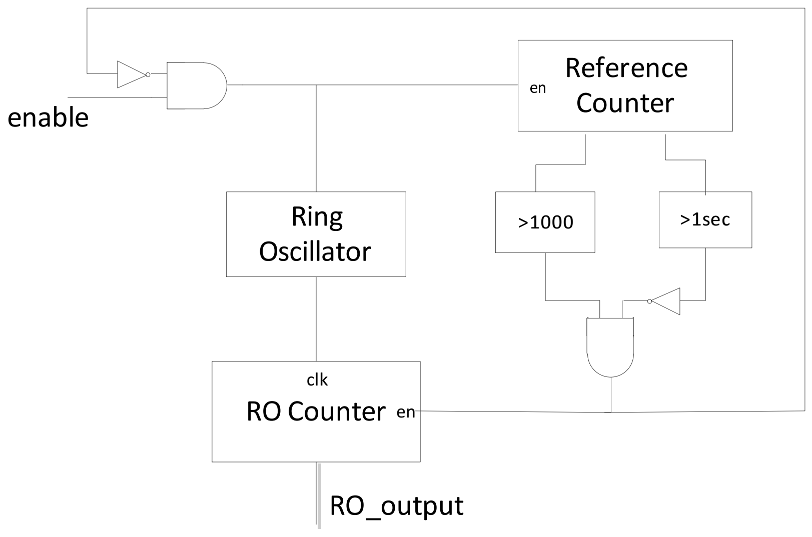


Figure 5.2: Ring Oscilator Frequency Measurement

The ring oscillator frequency is measured using the circuit shown in the Fig. 5.2. A reference counter enables or disables the RO counter. The clock of RO counter is the output of the ring oscillator. First, the ring oscillator and the reference counter are enabled and RO counter is disabled, which gives time for ring oscillator to warm up. After 1000 clock cycles, RO counter will be enabled and it will count the ring oscillator clock cycles. After one second, the RO counter and the reference counter are disabled. The RO counter output is the frequency of the ring oscillator in Hz.

Because the ring oscillator is a logical loop, the simulator can not simulate it. Hence, to measure the ring oscillator frequency, we loaded the design on the FPGA board and obtained on-board ring oscillator frequency of **101.85MHz**. To check correctness of the implementation, we used a testbench to produce the ring oscillator frequency.

5.2 Glitch Measurement

5.2.1 Implementation Details

Figure 5.3 shows the block diagram for the implementation of the on-board glitch detection circuit. All the modules are controlled by the controller module (controller module not shown in the fig.) to ensure correct operation. The highlighted green blocks are placed at fixed locations on FPGA. The signals highlighted in red are the external input-output signals of the design. The on-board glitch detection experiment is done in three steps: *determine ring oscillator frequency*, *determine glitch detection circuit calibration* and *glitch detection.*

This design first determines the ring oscillator frequency using RO counter, reference counter and ring oscillator as explained earlier. Then, for the calibration, the ring oscillator output is applied to the variable delay circuit. At every clock cycle, the variable delay value changes and the ring oscillator output is sampled by the detection circuit (LUT based glitch capture circuit). The sampled data is stored in BRAM.

