POLITECNICO DI TORINO

MASTER OF SCIENCE THESIS

Design, reliability evaluation, and hardening of vision-oriented hardware accelerators

Supervisor:

Prof. SONZA REORDA MATTEO

Author:

Oguz Sensoz

Co-supervisors:

Dr. GUERRERO BALAGUERA JUAN DAVID

Dr. RODRIGUEZ CONDIA JOSIE ESTEBAN

Master of Science degree in

Electronics Engineering

Department of Electronics and Telecommunications

February 2025

ABSTRACT

An increase in the complexity of electronic systems leads to less tolerance of such systems for performance degradation and safety hazards. Therefore, guaranteeing the reliability of electronic systems is crucial for safety-critical applications. It is needed to identify any kinds of errors and the possible origin of the error as early as possible and find solutions not to sacrifice for performance.

An error occurrence can depend on different reasons including manufacturing defects, aging, environmental interruption, and so on. Some of these errors can be detected and corrected after production however some of them may also arise after use. This may cause a failure if the designer is not aware of such cases while designing the product. There might be such cases where even engineers can not intervene to fix the error for instance space applications. Hence, the electronic systems should be capable of rectifying the errors or be resistant to the errors.

Modeling faults and analyzing their effects are significant for performance. A fault can be defined as a representation of a defect that is unexpected behavior between a planned design and an implemented design. There are different fault models to test the designs. The thesis focuses on stuck-at and transient fault models to test hardware.

The thesis studies the reliability assessment of hardware faults affecting a stereo core based on census transform. The accelerator calculates the depth information from two images of the same scene but at different angles. The evaluations were conducted in a set of images obtained from Middlebury Stereo Datasets. Four scenes are used: Tsukuba, Cones, Venus, and Teddy. The accelerator creates a disparity map showing the depth of the objects. The Middlebury Stereo Datasets also provide ground-truth images for the scenes and they are used to calculate accuracy, peak-signal-to-noise ratio (PSNR), and signal-to-noise ratio (SNR). These are also the metrics used to evaluate the impact of faults on the output images generated by the accelerator.

The accelerator is based on census transform which has different configurations. Kernel size (5x5, 7x7, and 9x9) and the number of neighborhoods (9 configurations) are features for the census transform [1]. The provided hardware was implemented with a 7x7 kernel and uniformly distributed neighborhoods. To gain a comprehensive view, all combinations of the features are implemented since the resilience to a fault for one configuration may not be the same for another. In other words, faults that do not arise in one configuration may appear.

However, the quality of the dispart map can also be different. Therefore, all configurations of the census transform are implemented and adapted in the accelerator. The metrics were calculated for four scenes with Matlab. This forms the initial part of the thesis. The metrics for fault-free simulations are present at this stage.

After that work, an analysis for fault-injected hardware starts. Since there are a lot of configurations, an automated framework is needed to expedite the injection process. The framework needs a fault list that stores locations where the fault will be injected. Stuck-at faults are implemented via the Questa simulator.

The framework takes the selected hardware and then compiles it. A script reads the fault list line by line and for each fault, it simulates the selected hardware. The output of the accelerator, a disparity map, will be compared with the ground-truth image with the help of a Python script which calculates the metrics and writes them down in a CSV file.

The creation of the fault list is completed after several trials. The first list targets locations that are intuitively selected. It requires an understanding of the hardware. The most significant bits of signed additions and subtractions are mostly targeted. There are also comparators in the hardware and some bits are also targeted. Since there are millions of bits, it is impossible to simulate all of them within a certain period. That is the reason for fault list creation. Nevertheless, the time is still limited to simulate all designs. At this point, a second list emerged. This list targets the most significant bits of outputs of all modules in the design. The aim is to identify the effect of sign bits in the circuit. Finally, simulations were conducted as time permitted. Metrics are calculated for 7x7 windows for all neighborhood configurations. As a result, sign bits make a difference for stuck-at-0 fault for all designs. The second stage of the thesis is concluded with these simulations.

Finally, the thesis seeks to solution to accelerate the simulations. The solution is to migrate the framework into the FPGA environment. As it is apparent, FPGA can enable the simulations to run almost in real-time. Hyper FPGA, Trenz SOM TE0803-03-4BE11-A, is used for the purpose. However, it is provided to us by the Multidisciplinary LABoratory (MLAB) from The Abdus Salam International Centre for Theoretical Physics (ICTP, Italy). The framework was introduced by them. The connection to hyper FPGA is provided to us via Jupyter Notebook.

The framework aims to connect any custom design into FPGA via the communication block IP core (core ComBlock) which provides interfaces such as registers and FIFOs to the

programmer of the programmable logic (PL). It helps to bypass the complexity of the bus provided by the processing system (PS).

This stage plans to set hardware for: (i) input image selection, (ii) location of the fault injection/s, and (iii) collection of statistics automatically. These require modifications on hardware and additional controller hardware for the status of the hardware.

The hardware needs to be inserted with saboteurs which are the small hardware to select fault types (stuck-at-0, stuck-at-1, and transient fault specifically single-event-upset (SEU)). Besides the saboteurs, a shift register has to be inserted to store and enable signals. These are inserted into comparator modules of the accelerator for trial since it is a small hardware. The injection process is also done automatically with a Python script. After successful compilation, the controller module is designed.

The controller is a hardware component that lets the programmer know about the status of the hardware. It establishes a kind of handshake protocol between the programmer and the FI infrastructure inside the target hardware. For instance, if the hardware received the input image and is ready to start the simulation or it concludes the simulation. This type of message is important to obtain reliable results. Then the accelerator with the controller is implemented and interfaces are also handled. This was the last stage of the thesis.

In conclusion, the FPGA framework needs to be tested for functionality which is a future work. However, the effort put into this thesis is important to test any hardware via FPGA automatically and contribute to running simulations faster. The proposed framework can contribute to the designer to test their design faster and build fault correction algorithms accordingly.

Content

1. Introduction 1

2. Background 5

2.1. Stereo Vision 5

2.1.1. The Census Transform [1] 8

2.1.2. The Sparse Census Transform 9

2.1.3. Strategies for Reliability Evaluation [10] 12

2.2. Levels of Fault Models 14

2.3. Stuck-at Fault 14

2.4. Transient Fault [13] 15

2.5. Evaluation Metrics 16

2.6. The Field Programmable Gate Array (FPGA) 19

2.6.1. The Hyper FPGA 19

2.6.2. The ComBlock [17] 20

3. Stereo Matching Accelerator [1] 24

3.1. Census Transform Module 24

3.2. Sum of Hamming Distance (SHD) Module 26

3.2.1. Hamming Distance (HD): *Number_of_ones* module 27

3.2.2. Sum of Hamming Distance (SHD): *Window_Sum* module 28

3.2.3. Comparator: *Disp_Cmp* module 29

3.3. Left-Right Consistency Check (LRCC) Module 30

4. Simulation Environment for Reliability Assessment 33

4.2. Fault-Free Simulations 34

4.3. Faulty Simulations 35

4.3.1. Fault List: Where to Inject the Fault 36

5. Emulation-Based Strategy 39

5.1. Methodology 39

5.2. Block Diagram 40
5.3. Hardware Modifications 41
5.3.1. Yosys [19] 41
5.3.2. Sabotuers and Injection 41
5.3.3. The FI Controller 45
6. Evaluation Results 52
6.1. Fault-Free Simulation Results 52
6.1.1. Fault-free Results: 5x5 Window Size 52
6.1.2. Fault-free Results: 7x7 Window Size 56
6.1.3. Fault-free Results: 9x9 Window Size 59
6.2. Faulty Simulation Results 62
7. Conclusion and Future Work 82
7.1. Future Work 83
Bibliography 85

List of Figures

Figure 1 Top: Images recorded by two cameras with significant differences in extrinsic parameters and also differences in intrinsic parameters. Bottom: Two geometrically rectified images taken at different viewing locations by two cameras installed in a crash-test hall [6]... 6

Figure 2 Stereo matching among two images of the same scene [7] 7

Figure 3 Disparity versus Distance [8] 7

Figure 4 Census transform example 8

Figure 5 Census Transform kernel configuration for 5x5 window 10

Figure 6 Census Transform kernel configuration for 7x7 window 11

Figure 7 Census Transform kernel configuration for 9x9 window 11

Figure 8 An example of a single stuck-at-fault. 15

Figure 9 The histograms of the Tsukuba image for 7x7 window size implementation (threshold is 70, the red line): a) Uniform. b) Non-redundant. c) Full. d) 16-Point. e) 12-Point. f) 8-Point. g) 4-Point. h) 2-Point. i) 1-Point. 17

Figure 10 The histograms of the image of the cone for 7x7 window size implementation (threshold is 33, the red line): a) Uniform. b) Non-redundant. c) Full. d) 16-Point. e) 12-Point. f) 8-Point. g) 4-Point. h) 2-Point. i) 1-Point. 17

Figure 11 The histograms of the venus image for 7x7 window size implementation (threshold is 30, the red line): a) Uniform. b) Non-redundant. c) Full. d) 16-Point. e) 12-Point. f) 8-Point. g) 4-Point. h) 2-Point. i) 1-Point. 18

Figure 12 TE0803-01 MPSoC module 19

Figure 13 ComBlock 21

Figure 14 ComBlock with ports 22

Figure 15 Ports of Census Transform Module 24

Figure 16 Window buffer for 7x7 census window and 13x13 correlation window 25

Figure 17 SHD Module with inputs/outputs 26

Figure 18 Stereo core architecture 27

Figure 19 Calculation of hamming distance 27

Figure 20 Computing Column Sum [1] 28

Figure 21 Computing Window Sums [1] 28

Figure 22 The architecture of the SHD module [1] 29

Figure 23 Comparator Module. n is several bits as the output of the census transform. Wh is the size of the correlation window 30

Figure 24 LRCC module 31
Figure 25 Middlebury Stereo Vision dataset: left images, right images, and ground-truth 34
Figure 26 Testbench environment for fault-free simulations 34
Figure 27 Testbench environment for faulty simulation..... 36
Figure 28 Example index for fault list..... 36
Figure 29 Proposed Framework 39
Figure 30 Block Design..... 40
Figure 31 Sabotuer injection with super saboteurs [20]..... 42
Figure 32 New ports 44
Figure 33 Saboteur for input signal 44
Figure 34 Logic for SEU 45
Figure 35 Controller signals 46
Figure 36 Status Register..... 48
Figure 37 The internal architecture of the controller..... 49
Figure 38 Timing diagram for stuck-at-fault 49
Figure 39 Timing diagram for transient fault 50
Figure 40 Tsukuba results for 5x5 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 53
Figure 41 Cones results for 5x5 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 54
Figure 42 Venus results for 5x5 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 55
Figure 43 Tsukuba results for 7x7 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 56
Figure 44 Cones results for 7x7 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 57
Figure 45 Venus results for 7x7 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 58
Figure 46 Tsukuba results for 9x9 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 59
Figure 47 Cones results for 9x9 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 60

Figure 48 Venus results for 9x9 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full. 61
Figure 49 The stuck-at-0 accuracy results of Tsukuba for uniform and 7x7 window size 63
Figure 50 The stuck-at-0 PSNR results of Tsukuba for uniform and 7x7 window size63
Figure 51 The stuck-at-0 SNR results of Tsukuba for uniform and 7x7 window size . 64
Figure 52 The stuck-at-1 accuracy results of Tsukuba for uniform and 7x7 window size 64
Figure 53 The stuck-at-1 PSNR results of Tsukuba for uniform and 7x7 window size65
Figure 54 The stuck-at-1 SNR results of Tsukuba for uniform and 7x7 window size . 65
Figure 55 Faulty Tsukuba for component 33. Stuck-at-1 fault is injected at: a) Bit-11, b) Bit-15, c) Bit-16, d) Bit-17. 66
Figure 56 Faulty Tsukuba for component 63. Stuck-at-1 fault is injected at: a) Bit-11, b) Bit-15, c) Bit-16, d) Bit-17. 67
Figure 57 The stuck-at-0 accuracy results of Cones for uniform and 7x7 window size 68
Figure 58 The stuck-at-0 PSNR results of Cones for uniform and 7x7 window size .. 68
Figure 59 The stuck-at-0 SNR results of Cones for uniform and 7x7 window size 69
Figure 60 The stuck-at-1 accuracy results of Cones for uniform and 7x7 window size 69
Figure 61 The stuck-at-1 PSNR results of Cones for uniform and 7x7 window size .. 70
Figure 62 The stuck-at-1 SNR results of Cones for uniform and 7x7 window size 70
Figure 63 Faulty cones for component 33. Stuck-at-0 fault is injected at: a) Bit-10, b) Bit-11, c) Bit-12, d) Bit-16, e) Bit-17, and f) Bit-18. 71
Figure 64 Faulty cones for component 63. Stuck-at-0 fault is injected at: a) Bit-10, b) Bit-11, c) Bit-12, d) Bit-16, e) Bit-17, and f) Bit-18. 72
Figure 65 Cones images: a) sa0 fault at the 0th-bit position for component 30. b) Fault-free image 72
Figure 66 The stuck-at-0 accuracy results of Venus for uniform and 7x7 window size 73
Figure 67 The stuck-at-0 PSNR results of Venus for uniform and 7x7 window size... 74
Figure 68 The stuck-at-0 SNR results of Venus for uniform and 7x7 window size..... 74
Figure 69 The stuck-at-1 accuracy results of Venus for uniform and 7x7 window size 75