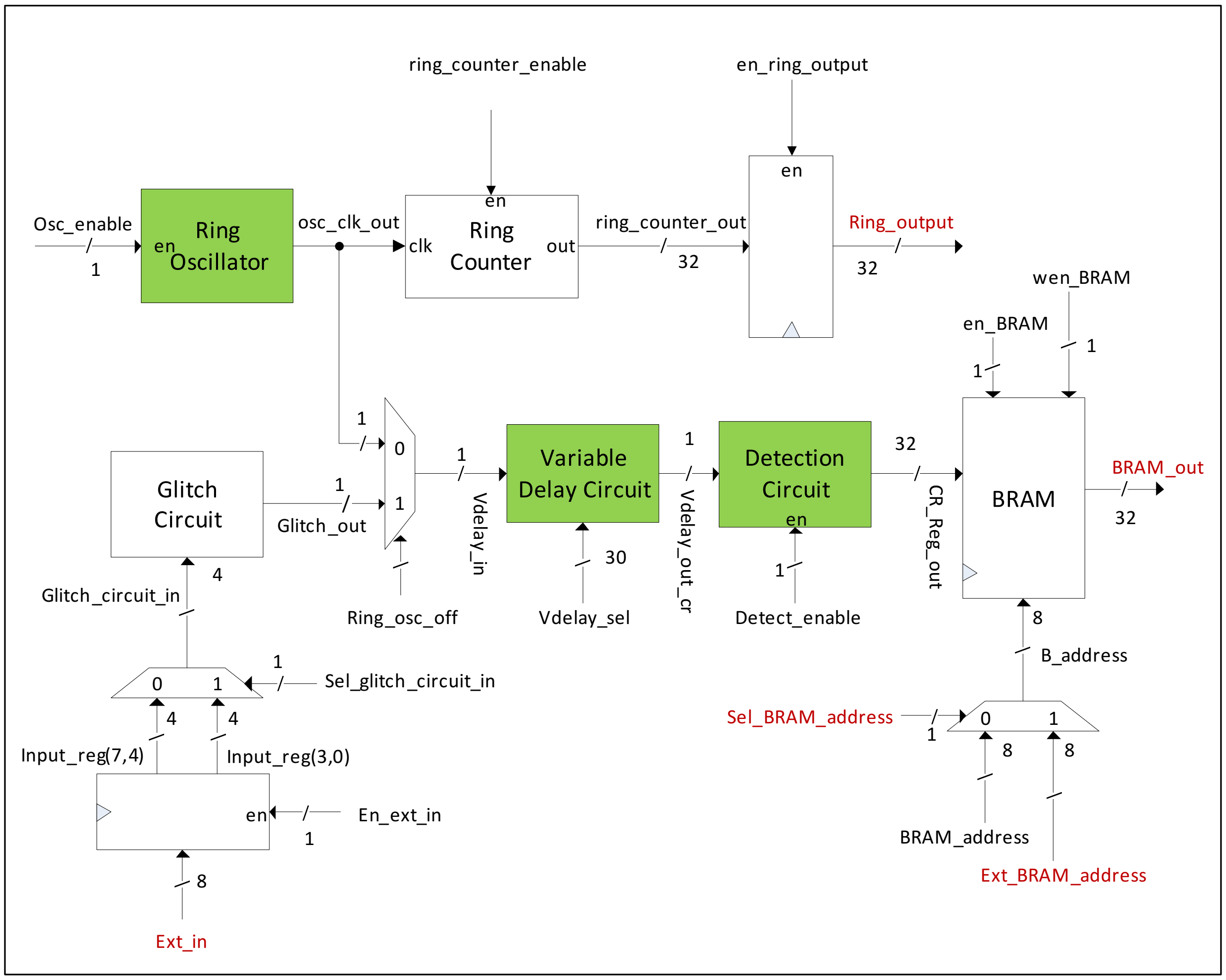


Figure 5.3: Complete Glitch Detection Datapath

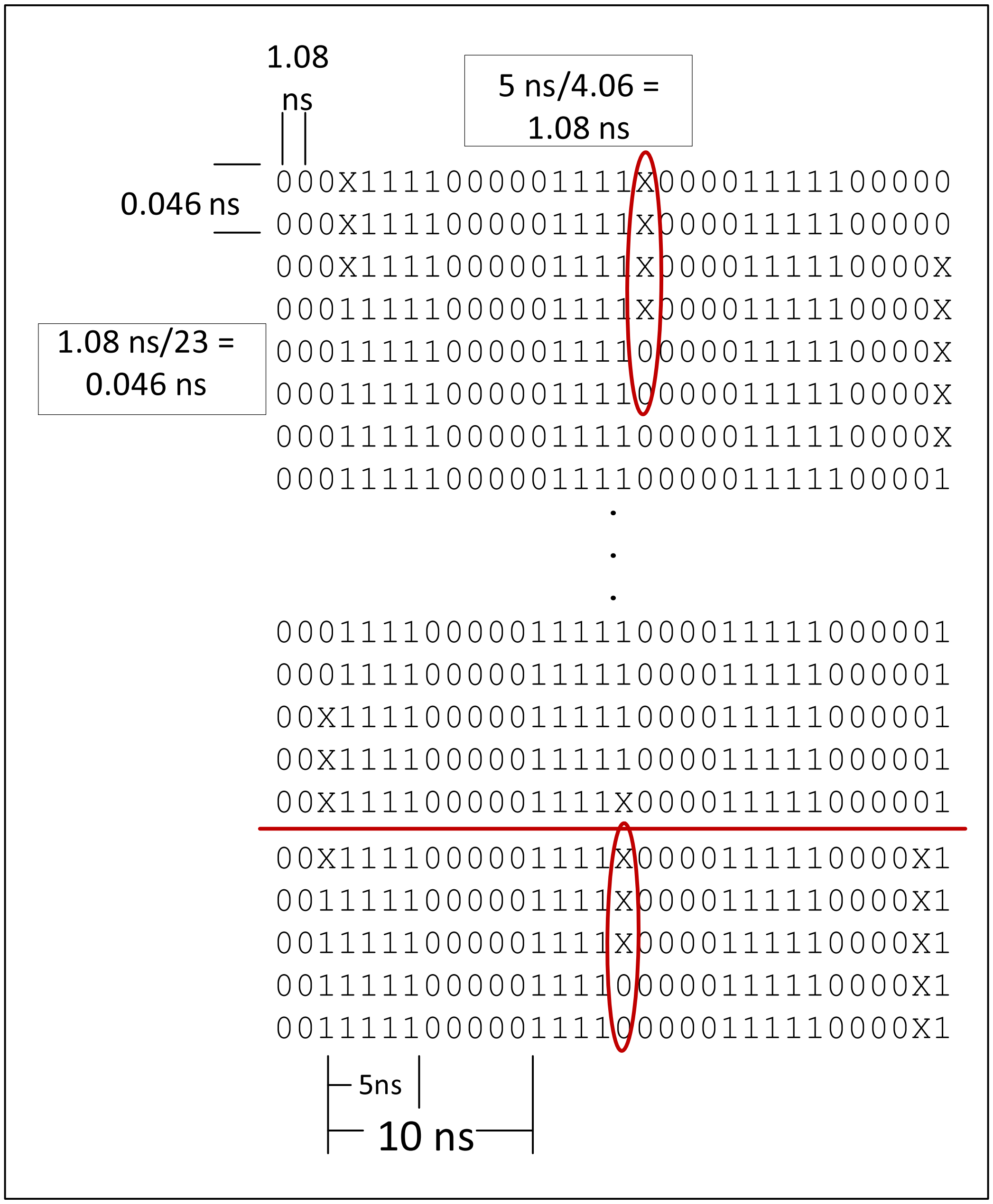
These steps are repeated 30 times. Then the circuit asserts a signal indicating that the ring oscillator data is ready. The process halts until the BRAM data is read and circuit receives an external signal, indicating the data has been read. Then it enters in the glitch detection part.

In glitch detection, the 4 input change circuit, explained in Chapter 3 is used as the glitch circuit. In one clock cycle, the first 4 bits value is applied (0000). In the next clock cycle a value, whose all 4 inputs are changed from the previous inputs is applied (1111). Then the glitch signal is captured by detection circuit. The sampled data is stored in BRAM and delay is changed in variable delay circuit. Again, in next cycle 0000 is selected. These steps are repeated for 30 times. Then glitch data ready signal is asserted and again the process will halt until the glitch data is read and it receives the external signal for it. Then the design goes back to the starting state.