Figure 70 The stuck-at-1 PSNR results of Venus for uniform and 7x7 window size... 75
Figure 71 The stuck-at-1 SNR results of Venus for uniform and 7x7 window size..... 76
Figure 72 Faulty venus for component 33. Stuck-at-0 fault is injected at: a) Bit-11, b) Bit-16, and c) Bit-17..... 76
Figure 73 a) Golden disparity map of Venus (97.03% accuracy). b) When a fault is injected at component 10, bit position 1 for *sa0* (95.42% accuracy). 76
Figure 74 Accuracies for Cones, Tsukuba, and Venus images based on the number of comparators for SA1. 77
Figure 75 PSNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA1. 78
Figure 76 SNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA1. 78
Figure 77 Accuracies for Cones, Tsukuba, and Venus images based on the number of comparators for SA0. 79
Figure 78 PSNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA0. 79
Figure 79 SNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA0. 80
Figure 80 Example connection of the controller with the ComBlock..... 83

List of Tables

Table 1 Thresholds for the images 16
Table 2 Accuracy, PSNR, and SNR values for Tsukuba images, 5x5 window size 52
Table 3 Accuracy, PSNR, and SNR values for Cones images, 5x5 window size 53
Table 4 Accuracy, PSNR, and SNR values for Venus images, 5x5 window size 54
Table 5 Accuracy, PSNR, and SNR values for Tsukuba images, 7x7 window size 55
Table 6 Accuracy, PSNR, and SNR values for Cones images, 7x7 window size 56
Table 7 Accuracy, PSNR, and SNR values for Venus images, 7x7 window size 57
Table 8 Accuracy, PSNR, and SNR values for Tsukuba images, 9x9 window size 58
Table 9 Accuracy, PSNR, and SNR values for Cones images, 9x9 window size 59
Table 10 Accuracy, PSNR, and SNR values for Venus images, 9x9 window size 60
Table 11 The metrics for component 63, *sa0*, *sa1*, and bit positions 11, 15, 16, and 17 67

Chapter 1

1. Introduction

The rapid advancement in embedded systems has typically transformed modern technology, enabling innovative solutions across various industries. Among these advancements, vision accelerators have gained considerable interest in extracting meaningful information. Besides that, many machine vision techniques have been developed through the decades for that purpose. The vision accelerators are specialized hardware that implement efficient artificial vision algorithms. The aim is to reduce power consumption and computational load. They inherently enable parallel computing.

One type of vision accelerator is a stereo-vision accelerator which estimates depth information through disparity calculations. Stereo vision systems rely on two or more images of the same scene. The images are captured by cameras placed a fixed distance apart. The stereo vision technique is not only used for extracting depth information but also 3D mapping, machine vision and autonomous navigation such as drones. However, it requires lots of data to be processed and this creates challenges related to computational complexity, memory bandwidth, and latency [2]. To deal with these challenges, stereo vision accelerators incorporate optimized architectures along with dedicated boards such as NVIDIA Jetson Nano, Google Coral Dev Board, and Avnet Ultra96-V2.

The NVIDIA Jetson Nano, for instance, is a small artificial intelligence (AI) computer delivering compute performance to run modern AI algorithms at unique size, power and cost. The 128-core Maxwell GPU provides substantial parallel processing capabilities. Image signal processor (ISP) and JPEG Processing Block are, for example, specialized hardware to support computational workload [3].

Drones are an example applications of autonomous systems to apply the stereo vision technique. Detecting and avoiding obstacles, autonomous navigation is required to accurately perceive depth and construct 3D map of surroundings in real-time. There are powerful hardware as mentioned above and cost-effective algorithms to make drones work. However, there is another crucial factor that needs to be considered. It is the reliability.

Since autonomous systems are expected to work without human intervention, any failure in the system can cause a disaster in terms of safety, mission, and so on. In autonomous

cars, for example, a failure in reliability may lead to jeopardy for safety of the passengers. In case of drones, it causes mission abortion and loss of equipment. Therefore, ensuring reliability in such systems is not only requirement but also critical factor for public trust.

Reliability is a continuity of correct service [4]. Reliability ensures to maintenance of the functionality and the performance of the system even in the presence of faults or failures. It is also highly related to other attributes of secure computing such as availability, safety, and security. As systems advance and become more complex, ensuring reliability becomes more significant. Therefore, achieving reliable systems is a fundamental goal of the designers for the trustworthiness and success of the technology. Fault forecasting can be conducted for an evaluation of the system behavior in the presence of the fault.

Assessing the reliability of hardware accelerators is crucial, for identifying critical hardware structures and consequently devising fault tolerance mechanisms to mitigate such fault effects. The occurrence of the error can depend on different reasons including manufacturing defects, aging, environmental interruption, and so on. Some of these errors can be detected and corrected after production however some of them may also arise during the in-field operation of the system. This may cause a failure if the designer is not aware of such cases while designing the product. There might be such cases where even engineers can not intervene to fix the error for instance space applications. Hence, the electronic systems should be capable of rectifying the errors or be resistant to the errors.

When a product is delivered to the user, it may still produce the wrong output due to some problems mentioned before. To reduce the effect of such errors, fault analysis is needed. The designer deliberately introduces a fault in the design and examines the behavior of the hardware. With the help of some techniques, side effects can be minimized and eventually disappear. The goal of the thesis is to develop an automated framework for fault injection for any hardware design. Furthermore, understanding and evaluating the effects of such faults are also aimed.

This thesis focused on evaluating the reliability assessment of faults affecting a stereo vision accelerator for safety-critical applications. The experimental evaluations were first conducted at the RTL level by systematically injecting faults on specific locations of a selection of internal components of the accelerator after performing an architectural analysis. In addition, different versions of census transform were implemented to study the impact of such transformations on the reliability of the accelerator. To quantify the impact of fault on the stereo

vision accelerator. This thesis adopted three different evaluation metrics (Accuracy, PSNR, and SNR). The metrics are calculated on the obtained images for each injected fault.

All the experiments were conducted in an FI framework that allows the injection of faults in the internal structures of the target accelerator. In this thesis, the first approach of the FI framework was developed using Questasim simulator to simulate permanent faults. Unfortunately, the simulation time required by this FI tool resulted in excessive simulation time, requiring between 15 to 30 minutes per fault, which limited the number of faults and the scenario under evaluation in a reasonable amount of time.

To tackle this problem, a new alternative strategy was devised by adopting emulation strategies. Emulation FI inserts additional circuits, called saboteurs, inside the RTL or gate-level descriptions of the target hardware that later can be implemented on FPGA devices. This FI method allows the activation of a given fault during the runtime of the accelerator making the evaluations faster. In this thesis, the first version of the emulation infrastructure is presented demonstrating that the simulation time is 27 times faster than the simulation approaches.

Finally, simulations are run with and without a fault injection. A reference metrics are calculated with the fault-free simulations by comparing output disparity maps with their ground-truth images. Then, the faults are injected, and the metrics with related images are presented. In addition, the accelerator is adapted to the emulation-based simulation framework to benefit from the simulation times. That part relies on hardware modifications with a sabotuer injection.

Chapter 2

2. Background

To analyze any hardware for potential faults that may arise before and after production, engineers model the fault. Modeling is a method that describes how hardware behaves under the fault and it helps in developing the architecture for fault-free usage. To understand the impact of the fault, fault injection framework is developed.

There are different models however some of them are common fault models like stuck-at, transient, and single-event upset. These methods served as a fault detection on the thesis to meet reliability and performance standards.

2.1. Stereo Vision

It is not possible to estimate the depth of an object in an image with one camera. Stereo vision technique is used capture such information. There are at least two camera taking picture of the same scene from different angle. The human visual system is a proof. The difficulty for this technique is to determine corresponding points in one image in the other.

The system consists of four main steps: calibration, rectification, stereo-matching, and triangulation [5].

- Calibration: It is related to calculating intrinsic and extrinsic parameters of the camera. Intrıinsic parameters' calculations occur simultaneously with the estimation of extrinsic parameters (the pose of the camera concerning a known calibration target. Intrinsic parameters of the camera estimate parameters such as focal length, optical center, and lens distortion. On the other hand, extrinsic parameters depend on the relative position between cameras.
- Rectification [6]: Matching the corresponding points in pairs of the input images is the complexity of stereo vision. To reduce the complexity, the rectification process involves warping the input image pairs such that they are recorded in canonical stereo geometry. Figure 1 shows an example of the process.
- Stereo matching [7]: As the name indicates, the aim is to find the same points in two images. Assume that in Figure 2, left image (B) and right image (L) are the

match images. The pixel p with its neighborhood defined by a square window in the left image is compared with the pixel q with its neighborhoods on the epipolar line. A matching operation is performed on the same row due to canonical stereo geometry. How to know if pixels match is based on algorithms like the sum of absolute differences (SAD), census transform with hamming distance, and normalized cross-correlation.

- Triangulation: It is a technique used to determine the depth of the object in the scene. By using image pairs and disparity of the same points, the coordinates of the point in reality are calculated.

Figure 1 Top: Images recorded by two cameras with significant differences in extrinsic parameters and also differences in intrinsic parameters. Bottom: Two geometrically rectified images taken at different viewing locations by two cameras installed in a crash-test hall [6]

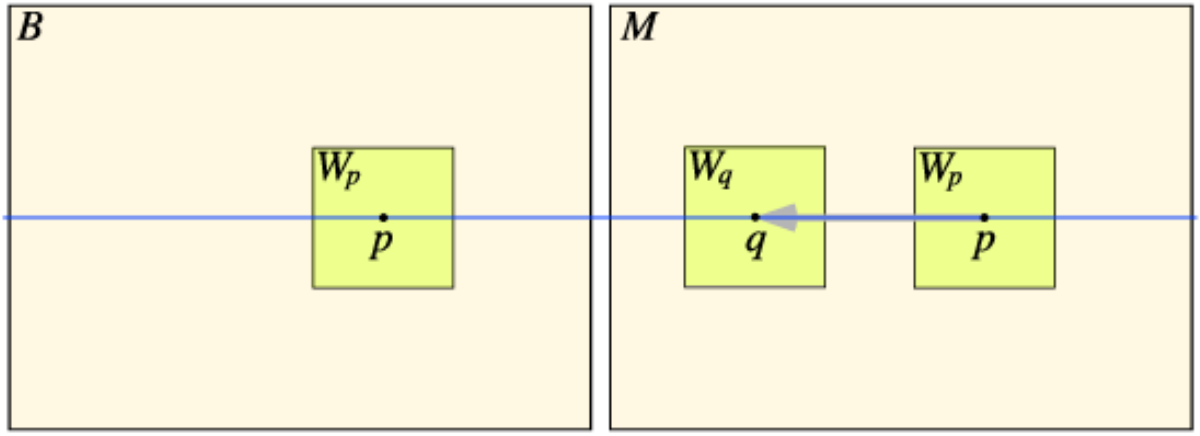

Figure 2 Stereo matching among two images of the same scene [7]

The depth is inversely proportional to disparity as shown in Equation 1 where f is the focal length of the camera, d is the physical size of a pixel in the camera sensor CMOS/CCD, T is the baseline distance between the center of the left and right cameras, and D is a disparity [8]. Figure 3 illustrates the relationship between the disparity and the distance of the object. When an object is close to the camera, the disparity gets higher.

$$Z = \frac{f}{d} \times \frac{T}{D}$$

Equation 1 Distance between point P and camera center

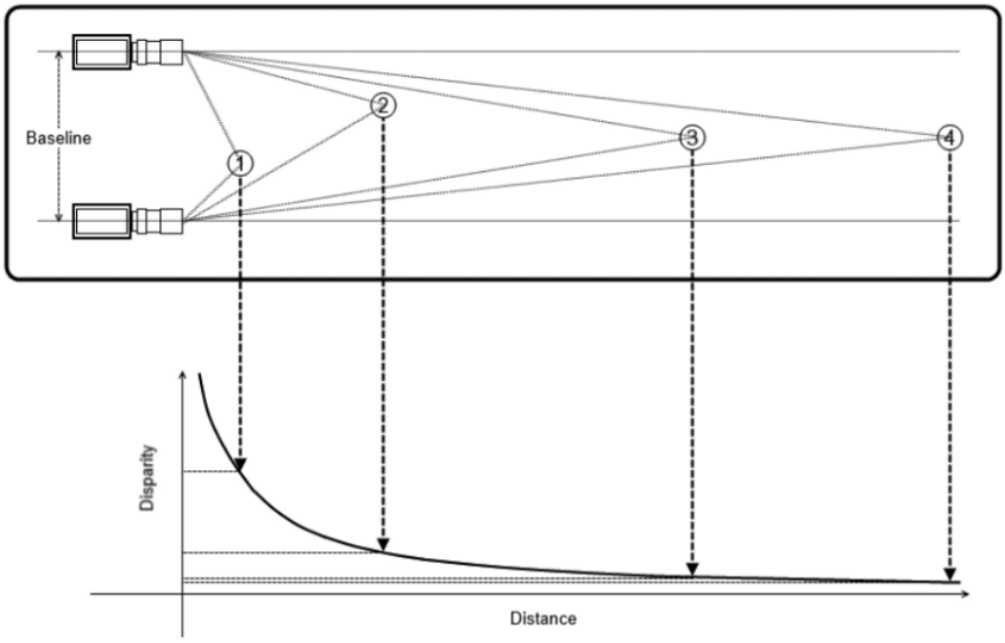

Figure 3 Disparity versus Distance [8]

Another observation from Equation 1 is that the distance between cameras T is proportional to the disparity. The disparity is small when the cameras are close to each other for the same point P.

Overall, these two parameters are closely related to depth estimation for stereo vision before applying stereo match algorithms. However, the thesis uses the Middlebury dataset [9] which provides image pairs with their ground-truth images. So, camera calibration and rectification steps are not the concern of the thesis. The following two chapters discuss the stereo-matching algorithm based on census transform. It is also the algorithm implemented in the stereo vision accelerator.

2.1.1. The Census Transform [1]

Census transform is a kernel-based nonlinear transform used for feature extraction in image processing applications. It is used to extract pixel intensity for depth information of an image. The transform compares pixel intensities with a reference pixel value and generates a binary number for each pixel in the image according to the size of the kernel. Figure 4 gives an example of a 3 by 3 kernel where each value around P1 is compared with P1 and if they are less than or equal to the P1, the output value is set to 1, otherwise 0. This is how the census vector is generated.

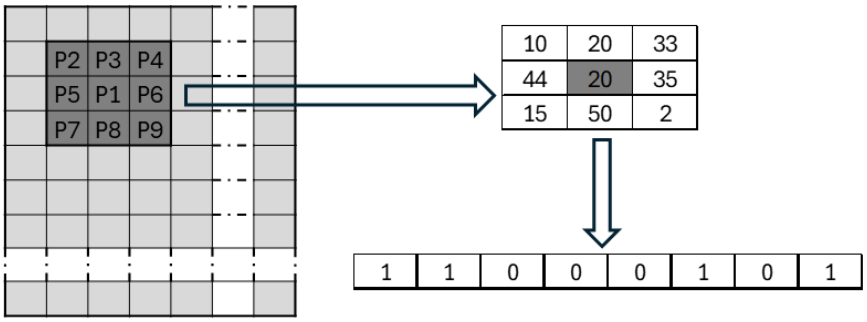

Figure 4 Census transform example

The disadvantage of the census transform is the large size of the census vector in terms of the number of bits. It means that a number of pixel comparisons is performed by the transform and this affects the amount of the hardware resources. Assuming a 9 by 9 kernel size, there are 80 pixels to be compared with the central pixel. The length of the census vector also affects the amount of hardware resources for calculating the hamming distance. Therefore, the requirement for the hardware resources is proportional to the size of the census vector.

The size of the census vector also determines the length of the hamming distance value as a representation number of bits which is proportional to $\lceil log_2(n+1) \rceil$ -bits for n-bit census vector. Furthermore, the summing of the distances requires adders whose sizes are also proportional to $\lceil log_2(n+1) \rceil$.

Overall, a motivation behind the sparse census transformation emerges. It reduces the census vector size without sacrificing the accuracy of correlation. In the thesis, the stereo vision accelerator is based on the sparse census transform.

2.1.2. The Sparse Census Transform

According to [1], there is redundancy in the computation of the census vector. Assuming that the census transform is computed around pixel p with comparisons of nearby pixels p'. When the census transform is computed for the pixel p' later, the transform still includes the comparison between pixels p and p'. The double computation contributes to being weighted twice the pixels. This is correct for the pixels not located on the edge of the correlation window. The pixels on the edge of the correlation window become less weighted, therefore the effects of the pixels on the edge weaken.

The census transform can be optimized to reduce the amount of required hardware during their implementation with minimal impact on the final obtained output from the accelerator. As a result, it is possible to use nine different kernel configurations configurations for the census transforms. The dark color pixels are compared to the pixel in the center of the matrix. These kernel types will be applied to the different window sizes. Authors in [1] demonstrated that these optimized kernel sizes have a limited impact on the accuracy of the accelerator. These optimizations can dramatically affect the number of resources in the design and also can play a crucial role in terms of fault tolerance features.

The literature also reports three different windows or kernel sizes for census transform: $5x5$, $7x7$, and $9x9$. It is worth noting that the same patterns can be applied to other kernel sizes. Accuracy value may not vary over window size for the same configuration of the kernel (see

Chapter 6.1). These trade-offs can affect the choice of the census transform module. The reader will also see the results under stuck-at-0 and stuck-at-1 fault simulations.

Nine kernel configurations are full (normal census transform), uniform, non-redundant, 16-point, 12-point, 8-point, 4-point, 2-point, and 1-point. Applying the configuration on the different window sizes, there are 29 kernels. These are shown in Figure 5, Figure 6, and Figure 7.

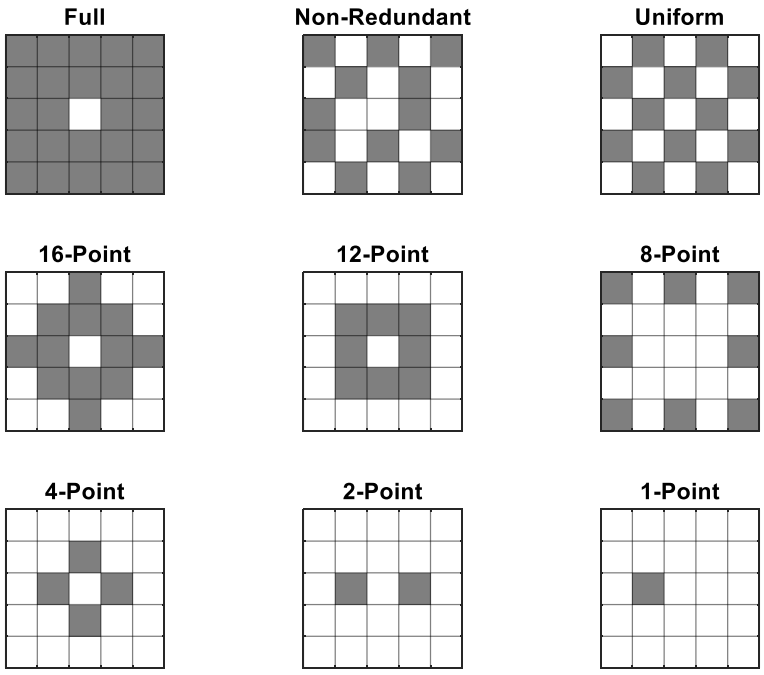

Figure 5 Census Transform kernel configuration for 5x5 window

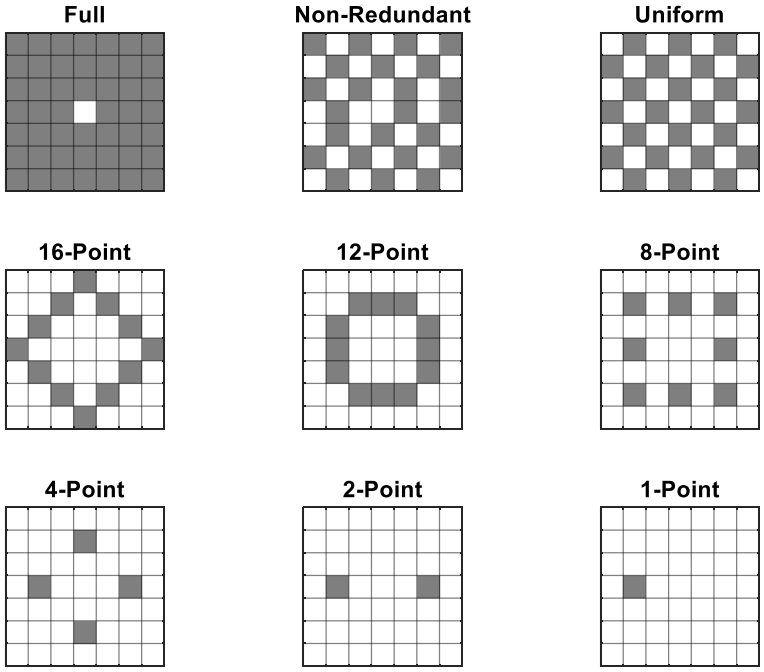

Figure 6 Census Transform kernel configuration for 7x7 window

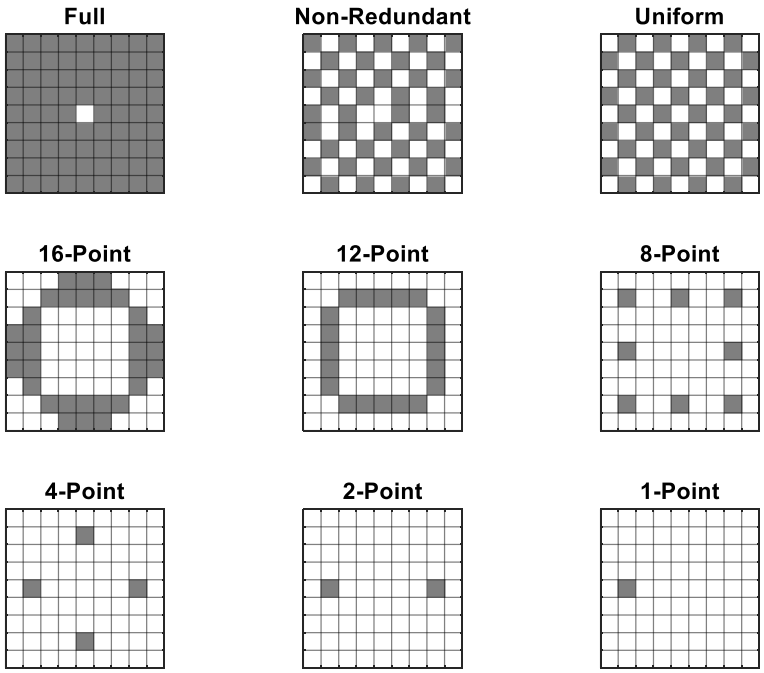

Figure 7 Census Transform kernel configuration for 9x9 window

2.1.3. Strategies for Reliability Evaluation [10]

The stereo vision accelerator is implemented with different census transform methods. The comparisons between the methods are made and the result is presented. However, it is crucial to know how the hardware behaves under a fault and how reliable the hardware is.

In this section, we present different strategies to evaluate the reliability of the systems. The reliability evaluation is usually carried out experimentally by adopting fault injection (FI) campaigns. There are four different FI strategies, i) simulation-based, ii) hardware-based, iii) software-based, and iv) emulation-based.

i) Simulation-based strategies:

Hardware description of the tested circuit is simulated with fault injected on specified locations. The fault can be injected in two methods. The tools can insert necessary components for fault injection into the tested circuit or the fault can also be inserted by simulator commands. The second method enables to modification of the signals and variables of the tested circuit.

The disadvantage of simulation-based strategies is that they are very slow. An actual circuit operation is multiple orders of magnitude faster than a simulation of the hardware description of a circuit. For instance, simulating the stereo vision accelerator with a 7x7 window size and uniform configuration for just one fault takes about 7 minutes for an image size of 384x288 in the Questa simulator. The simulation is run with simulator commands. A selected signal is targeted to stuck-at fault. When image sizes increase, as expected, the simulation requires more time.

ii) Hardware-based strategies:

With a cause of heavy ion radiation, electromagnetic interferences or other physical and environmental effects, the hardware can be disrupted for fault injection. To apply this method, it is necessary to have the final device. Therefore, the fault injection is directly applied to the final device, this is the main advantage. On the other hand, controlling the effect of physical injection is difficult. It is hard to perform the same experiment. Instead, to achieve repeatability, disruptions are emulated at the pin level rather than injecting a physical fault.

When the final device is exposed to the radiation, there is a possibility to observe transient faults. The radiation may cause one or more bit-flips at random locations which cannot be done with pin-level injection.

Finally, the tools for handling this type of injection are hardware-dependent and the setup of the tools is complicated.

iii) Software-based strategies:

Software-implemented fault injection (SWIFI) tools are used to implement the fault at the software level. The aim is to create the faults at the hardware level by using software. This is not needed to modify the hardware. SWIFI enables testing the whole system including the operating system. There are several tools to use SWIFI techniques such as CEU, EFS, and XCEPTION. The fault models can be bit-flip, data modification, stuck-at faults, and code insertion. Skipping one or more instructions or modifications of the instructions are common methods to apply such faults.

The cost is speed, memory consumption and so on. Since the fault injection campaign requires to modify the program, it causes execution overhead. For instance, EFS tools need context switches between real system processes and fault injection processes. The primary drawback of SWIFI is that the tools create faults beyond the other techniques. Assuming a 32-bit system and 2^{32} possible instruction values, a number of existing wires might fall that value. The generated faults may not cause failures or the error cannot appear during execution of the program. Therefore, generating a more useful set of faults is the challenge which also includes optimization phases during simulation.

iv) Emulation-based strategies

This technique offers an alternative solution to simulation-based strategies to reduce simulation time. Field Programmable Gate Arrays (FPGA) based logic emulation systems are used for hardware prototyping.

A real behavior of the design can be observed under the fault injection with real-time interactions. The designer has to provide the complete synthesizable VHDL description. In addition, the synthesizable VHDL description requires additional mechanisms to inject faults at specific times and locations. This may lead to overhead in the circuit if the number of fault-injectable locations increases.

2.2. Levels of Fault Models

Modeling of faults is closely related to the modeling of the circuit. In the design hierarchy, the level refers to the degree of abstraction. Manufacturing defects may not be correlated with fault models for the behavioral level which has fewer details of implementation. High-level fault models play a terrific role in simulation-based design verification than testing. However, semiconductor memories have exceptions due to simpler functionality, and an exhaustive functional test is possible.

Stuck-at faults are quite useful and popular fault models at the register-transfer level (RTL) or logic level in digital testing. Other fault models at this level are delay faults and bridging faults.

At the component levels including transistors and other lower levels, stuck-open types of faults can be used. These faults are also technology-dependent faults. Component-level faults are mostly introduced in analog circuit testing. Lastly, other models do not correspond to any level of design hierarchies. The quiescent current defect is a typical example.

2.3. Stuck-at Fault

To model stuck-at-fault, a design with its interconnections (netlist) of Boolean gates is provided. The intention is to insert the fault between interconnections. There are two types of faults: stuck-at-0 and stuck-at-1 (s-a-0 and s-a-1 respectively). It means that a line with an s-a-1 fault, for instance, has always a logic 1 regardless of the correct logic driving it.