Once the calibration process is done and the ring oscillator frequency is known, we can bypass the ring oscillator stage. Hence, from the starting state, the design can directly enter in the glitch detection stage just by applying an external signal indicating start of glitch detection. The same glitch detection process explained above can be repeated for different 4 inputs change signals.

5.3 Simulation and Analysis

We first verified the design by simulation. Since we could not get output from the ring oscillator in simulation, we applied periodic signal with 10ns period as ring oscillator signal. A sample of the output for calibration is shown in Fig. 5.4. We analyzed the obtained data for the circuit calibration. We observed that the circled sequence XXX00 at the bottom is overlapping with top of the next column. It is same for the other columns also. Hence, to avoid overlapping we removed last five rows from the output, which leaves 23 rows. We also observed that the average number of consecutive 0's and 1's (considering X's are combined in sequence of 0's and 1's at



```
000X1111000001111X000011111100000
000X1111000001111X000011111100000
000X1111000001111X00001111110000X
00011111000001111X00001111110000X
000111110000011110000011111100000X
00011111000001111000001111110000X
00011111000001111000001111110000X
000111110000011110000011111100001
                 .
                 .
                 .
0001111000001111100001111110000001
0001111000001111100001111110000001
00X111100000111110000111111000001
00X111100000111110000111111000001
00X11110000011111X0000111111000001
00X1111000001111X00001111110000X1
0011111000001111X00001111110000X1
0011111000001111X00001111110000X1
00111110000011110000011111100000X1
00111110000011110000011111100000X1
```

Figure 5.4: Calibration Example

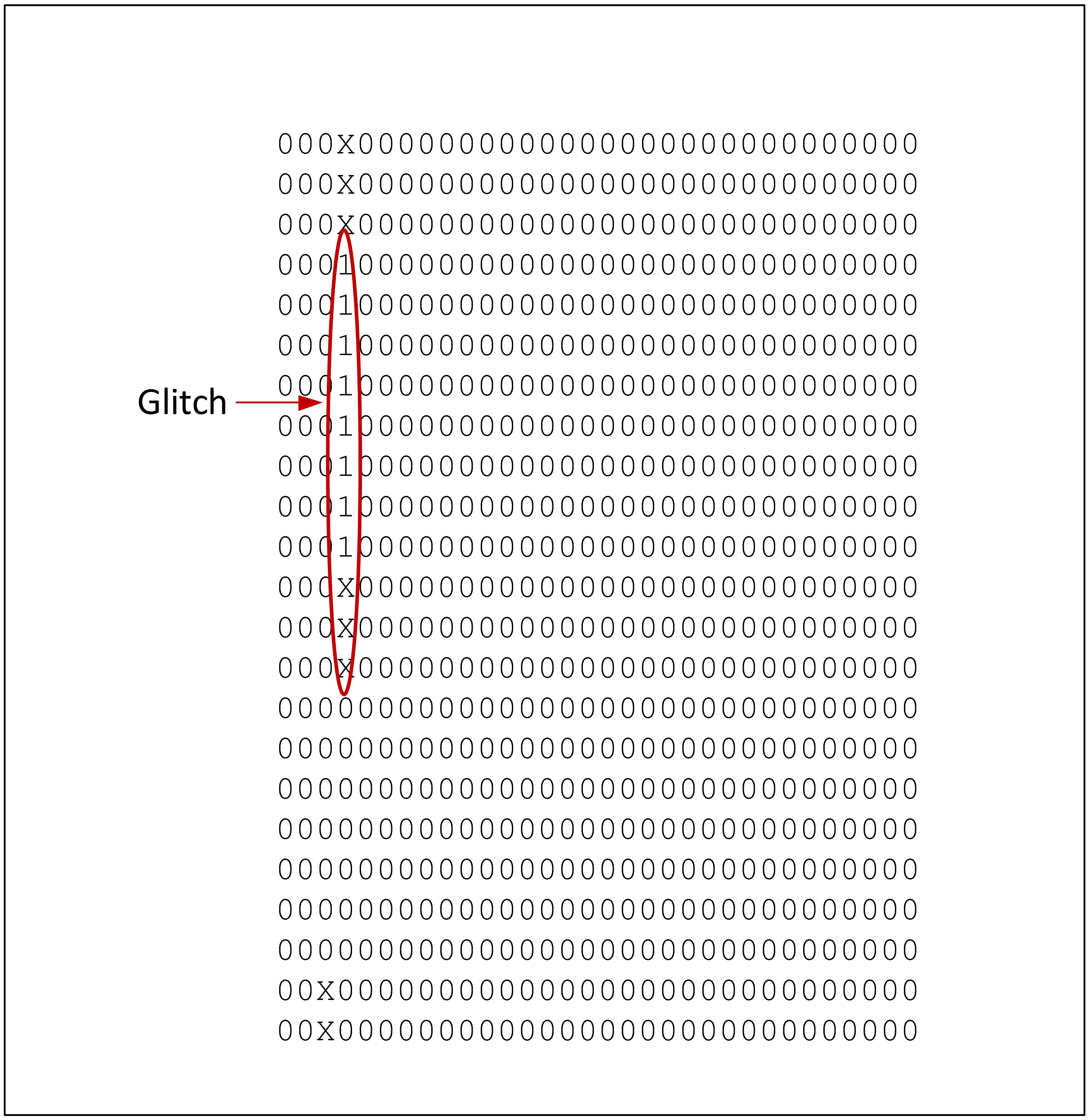
the end) is 4.6. The clock period for the circuit is 10 ns. A time period has two logic values 0 and 1 and considering 50% duty cycle, we can say that the sequence of 4.6 0's or 1's occurs in 5 ns period. Dividing 5 ns by 4.6 gives us the sampling delay of detection circuit of **1.08ns**. Now, dividing 1.08ns with 23 gives the delay difference of the variable delay circuit, which is **0.046ns**.