It is possible to have multiple stuck-at faults in the circuit simultaneously. A circuit with n interconnections can have 3^{n-1} possible targets for the faults. It is assumed that each connection has three states: s-a-1, s-a-0, or fault-free. The case when all interconnections are fault-free is not counted. When n is getting large, the number of stuck-at faults explodes. Therefore, modeling only a single stuck-at fault is a common practice that reduces the number of faults to 2n at most. Furthermore, the number can be reduced by a technique known as fault-collapsing [11]. On the other hand, some faults can not propagate to the output with any input combinations. It is called an untestable fault. The presence of such faults may or may not change the input-output behavior of the circuit [12]. Some redundant logic might be introduced to the circuit and these may lead to present untestable faults.

In conclusion, the stuck-at model is very useful since it is simple, numerable, and well-supported by CAD tools. It is widely used and able to model several physical defects. However, it can not model the totality of the physical defects like open fault, delay fault, and so on. For instance, stuck-at-fault does not consider timing-related problems where the transition is correct but it violates the timing requirement like in the case of the delay fault model. Another example is that if a wire or transistor is not connected, it may lead to unpredictable floating values which cannot be modeled with the stuck-at model. One should also consider that several faults that cannot be modeled as stuck-at faults can be detected by test patterns generated to cover stuck-at faults.

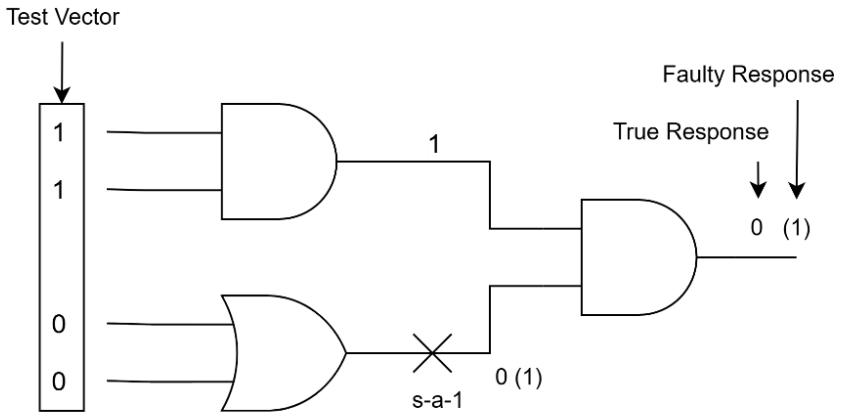

Figure 8 An example of a single stuck-at fault.

2.4. Transient Fault [13]

Faults can be permanent or non-permanent. The stuck-at fault model is a permanent type of fault which means that an inteconnection's value is fixed to some value and it can not be changed ever. On the other hand, non-permanent faults are present only part of the time and occur randomly. These faults especially appear in memory integrated circuits however, a solution to this problem is information redundancy or redundant error-correction codes.

Transient faults are a type of non-permanent fault caused by environmental conditions like cosmic rays, air pollution, humidity, temperature, power supply fluctuation, and so on. Fault models for transient faults are getting more and more important due to an increasing number of fault-tolerant applications. There are two significant fault models for transient fault: single event upset (SEU) and single event transient (SET) fault model.

SEU models a bit flip in any memory element at any clock cycle. Engineers can measure the sensitivity of the circuit or system for given input data. In a simulation environment, a bit flip can be injected and maintain this state within a specific time interval. A drawback is that the number of faults may grow rapidly. Let's assume there is an accumulator with 16-bit wide and there are two inputs with 8 bits each. Also, assume that multiplication takes 256 clock cycles. Finally, the number of possible SEUs equals to $(8 + 8 + 16) \times 256 = 8192$.

In conclusion, SEU is a model that implements faults for safety-critical applications such as space applications, avionics, and autonomous systems. There are techniques to eliminate the effect of such faults for instance error correction code (ECC) and triple modular redundancy (TMR).

2.5. Evaluation Metrics

Accuracy, PSNR, and SNR are the metrics for the comparison of the results. The measurements are calculated by comparing the ground truth of the image with the output disparity map of the accelerator.

Accuracy comes from the [14]. It is based on the percentage of bad-matching pixels. This is an average of the absolute difference between a disparity map and a ground-truth image in comparison to a threshold. And accuracy value is calculated by subtracting the *B* value from 100. δ_d value is selected according to the histogram of the image.

$$B = \frac{1}{N} \sum_{(x,y)} (|d_C(x,y) - d_T(x,y)| > \delta_d)$$

Equation 2 Where d_C is the disparity map, d_T is the ground truth and δ_d is the disparity error tolerance.

Then, the accuracy value is calculated by subtracting the *B* value from 100. δ_d value is selected according to the histogram of the images. Its value for four images is given in Table 1.

Images	Tsukuba	Cones	Venus
$\boldsymbol{\delta_d}$	70	33	30

Table 1 Thresholds for the images

The histograms show pixel counts versus absolute difference matrixes which are calculated by subtracting from the output disparity map of the accelerator to its ground-truth image and then taking absolute values of each value. It shows how the output image is different

from the ground-truth image. It is expected that the difference will be small. Figure 9, Figure 10, and Figure 11 illustrate the histograms of the images Tsukuba, Cones, and Venus respectively along with different kernel configurations shown in Figure 6.

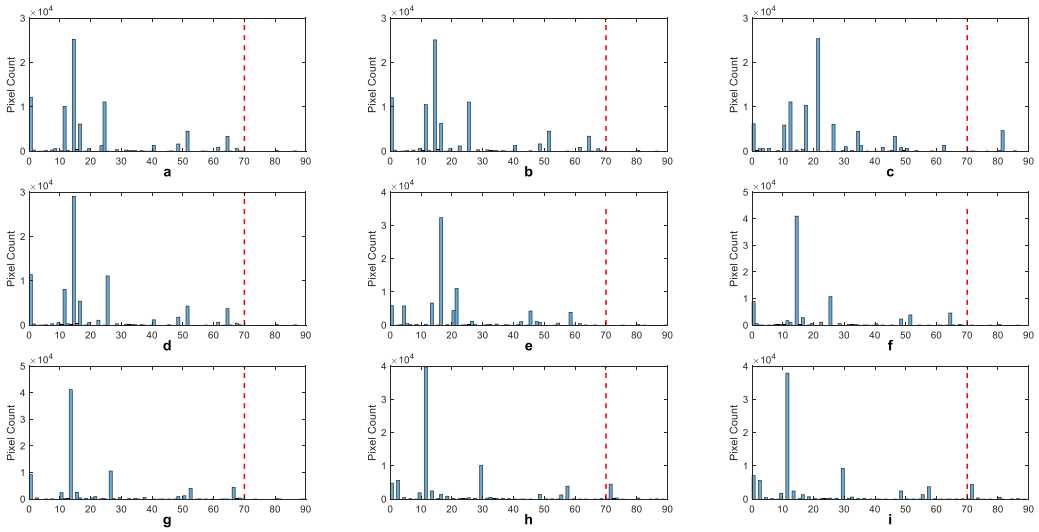

Figure 9 The histograms of the Tsukuba image for 7x7 window size implementation (threshold is 70, the red line): a) Uniform. b) Non-redundant. c) Full. d) 16-Point. e) 12-Point. f) 8-Point. g) 4-Point. h) 2-Point. i) 1-Point.

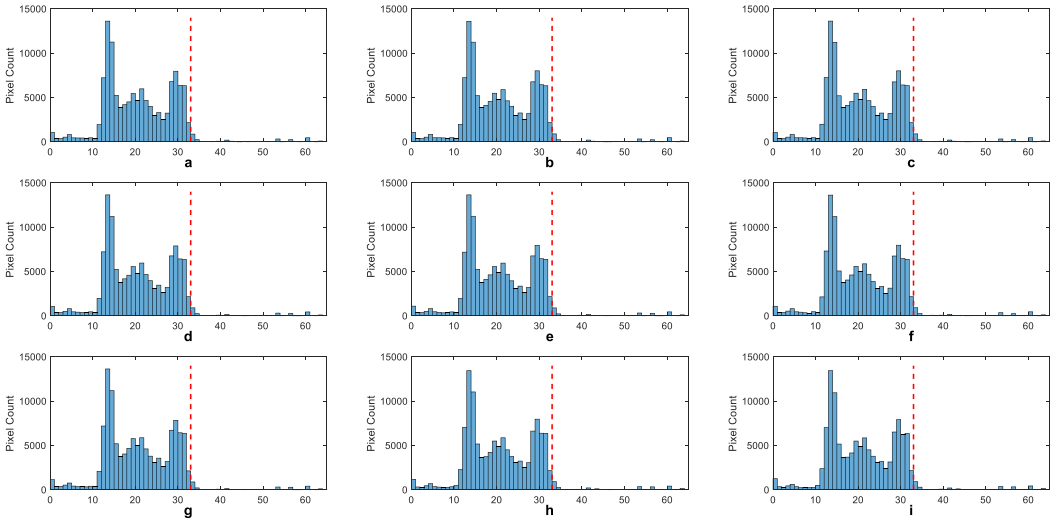

Figure 10 The histograms of the cones image for 7x7 window size implementation (threshold is 33, the red line): a) Uniform. b) Non-redundant. c) Full. d) 16-Point. e) 12-Point. f) 8-Point. g) 4-Point. h) 2-Point. i) 1-Point.

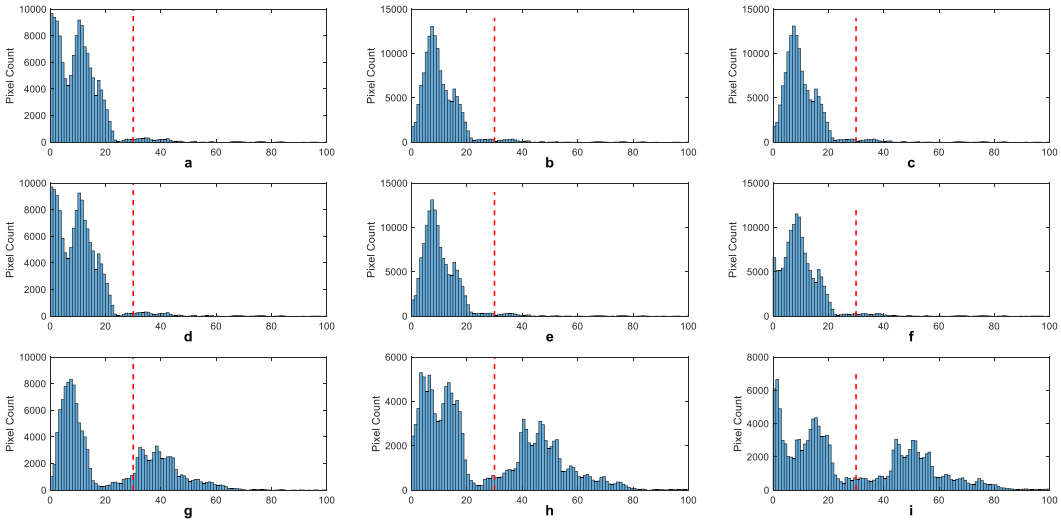

Figure 11 The histograms of the venus image for 7x7 window size implementation (threshold is 30, the red line): a) Uniform. b) Non-redundant. c) Full. d) 16-Point. e) 12-Point. f) 8-Point. g) 4-Point. h) 2-Point. i) 1-Point.

Peak Signal-to-Noise Ratio (PSNR) is the second metric for the evaluation. It measures the relationship between the noise (or error) in an image and the original image's signal. A high PSNR indicates that the difference between the original image and the target (or predicted) image is small, meaning the image quality is better. It is calculated by comparing the maximum pixel value with the mean square error (MSE) of the image as given in Equation 3.

$$PSNR = 10 \times log_{10}(\frac{peak\_value^2}{MSE})$$

Equation 3 PSNR calculation

The third metric is the signal-to-noise ratio (SNR) which evaluates the quality of a signal in the presence of noise. Higher SNR values typically indicate better image quality. Equation 4 shows the calculation.

$$SNR = 10 \times log_{10} \frac{P_{signal}}{P_{noise}}$$

Equation 4 SNR calculation

2.6. The Field Programmable Gate Array (FPGA)

The FPGA is an integrated circuit that can be repeatedly reprogrammed. It has been extensively utilized for ASIC prototyping, emulation, and high-performance computing [15]. To apply the fault injection with an emulation-based strategy, a hyper FPGA is used.

A framework is provided to connect FPGA online. To interface with FPGA and make it more general to use it, ComBlock is introduced. The entire interface management is carried out through this block. Therefore, custom designs are connected to ComBlock and data is sent by CPU.

2.6.1. The Hyper FPGA

The FPGA used for the project is Trenz SOM TE0803-03-4BE11-A which features from the AMD/Xilinx Zynq™ UltraScale+™ XCZU4EG-1SFVC784E. The board support package is available [16]. It is used in the creation of the Vivado project.

The term "Hyper FPGA" typically refers to an advanced FPGA platform or technology that provides high performance, versatility, and scalability beyond conventional FPGA designs. The specifications are :

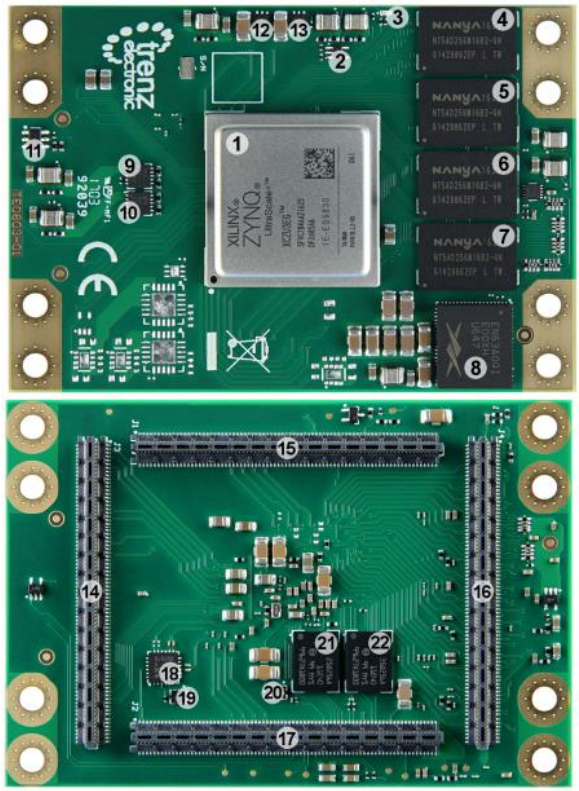

Figure 12 TE0803-01 MPSoC module

- Xilinx ZYNQ UltraScale+ MPSoC, U1
- 2-Input AND Gate, U39
- Red LED (DONE), D1
- 256Mx16 DDR4-2400 SDRAM, U12
- 256Mx16 DDR4-2400 SDRAM, U9
- 256Mx16 DDR4-2400 SDRAM, U2
- 256Mx16 DDR4-2400 SDRAM, U3
- 12A PowerSoC DC-DC converter, U4
- 1.5A LDO DC-DC converter, U10
- 1.5A LDO DC-DC converter, U8
- Voltage monitor circuit, U41
- 0.35A LDO DC-DC converter, U26
- 0.35A LDO DC-DC converter, U27
- Ultra fine 0.50 mm pitch, Razor Beam™ LP Slim Terminal Strip with 160 contacts, J3
- Ultra fine 0.50 mm pitch, Razor Beam™ LP Slim Terminal Strip with 160 contacts, J1
- Ultra fine 0.50 mm pitch, Razor Beam™ LP Slim Terminal Strip with 160 contacts, J4
- Ultra fine 0.50 mm pitch, Razor Beam™ LP Slim Terminal Strip with 160 contacts, J2
- 4-channel programmable PLL clock generator, U5
- Low-power programmable oscillator @ 25.000000 MHz, U5
- Low-power programmable oscillator @ 33.333333 MHz (PS_CLK), U32
- 256 Mbit serial NOR Flash memory, U7
- 256 Mbit serial NOR Flash memory, U17

2.6.2. The ComBlock [17]

The Communication Block IP Core (Core ComBlock) is the result of a collaboration between the Multidisciplinary LABoratory (MLAB) from The Abdus Salam International Centre for Theoretical Physics (ICTP, Italy) and the FPGA division of Micro and Nanotechnology from the National Institute of Industrial Technology (INTI, Argentina). It is licensed under the BSD 3-clause.

MLAB projects are characterized by solving high-speed acquisitions and processing in the FPGA and sending the resulting data to a PC. The Processor included in devices such as Zynq is mainly considered a provider of data storage (DDR memory) and Ethernet connections. The ComBlock was created to provide known interfaces (registers, RAM, and FIFOs) to a user

of the Programmable Logic (PL), avoiding the complexity of the bus provided by the Processor System (PS), which is AXI in the case of the Zynq-7000.

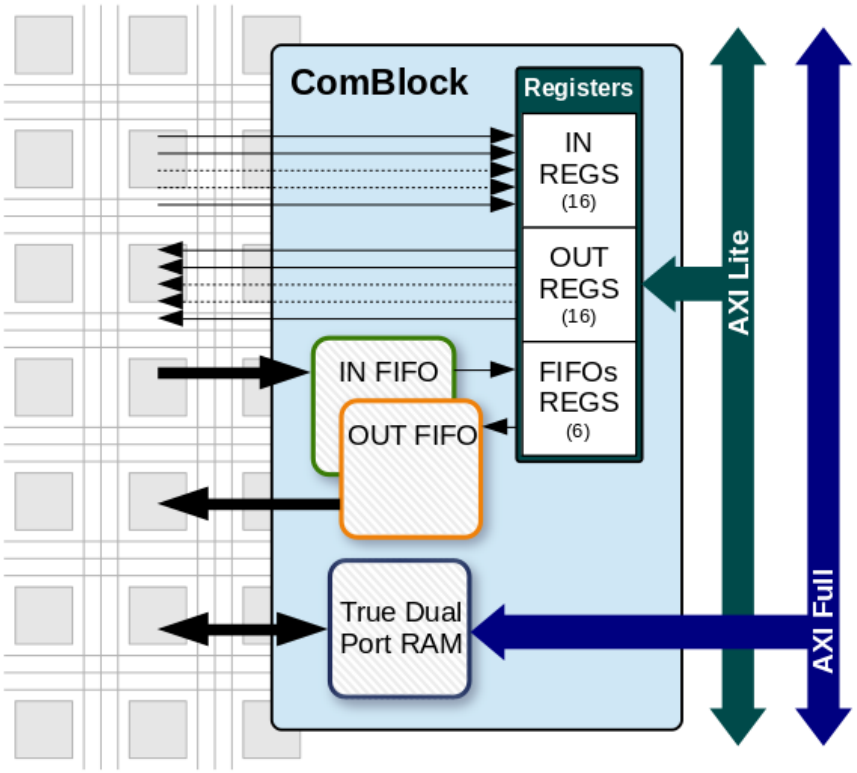

Figure 13 ComBlock

The block provides:

- Up to 16 input and/or output registers (configurable from 1 to 32 bits).
- A True Dual-Port RAM offers a straightforward RAM interface for user interaction. It allows customization of parameters such as data width, address width, and memory depth based on specific requirements.
- Two asynchronous FIFOs, one from PL to PS and another from PS to PL, with indications of empty/full, almost empty/full, and underflow/overflow conditions. Their inclusion, the data width and the memory depth can be configured.

As it is illustrated in Figure 14, there are 5 interfaces for the user on the FPGA side:

- IN_REGS: input registers.
- OUT_REGS: output registers.
- IO_DRAM: input/output True Dual Port RAM.
- IN_FIFO: input FIFO.

- OUT_FIFO: output FIFO.

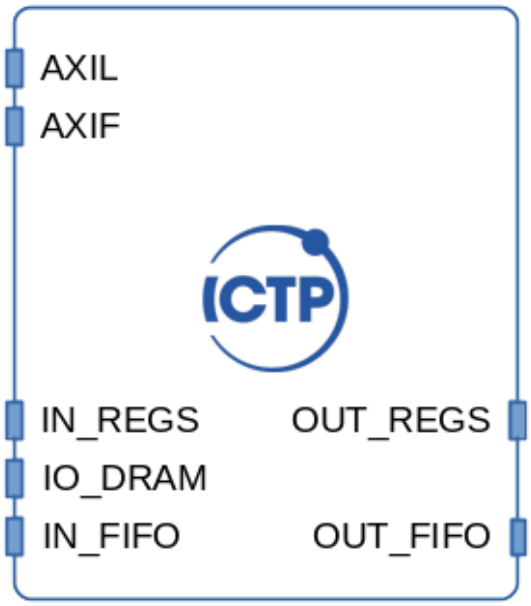

Figure 14 ComBlock with ports

In the case of the Vivado version, it provides 2 interfaces for control on the Processor side:

- AXIL: AXI4 Lite with the registers and FIFOs.
- AXIF: AXI4 Full with the RAM.

Chapter 3

3. Stereo Matching Accelerator [1]

The target stereo-core accelerator evaluated in this thesis is based on census transform and the sum of hamming distance. It measures the depth of the object in the image and creates a disparity map accordingly. Understanding the architecture of the hardware is needed for intuitive testing and also for analyzing the most sensitive part of the circuit.

The hardware consists of three main sub-modules and each one will be explained in the following chapters.

3.1. Census Transform Module

The census transform can be thought of as two steps. The generation of the census transform vector is the first step followed by the calculation of the hamming distances between the vectors. The census transform module handles the first step. The kernel configurations of the transform can be implemented with configurations as given in 2.1.2.

The census module's ports are shown in Figure 15. The *i_data* is the 8-bit pixel value of the image and *i_dval* is its validation signal. The census vectors are generated by first buffering the pixels and then transforming them. Notice that there are two output vectors (*o_data_L* and *o_data_R*) due to that design choice. In Figure 16, the upper and lower windows can be seen. The validation signal for both outputs is the *o_dval* signal. Lastly, there are two images of the same scene for disparity map generation and stereo image processing. Therefore, two census transform modules are implemented.

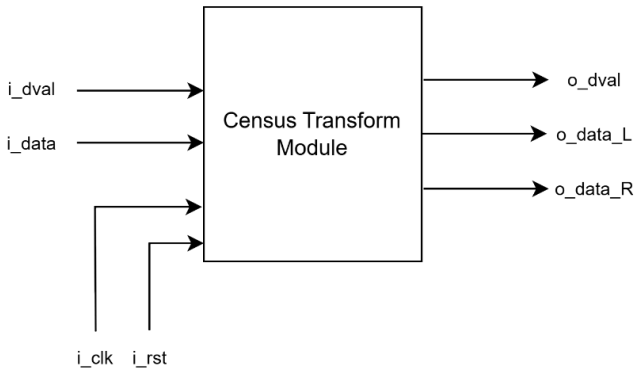

Figure 15 Ports of Census Transform Module

The census module employs a computational approach called stream processing, which implements a row buffer architecture used for storing only the necessary image rows to compute the kernel size of the census transform. This requires $W_c xM$ storage elements where M is the length of the image and W_c is the kernel size. After the census transformation, the next step consists of the implementation of the Hamming distance calculation over a correlation window W_h. To avoid implementing a second-row buffer, and in turn, saving memory resources, this stage is implemented by increasing the size of the census row buffer to match the correlation window of the sum of hamming' distances. Therefore we need to store an extra W_c line of the image. Finally, the size will be $(W_h + W_c) \times M$. This is illustrated in Figure 16.

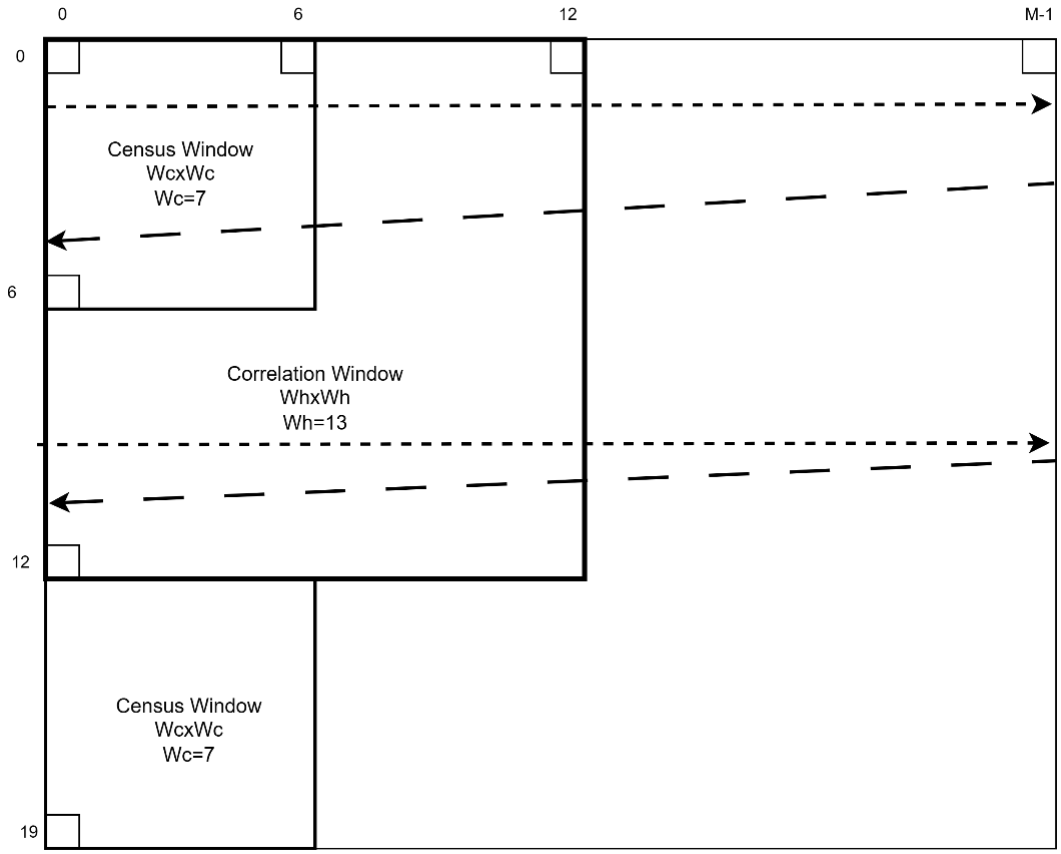

Figure 16 Window buffer for 7x7 census window and 13x13 correlation window

Output *o_data_L* comes from the upper census window and *o_data_H* comes from the below census window. Window buffer acts like shift registers which forwards it to the register next to itself. When it reaches the end of the current line, the last pixel is forwarded to the first register of the next line.

Finally, the *o_dval* signal will be given when the center pixel value of the upper Census Window arrives and then the comparison can start. In the case of $7x7$ census window with $M = 384$, for instance, the time required is $384 \times 3 + 4 = 1156$ clock cycle.

3.2. Sum of Hamming Distance (SHD) Module

The SHD module is the most complex module in the accelerator and consists of three different types of modules. The first is called the *Num_of_Ones* module which calculates the hamming distance for outputs of census transforms. This is represented as an HD module in Figure 17. The other two modules reside inside the SHD module in the same figure. Names of the these modules are *Window_Sum* and *Disp_Cmp*. *Window_Sum* calculates correlation values between pixels in left and right images. Then, the *Disp_Cmp* module selects the best disparity level by checking correlation values coming from the *Window_sum* module.