5.4 Observation

For the simulation, we applied the input values to the 4 input glitch detection circuit using testbench. When we applied output of the 4 inputs change glitch circuit to the detection circuit, we observed a glitch as shown in the Fig. 5.5. We found a glitch with **0.506 ns** width using the calibration obtained by ring oscillator. As shown in Chapter 3, We found the maximum width of glitch of **32 ps** from 4 input change glitch circuit. Moreover, we found only one glitch pulse from the simulation, while the 4 inputs change circuit produced two glitch pulses. Hence, we forwarded our experiments to on-board.

5.5 On Board Glitch Detection

First, we obtained on-board ring oscillator frequency of **101.85MHz** (9.8 ns clock period). Then we obtained the ring oscillator output for the circuit calibration as explained in the previous section. We found the sampling delay of **1.2ns** for detection circuit and the delay difference of **0.054 ns** for variable delay circuit. For the on-board glitch detection, we applied an input change of 0000 $\rightarrow$ 1111 to the 4 inputs change circuit through computer via USB interface. The BRAM stores the output data obtained from the glitch detection circuit, which is sent back to computer via USB for post processing and waveform generation.

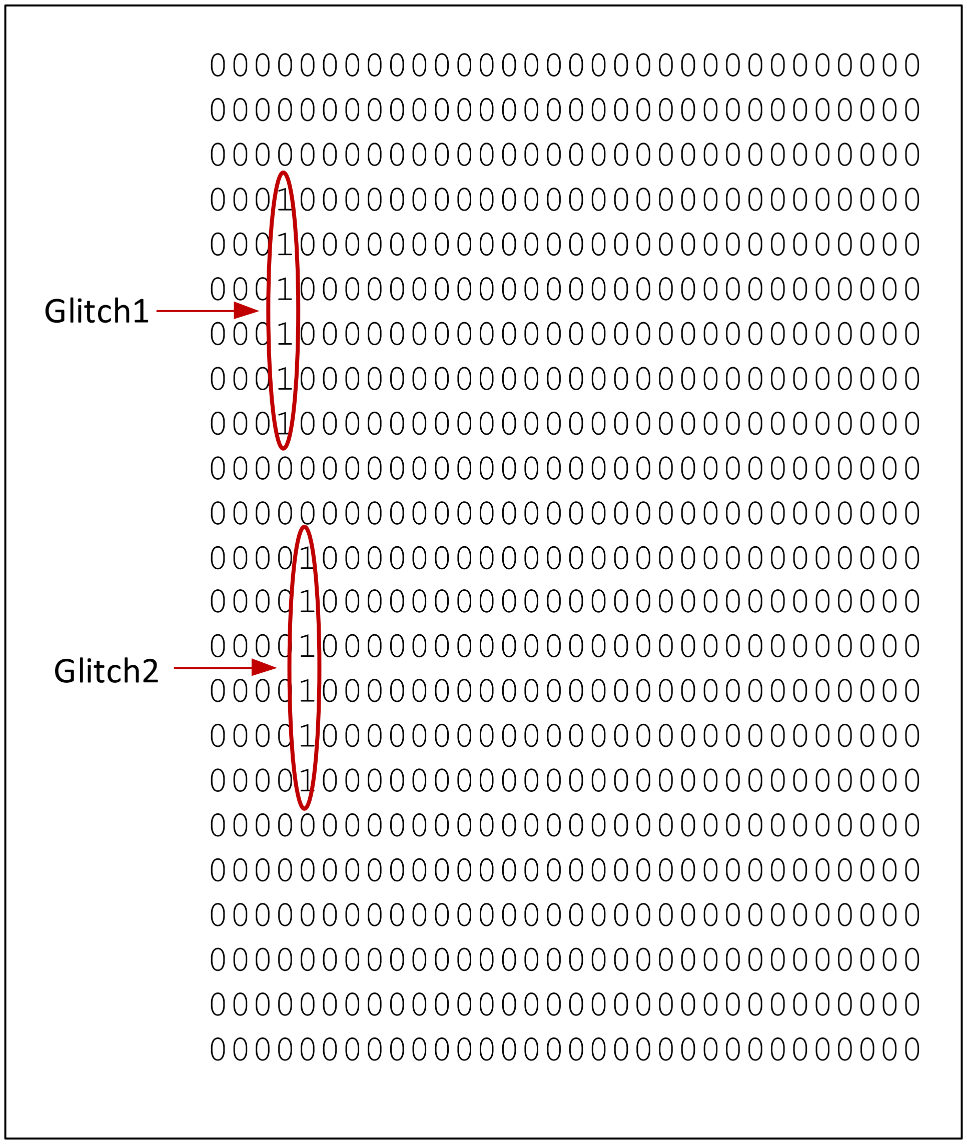


```
000X0000000000000000000000000000
000X0000000000000000000000000000
000X0000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
000X0000000000000000000000000000
000X0000000000000000000000000000
000X0000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00X00000000000000000000000000000
00X00000000000000000000000000000
```