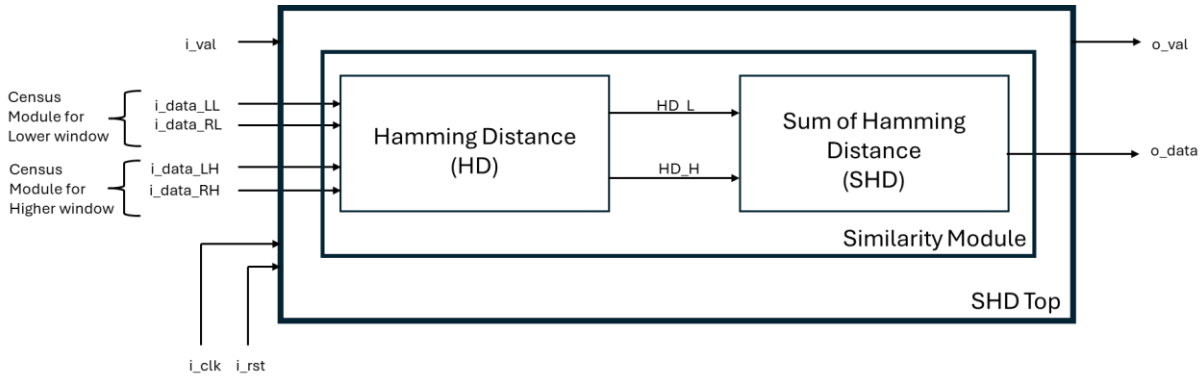

Figure 17 SHD Module with inputs/outputs

Number of the modules depends on the designer's choice which is based on the experiments. Our design is based on the disparity level of 64. Therefore, we will present the whole design based on that number. Figure 8 can be seen as a disparity level of 1. However, there will be no need for a comparator for that case. The reader has to also know that the stereo core used a left-to-right consistency check (LRCC). This also affects a number of comparator modules. After this brief introduction, we give details of each module. Finally, the whole core architecture is given in Figure 18.

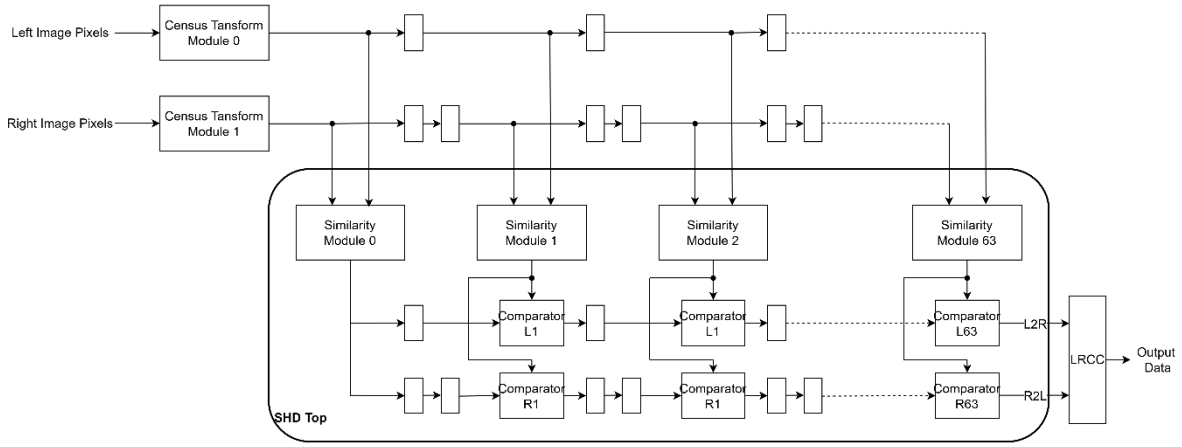

Figure 18 Stereo core architecture

3.2.1. Hamming Distance (HD): *Number_of_ones* module

This module takes the output of the census to transform for left and right images and then calculates the hamming distance. *XOR* operation results in finding the difference and then the rest of the circuit counts the number of 1s. For instance, if the left input is "1001" and the right input is "1111". After XOR operation output will be "0110". Then the circuit will calculate "10" which is 2 as integer values.

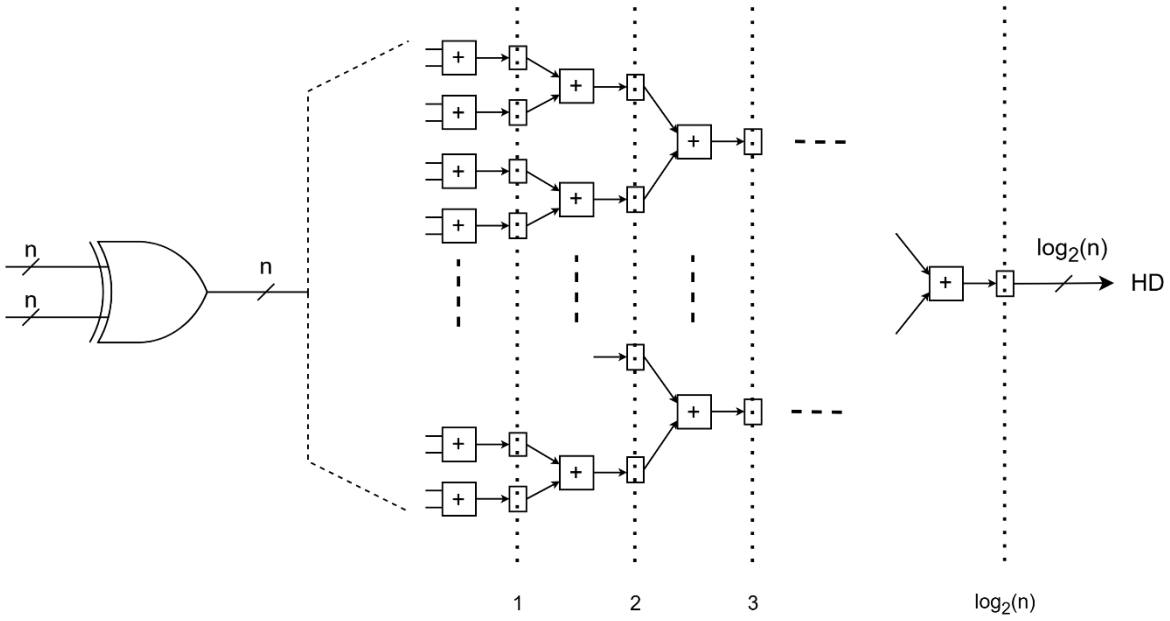

Figure 19 Calculation of hamming distance

3.2.2. Sum of Hamming Distance (SHD): *Window_Sum* module

As mentioned previously, this module is optimized to reuse some computation for each similarity measure evaluation [1]. For example, considering the similarity measure for a pixel in the left image at x_l and in the right image at x_r, both on the same pixel row, calculation is done as follows: if we want to compute similarity for $x_l + 1$ and $x_r + 1$ for window size W_h, $W_h - 1$ similarity computation will be exactly the same as for previously calculated similarity measure computation. Then, there is only one new computation is needed. Therefore, the new column reused $W_h - 1$ pixels from the same column of the previous row. The computation is illustrated in Figure 20.

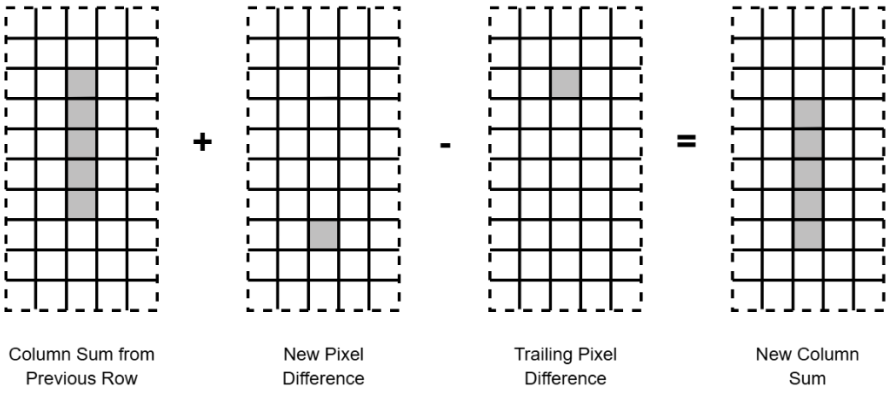

Figure 20 Computing Column Sum [1]

This is valid for the row computation. The new row reuses the previous $W_h - 1$ pixels in the same row. Figure 21 shows how the window summing optimization work. A row buffer with length of image and a column buffer with W_h length are required to obtain the trailing column sum.

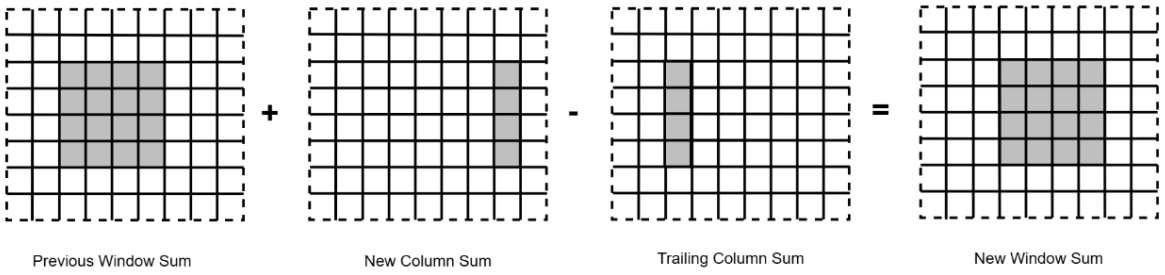

Figure 21 Computing Window Sums [1]

Discussion about such optimization is out of topic. The only thing we consider here is the architecture targeted by fault injection. After introducing the buffers, the general architecture of this module is shown in Figure 22. The output of the module goes to the comparator for the selection of disparity level. The reader has to be aware of the addition and substruction circuits since the sign bits involve the operation and stuck-at faults can dramatically affect the outcome of this module. This will be presented later. Finally, the number of modules is 64 as shown in Figure 18. Designers may select different disparity levels and the number of that module depends on that number.

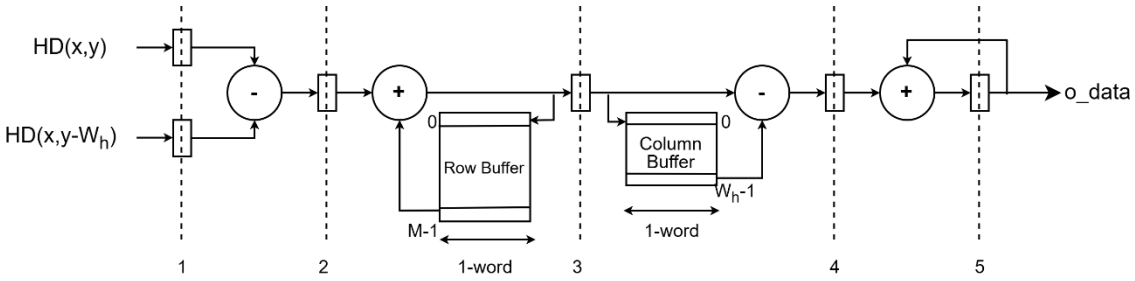

Figure 22 The architecture of the SHD module [1]

3.2.3. Comparator: *Disp_Cmp* module

The comparator module consists of a 2-2x1 multiplexer and allows to ranking of the disparity correlation among two different disparity levels and then selecting the one with the lowest rank. It takes *o_data* from the *Window_Sum* module which is the correlation value. There are two comparators for each *Window_Sum* module except for the first one. One output is for the disparity value and the other is for the selected correlation value. According to *o_data*, it selects the disparity level and that value propagates through the end.

To better understand, we analyze Figure 18 using *Comparator* L_1. The inputs in Figure 23 are the following: C_1 and C_2 come from *Window_Sum* module. C_1 comes from the first similarity module-0 and C_2 comes from similarity module-1.

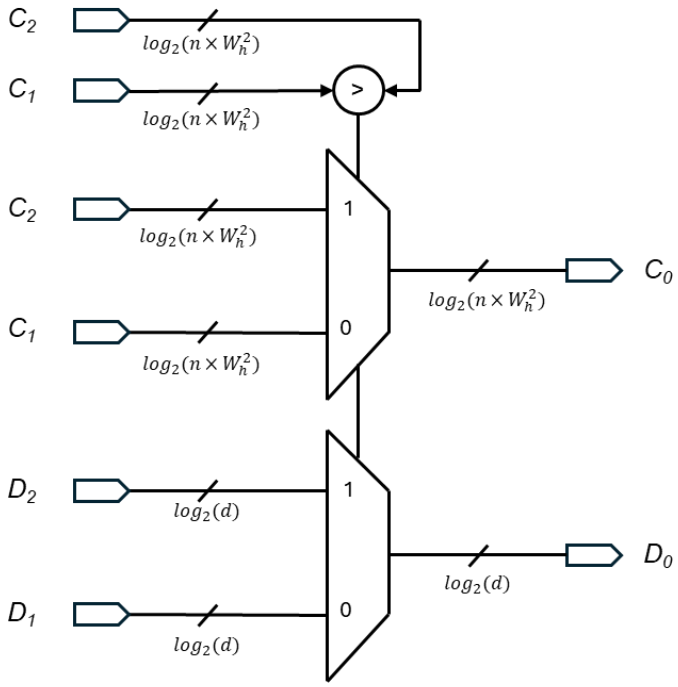

Figure 23 Comparator Module. n is a number of bits as the output of the census transform. W_h is the size of the correlation window

D_1 and D_2 represent disparity levels. D_2 depends on the level and it is 0 for *Comparator* L_1. For instance, it is 34 for *Comparator* L_{34}. On the other hand, D_1 is the previous output from a comparator. Its first value is 0.

Assume C_2 is 17 and C_1 is 10. This means that at the current level, the correlation is less than the previous one. It means that a pixel on the left image (x,y) has less correlation than a shifted pixel on the right image (x,y+z). Therefore, the output will be C_1. When the correlation is less than the previous one, disparity value D_1 will be selected.

In summary, the best match from one pixel in the left image with D pixels in the right image is the one that has the lower correlation value. The module selects the lowest correlation value and disparity value related to that correlation value. As we said, there are two of them for the same level. The second one is used for left-to-right consistency checks to be sure that the selection is correct. It is used to double-check.

3.3. Left-Right Consistency Check (LRCC) Module

The left-right consistency check is one of the most common port-processing steps used in the literature. This technique is very successful in eliminating false matches. To find the best match in the left image, a pixel in the left image is compared to the d (disparity) previous pixels in the

right image. Likewise, for the best match in the right image, the comparison is done in the left image. As it turns out, there is no need for a reference image for the implementation. Thanks to the LRCC module, similarity measures are computed for both images. In Figure 18, inputs of the LRCC module are *L2R* and *R2L* which correspond to “match for left reference image” and “match for right reference image” respectively. This structure is a linear search structure.

The module consists of two multiplexers, substruction, absolute, and less-than operators. Larger multiplexer is $64x1$ type and inputs come from *R2L* input. These are shifted through the bottom. After taking the absolute difference between pixel disparity values, that value is compared with a threshold. The threshold value can be selected arbitrarily during the operation of the accelerator. A small threshold value will perform strict LRCC while a larger threshold make this refinement more relaxed.. If the absolute difference is less than the threshold, we can say that this L2R input is a correct output disparity. Figure 24 shows the internal architecture of the LRCC module.

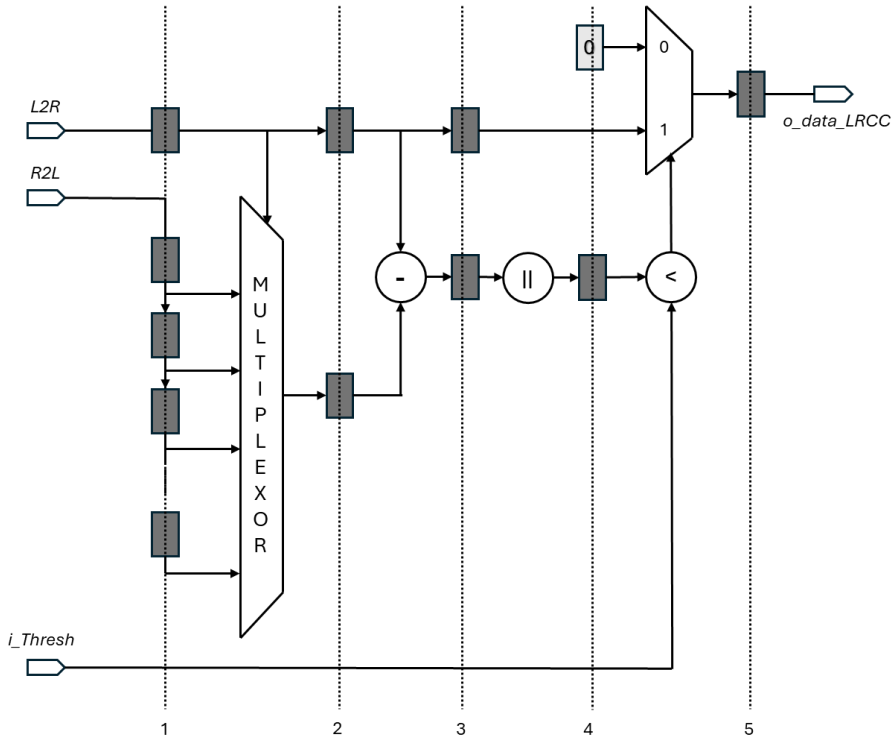

Figure 24 LRCC module

Chapter 4

4. Simulation Environment for Reliability Assessment

Simulations are executed in two ways. The first one is fault-free and the other is with fault injection. The fault-free simulations are normal simulations that require a testbench with input images then results in the creation of a disparity map for that input. The faulty simulations require a fault list that specifies where to inject stuck-at-fault. The output of such simulations is reported with statistics including accuracy, PSNR, and SNR. The following chapters will describe the simulations in detail.

4.1. Stereo Image Dataset

A stereo image pair is used for a stereo correspondence algorithm as an input to generate a disparity map. In computer vision, estimating a disparity map is a challenging task. This map can be converted into a map of distance. As a result, the words range map and depth map are used in the literature in addition to the disparity map.

Our design applies such a stereo correspondence algorithm and uses Middlebury's stereo benchmark dataset. Since this dataset is widely used and known, we decided to use them as input to our design. There are four different image pairs: Tsukuba, Venus, Teddy, and Cones. Dimensions of the images are different. The ground-truth images are also provided in the dataset.

These images are generated using different methods according to [18]. The Tsukuba image pair has a disparity range of 16 pixels, and its ground-truth image excludes a border of 18 pixels. Excluding the border of 18 pixels is applied to other disparity maps while calculating some metrics. The Venus image has a disparity range of 20 pixels. Lastly, The Cones image has a disparity of 60 pixels. These are shown in Figure 25. A structured light technique was used to generate their disparity ground-truth data. These methods involve projecting specific light patterns onto a scene to directly capture a range map of the scene.

It can be observed from the disparity maps that objects closer to the observer have higher gray level values (brighter regions). The depth information can be extracted from the disparity map in that way.

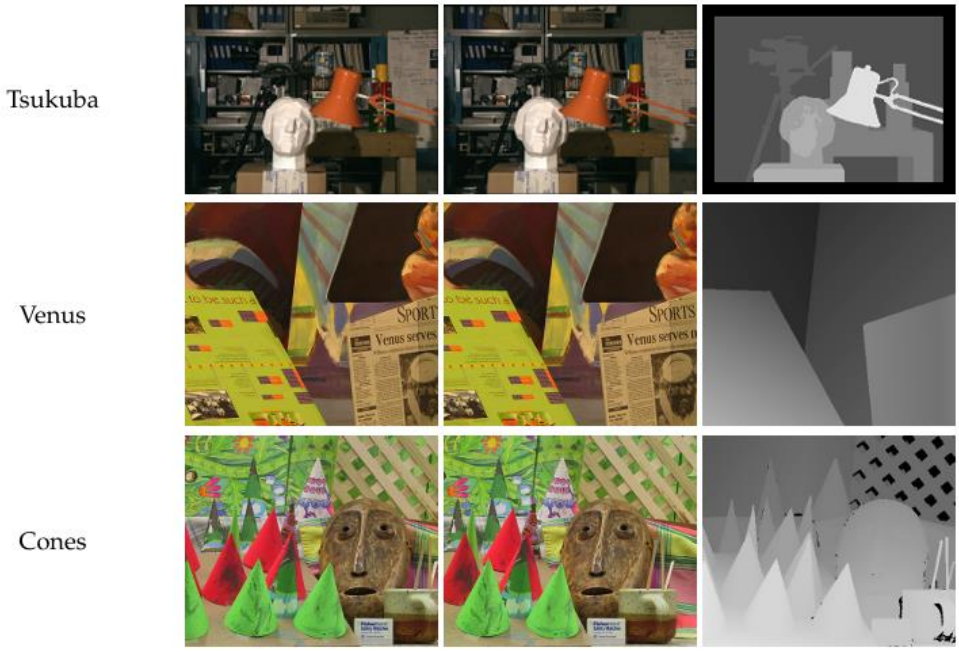

Figure 25 Middlebury Stereo Vision dataset: left images, right images, and ground-truth

4.2. Fault-Free Simulations

The testbench of the fault-free simulation is shown in Figure 26. As usual, left and right images are provided as input. These are converted to txt files and the testbench sends each pixel value with validation bits to the stereo core. Then, output disparity values with validation bits are saved in output txt files. With the help of Python script, a disparity map is generated.

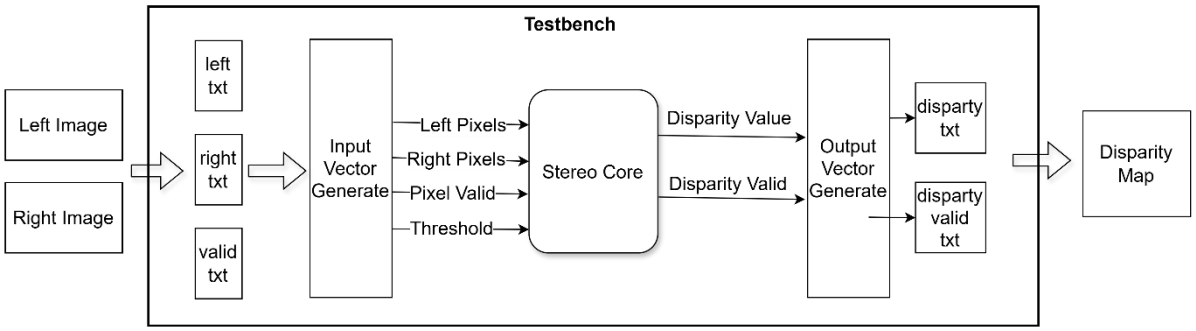

Figure 26 Testbench environment for fault-free simulations

The Python scripts enable the whole process with a single command. The script reads the input images with specified names, disparity level, kernel window size W_c, correlation window size W_h, and number of bits to represent a pixel, then compiles the design. After successful compilation, the simulation is started. As indicated before, output text files are converted to disparity maps. The elapsed time for one simulation depends on the input image size. However it does not take more than 10 minutes for our images. Window sizes also affect the elapsed time. For instance, with lower W_c, the simulation will take less time than higher W_c.

Finally, disparity maps are generated by the script with normalization of the pixels between 0 to 255. Minimum and maximum values are detected from the output and each pixel value are normalized with the following formula.

$$\frac{Pixel - Pixel_{min}}{Pixel_{max} - Pixel_{min}} \times 255$$

Equation 5 Image normalization

4.3. Faulty Simulations

The testbench of the faulty simulation environment is shown in Figure 27. The faulty simulation depends on the fault list which indicates locations where the fault will be injected in the stereo core. The fault type is a stuck-at fault. The output disparity map is compared to the ground truth of the input image then a script calculates statistics with the metrics. All statistics are written in a CSV file. The simulation continues until all faults are injected separately. When a simulation with a fault injected ends, the environment resets the fault and then continues with another fault in the list.

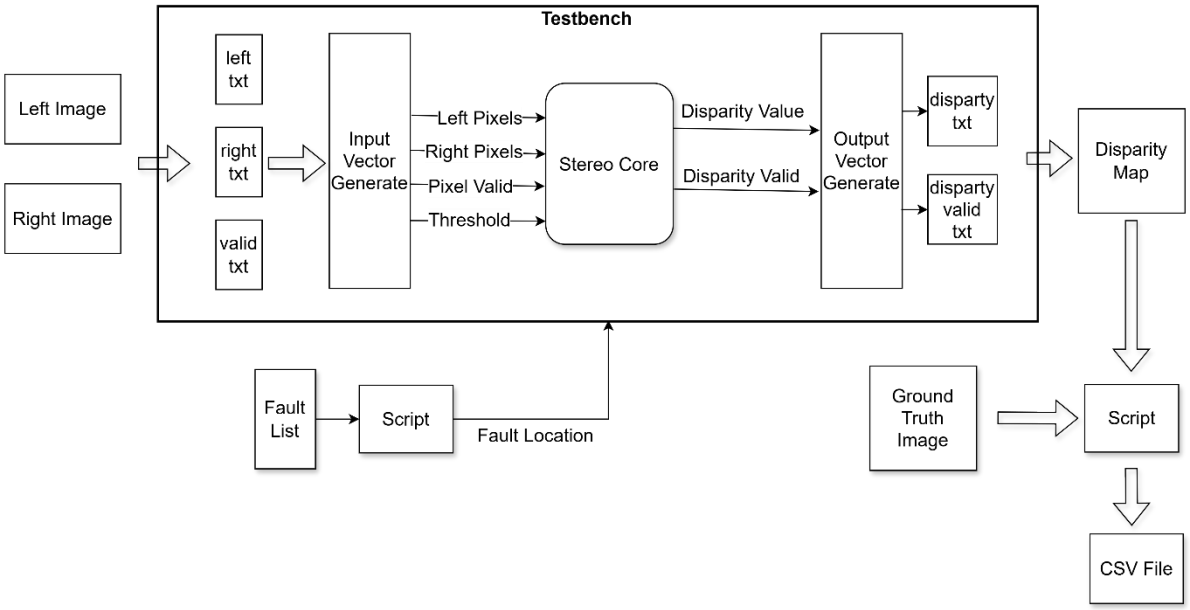

Figure 27 Testbench environment for faulty simulation

The format of the fault list follows the structure of "sa testability fault_site". "sa" is a type of stuck-at fault which is either "sa0" or "sa1". "testability" is always "NC" and "fault_site" shows where the location of the fault Figure 28 is an example. The "fault_site" is a path of the bit defined by the simulator which is QuestaSim in our case. The fault list consists of the bit location with sa0 and sa1 in the list to compare the effect of different faults on the same bit. Output CSV file is in the format of "fault_site,sa,(accuracy, PSNR, SNR)". This is generated for all images for $7x7$ window configurations.

```
sa0  NC  stereo_match_tb/UUT/Census_Left/o_data_H[0]
```

Figure 28 Example index for fault list

In conclusion, the environment reads the fault list line by line. Before starting the simulation, it freezes the specified bit in the architecture and start the simulation. When output is generated, it calculates the statistics and resets itself for another fault injection. This is a automitised environement. When a user runs the script, the simulations run till the end of the fault list.