Figure 5.5: Glitch Output from Simulation

5.6 Observation

From the on-board glitch measurement, we observed 2 glitch pulses at the output of the glitch detection circuit similar to the 4 inputs change circuit. As shown in the Fig. 5.6, we also observed the glitches with width of **0.324ns**. We can say that the simulation based glitch detection can not detect all the glitch pulses, which we found from on-board glitch detection.



```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00010000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00001000000000000000000000000000
00001000000000000000000000000000
00001000000000000000000000000000
00001000000000000000000000000000
00001000000000000000000000000000
00001000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Figure 5.6: Glitch Output from On-board Implementation

Chapter 6: Conclusion

Glitches are unproductive signal transitions, which increase the overall power consumption of the device. They also impose a security threat if the device is used for critical operation. The glitch elimination techniques require precise detection of possible glitches in order to remove glitches from the digital circuits implemented on hardware platforms such as FPGAs. The post place and route simulation provides glitch detection, but it cannot take process variations of the FPGA into account. Moreover, its accuracy depends on the simulation model only, which might not expose all the glitches from the implemented circuit. In this research we introduced a novel approach for glitch detection in hardware implementations on FPGAs, which does not merely detect the presence of glitches, but also captures the glitch waveform and provides the information about the position and the width of glitches in the design. We also propose a methodology to increase the resolution of the captured waveform and also show the calibration process required to obtain accurate values for width of glitches occurring in the hardware implementations on a given FPGA. Our results indicate that we can reliably reproduce the glitch waveform, which the simulation method could not. Our design has the resolution of **0.054 ns** and can reliably detect glitches with a width as small as **324 ps** on a Spartan-3E FPGA.

Bibliography

Bibliography

[1] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 203 –215, feb. 2007.

[2] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in virtex&#8482;-ii fpga family," in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, ser. FPGA '02. New York, NY, USA: ACM, 2002, pp. 157–164. [Online]. Available: http://doi.acm.org/10.1145/503048.503072

[3] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *11th ACM International Symposium on Field Programmable Gate Arrays.* Association for Computing Machinery (ACM), Feb 2003, pp. 175–184.

[4] J. Kao, S. Narendra, and A. Chandrakasan, "Subthreshold leakage modeling and reduction techniques [ic cad tools]," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, nov. 2002, pp. 141 – 148.

[5] H. Eriksson and P. Larsson-Edefors, "Glitch-conscious low-power design of arithmetic circuits," in *IEEE International Symposium on Circuits and Systems.* IEEE, 2004, pp. 281–284.

[6] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient fpgas," in *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, ser. FPGA '03. New York, NY, USA: ACM, 2003, pp. 175–184. [Online]. Available: http://doi.acm.org/10.1145/611817.611844

[7] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in fpgas using region-constrained placement," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, ser. FPGA '04. New York, NY, USA: ACM, 2004, pp. 51–58. [Online]. Available: http://doi.acm.org/10.1145/968280.968289

[8] J. Lamoureux, G. G. Lemieux, and S. J. Wilton, "GlitchLess: An active glitch minimization technique for FPGAs," in *15th International Symposium on Field Programmable Gate Arrays - FPGA07*, ACM/SIGDA. New York, NY, USA: ACM, Feb 2007, pp. 156–165.

[9] J. Lamoureux, G. Lemieux, and S. Wilton, "Glitchless: dynamic power minimization in FPGAs through edge alignment and glitch filtering," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 11, pp. 1521–1534, Nov 2008.

[10] E. Kusse and J. Rabaey, "Low-energy embedded fpga structures," in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, aug. 1998, pp. 155 –160.

[11] Z. Chen, S. Haider, and P. Schaumont, "Side-channel leakage in masked circuits caused by higher-order circuit effects," in *ISA 2009*. Berlin, Germany: Springer Verlag, june 2009, pp. 327–336.

[12] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked cmos gates," in *Proceedings of the 2005 international conference on Topics in Cryptology*, ser. CT-RSA'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 351–365. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30574-3_24

[13] D. Suzuki and M. Saeki, "Security evaluation of dpa countermeasures using dual-rail pre-charge logic style," in *Proceedings of the 8th international conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 255–269. [Online]. Available: http://dx.doi.org/10.1007/11894063_21

[14] "Dpa leakage models for cmos logic circuits." in *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, 2005, pp. 366–382. [Online]. Available: http://www.iacr.org/cryptodb/archive/2005/CHES/786/786.pdf

[15] P. Schaumont and K. Tiri, "Masking and dual-rail logic don't add up," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, ser. Lecture Notes in Computer Science (LNCS), P. Paillier and I. Verbauwhede, Eds., vol. 4727. Heidelberg: Springer, 2007, pp. 95–106.

[16] W. Shum and J. Anderson, "Fpga glitch power analysis and reduction," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, aug. 2011, pp. 27 –32.

[17] J. H. Anderson and F. N. Najm, "Switching activity analysis and pre-layout activity prediction for fpgas," in *Proceedings of the 2003 international workshop on System-level interconnect prediction*, ser. SLIP '03. New York, NY, USA: ACM, 2003, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/639929.639934

[18] H. Shin, N. Zang, and J. Kim, "Stochastic glitch estimation and path balancing for statistical optimization," in *SOC Conference, 2006 IEEE International*, sept. 2006, pp. 85 –88.

[19] A. Sayed and H. Al-Asaad, "A new statistical approach for glitch estimation in combinational circuits," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, may 2007, pp. 1641 –1644.

[20] L. Cheng, D. Chen, and M. Wong, "GlitchMap: an FPGA technology mapper for low power considering glitches," in *2007 44th ACM/IEEE Design Automation Conference*. IEEE, 2007, pp. 318–323.

[21] F. Najm, "Transition density: a new measure of activity in digital circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 310 –323, feb 1993.

[22] ——, “Power estimation techniques for integrated circuits,” in *Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on*, nov 1995, pp. 492 –499.

[23] A. Gaffar, J. Clarke, and G. Constantinides, “Modeling of glitch effects in FPGA based arithmetic circuits,” in *5th IEEE International Conference On Field Programmable Technology.* IEEE, Dec 2006, pp. 349–352.

[24] F. Najm, “A survey of power estimation techniques in vlsi circuits,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 446 –455, dec. 1994.

[25] D. Zhijian, W. Houjun, and L. Bing, “Research and implementation of a glitch capture technology,” in *Testing and Diagnosis, 2009. ICTD 2009. IEEE Circuits and Systems International Conference on*, april 2009, pp. 1 –3.

[26] J. Anderson and F. Najm, “Power-aware technology mapping for lut-based fpgas,” in *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on*, dec. 2002, pp. 211 – 218.

[27] Q. Dinh, D. Chen, and M. D. Wong, “A routing approach to reduce glitches in low power FPGAs,” in *International Symposium on Physical Design, ISPD’09.* ACM, March 2009, pp. 99–105.

[28] T. Czajkowski and S. Brown, “Using negative edge triggered FFs to reduce glitching power in FPGA circuits,” in *44th ACM/IEEE Design Automation Conference.* IEEE, Jun 2007, pp. 324–9.

[29] S. Cromar, J. Lee, and D. Chen, “FPGA-targeted high-level binding algorithm for power and area reduction with glitch-estimation,” in *46th ACM/IEEE Design Automation Conference (DAC-2009).* IEEE, July 2009, pp. 838–843.

[30] J. Leijten, J. van Meerbergen, and J. Jess, “Analysis and reduction of glitches in synchronous networks,” in *European Design and Test Conference, 1995. ED TC 1995, Proceedings.*, mar 1995, pp. 398 –403.

[31] A. Raghunathan, S. Dey, and N. Jha, “Register transfer level power optimization with emphasis on glitch analysis and reduction,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1114–1131, Aug 1999.

[32] L. Benini, G. De Micheli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, “Glitch power minimization by selective gate freezing,” in *1999 International Symposium on Low Power Electronics and Design*, vol. 8. IEEE, Jun 2000, pp. 287–298.

[33] *Spartan-3 Generation, FPGA User Guide*, Ug331 (v1.2) ed., Xilinx, Inc., Apr 2007.

[34] *Constraints Guide*, Ug625 (v11.4) ed., Xilinx, Inc., Dec 2009.

Curriculum Vitae

Kinjalben Shah received her B.S. degree in Electronics and Communication from Gujarat University, India. She is currently working towards her M.S. degree in Computer Engineering at George Mason University (GMU), Farifax, VA and planning to complete by Spring 2013. She joined Cryptographic Engineering Research Group (CERG) in Spring 2010. Her research interests include Computer Architecture and FPGA based Embedded Systems. She also has hands-on experience working on Electronic Power Conditioning systems at Indian Space Research Organization (ISRO). She is currently employed at Intel Corporation as a Component Design Engineer.