4.3.1. Fault List: Where to Inject the Fault

The question is where to inject the fault. There are millions of bit positions and it is not possible to target all bits with simulation-based techniques due to a timing issue. So, the selection of the bit locations depends on a knowledge of the internal architecture of the stereo vision accelerator.

Let's consider the census transform module that only compares two values. An input pixel is stored in the first register in the slide window and it is shifted to the end while new input pixels arrive. Any fault in the slide window will be propagated toward the last register in the slide window. It means that all pixels will be exposed to the same fault. Since the comparisons involve the unsigned values, the effect of the fault may not be significant. After census vectors are generated, the number of *1*'s is counted by the *num_of_ones* module in Figure 19. An assumption is that having a one-count difference will not also make a significant impact.

The faults in these modules cannot make a significant impact. On the other hand, the calculation of the hamming distance involves addition and subtraction. Since the outcome of the subtraction may be negative, any corruption in the sign bit can cause an obvious error. Even if each bit of subtraction, assuming the input port of the subtraction module, is corrupted, the error can be observable in the output image because it may lead to a wrong disparity selection. Likewise, any error in the disparity selection, by *Disp_Cmp*, may also lead to observable incorrectness. This is the case also for the *LRCC* module where a comparison exists.

A final comment on this is that the upper bits, above half, of a signal are the most critical bits if a comparison, addition, or subtraction operation is being performed. The *Disp_Cmp* module is targeted to the fault injection. The outputs of this module are injected with the stuck-at faults. The fault list is created for a uniform design with a 7x7 window size. There are 63 *Disp_Cmp* modules and each has outputs *o_data_C* with 12 bits and *o_data_D* with 7 bits. Considering stuck-at-0 and stuck-at-1 fault types, there are a total of 2394-bit locations.

Chapter 5

5. Emulation-Based Strategy

Simulation is the easiest way to know if our circuit works. It does not need any hardware and only software tools can be handled. Aims is to verify the functionality before embedding it into hardware. It helps to improve and make modifications before manufacturing and reduce costs if verification is done properly. One throwback of the simulation environment is time consumption. Simulations take days and it can be time-consuming. This is the case for the simulations of the stereo core. This chapter presents an emulation-based strategy instead of a simulation-based strategy.

The stereo core is a huge hardware which makes it impossible to target all bits for the fault model. Some sophisticated tools like TetraMax are good but the cost of such tools is very high. Another approach to applying the fault model to our circuit is to embed it into FPGA. This requires some modifications in our circuit. However, the modification can be handled easily and these do not require much hardware resources. Migration from the simulation environment to the FPGA world with a fully automized framework helps us to fasten fault injection operations.

5.1. Methodology

The framework is organized to fully automate fault injection campaigns. The aim is to select fault target and fault type from the software side and control the whole operation. All simulations run on the FPGA fabric and enable us to save time. The proposed framework is demonstrated in Figure 29.

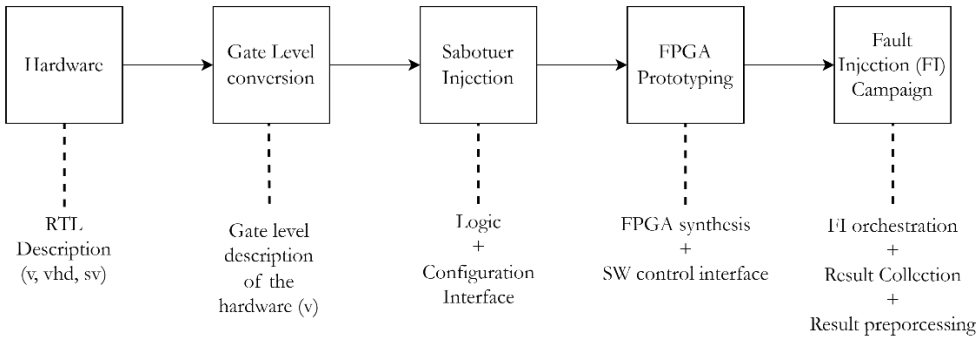

Figure 29 Proposed Framework

The diagram outlines a step-by-step process for testing the fault tolerance of a hardware design using FPGA-based fault injection techniques. The process begins with the Hardware, which serves as the foundation for the system under analysis. In the next step, Gate Level Conversion is performed to translate the design into a low-level gate representation suitable for fault modeling and testing. Subsequently, a Saboteur Injection step is introduced, where fault models (referred to as saboteurs) are incorporated into the system to simulate potential hardware errors. The design is then implemented on an FPGA in the FPGA Prototyping phase, where the prototype is used for real-time testing. Finally, in the Fault Injection (FI) Campaign, systematic fault injection tests are conducted to evaluate the design's robustness and reliability. This workflow is commonly applied in electronics engineering to ensure the fault resilience of critical hardware systems.

5.2. Block Diagram

The idea is to integrate an accelerator in FPGA with proper modifications. As it is seen in Figure 30, the accelerator is inserted with saboteurs and shift register. Shift register store enables bits for each SS and enable bits for individual bits inside the SS. Super sabotuer (SS) is explained following sections. In our design, comparator module is targeted however, sabotuer injection can be applied any component inside the accelerator. Comparator is targeted because it is easy to test and apply the idea.

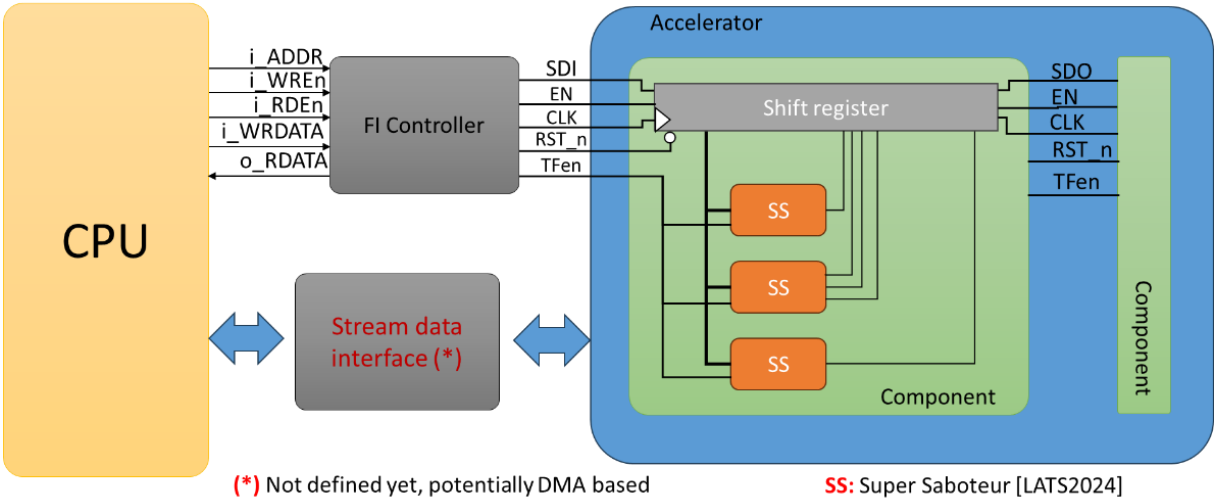

Figure 30 Block Design

The diagram illustrates a system architecture designed for fault injection and data processing. At the core of the system is the CPU, which acts as the primary control unit,

managing the overall operations and communication with other components. The FI Controller facilitates fault injection by coordinating the necessary signals and data between the CPU and the Accelerator. A Stream Data Interface enables seamless data transfer between the CPU and the Accelerator, ensuring efficient communication. Image data will be sent through this. Within the Accelerator, a Shift Register plays a key role in distributing enable signals to the SS components. The Component block, which consists of several sub-modules labeled as SS, is a target for fault injection. There might be several components which is why the component has to shift data inside the shift register.

5.3. Hardware Modifications

This chapter describes modifications done to the hardware. The steps cover gate-level conversion and saboteurs injection. Then the controller in Figure 30 Block Design will be introduced.

5.3.1. Yosys [19]

Yosys is a framework for RTL synthesis and more. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains.

This framework is used to translate the VHD files into gate-level Verilog files. We need a gate-level description to insert saboteurs between desired or all signals in the design. When the gate level design is ready, with the help of a script saboteurs will be injected automatically between each signal. Shift register will be also injected. With this modification, gate-level conversion is handled in Figure 29.

5.3.2. Sabotuers and Injection

When the gate-level netlist is ready, a script reads the netlist and inserts saboteurs and shift register. Before giving details of the hardware component, the script is introduced because it has 2 options. With this section saboteur injection part will be completed in Figure 29.

5.3.2.1. Python Script

This script aims to inject saboteurs for the stuck-at and transient fault model or to inject some logic to perform single-event upset (SEU) phenomena. To implement these, a target hardware needs some modifications. In both cases, a shift register has to be inserted in the target hardware for activation of the fault model for the corresponding signal. There are also control signals

which have to be added to port description to control shift register operation and fault models. Other modifications will be presented later.

The script takes filename and type of the operation (SABOTUER or SEU) and automatically updates the design for the operation. In addition to that, it creates a detailed report related to modification. Organization of the report will be described later.

To use the script, a Verilog description of the design has to be supported. If the programmer has a VHDL version of its design, the conversion from VHDL to Verilog is dealt with by Yosys. After that conversion verilog file contains the gate-level netlist of the design. On the other hand, if the Verilog file is already present, it has to represent a gate-level description. Since the gate-level description contains nets and connections, the script reads wires or registers and inserts saboteurs or some logic for each of them. The designer can run the script as follows:

```
python .\script_injection.py TYPE=SEU FILE=num_of_ones_7.v
```

After that, the script creates new .v file with modifications and a detailed report for that design. The report has following header: “SR Pos,Signal Name,Signal Type,Source Line Number,Source File Name,File Path”. The report is in csv format.

- SR Pos: The position of the enable bit for the corresponding signal.
- Signal Name: The name of the target signal.
- Signal Type: Type of the signal. It can be defined as either “wire” or “reg”.
- Source Line Number: The line number where the modified signal originates from.
- Source File Name: Gate-level netlist of the target design in Verilog format.
- File Path: It is the location where the report is located.

The designer can also know the length of the shift register from the report. In addition, the script also print some messages about the extracted signals with their number of bits.

5.3.2.2. Sabotuer Injection

The first case is saboteur injection. The script reads the Verilog design and finds every signal defined as wire with its number of bits except for inputs. Then it defines new temporary wires for each of them. The new name is created by adding "temp_" to the beginning of the original name. The aim is to insert a saboteur between these two signals. This is depicted in Figure 31. In the figure N bit signal is shown and it requires a super saboteur. In the case of a 1-bit signal,

a basic saboteur is enough to be inserted. In this case, the new wire name is “temp_signal” and the original wire name is “signal”.

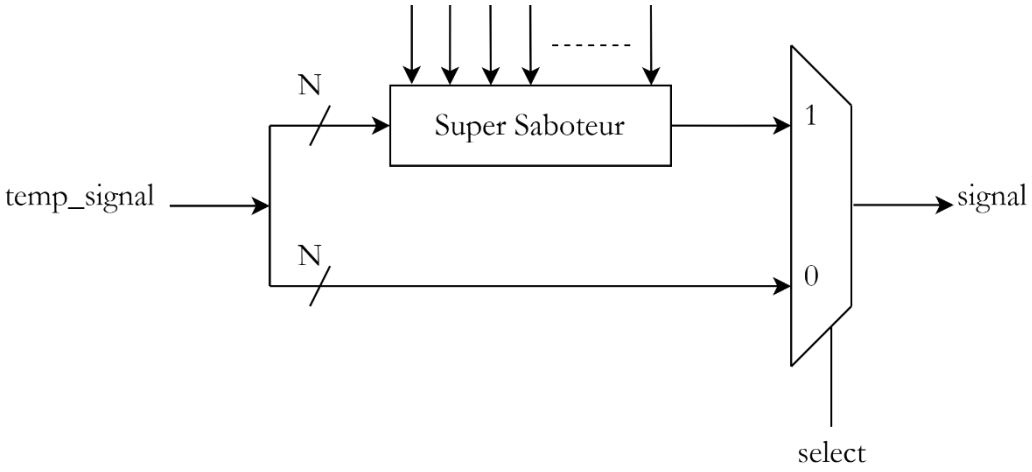

Figure 31 Sabotuer injection with super sabotuer [20]

There is another point that has to be modified by the script. This is an “assign” statement. Every signal assigned as an “*assign*” statement has to be replaced by its new definition. For instance, if signal is defined like “*assign _07_ = _05_ | _06_;*”, it will be transformed to “*assign temp__07_ = _05_ | _06_;*”. With this modification, saboteurs are inserted between every signal inside the design and can be targeted by fault injection. Lastly, the shift register has to be defined in the target design to enable signals in Figure 31.

Finally, some control signals are added to the port of the design. These signals are combined in 1 signal called “*i_FI_CONTROL_PORT[3:0]*”. It consists of *i_CLK_x*, *i_RST_x*, *i_TFEn*, and *i_EN_SR* signals in order from MSB to LSB. *i_CLK_x* and *i_RST_x* are global signals coming from the system. *i_TFEn* is enabled bit for fault injection and *i_EN_SR* is enabled bit for shift register. There is one more port with a 1-bit input called *i_SI*. It enables bits in Figure 31. Lastly one port is added for output signal called *o_SI*. The design will push *i_SI* bits to another module if exist.

The user can target any number of blocks in the top design. The block targeted for fault injection has to be modified as described above. Figure 32 gives an example connections with ports. Another question might be where the control signals coming from. The answer is a controller which will be described in another chapter.

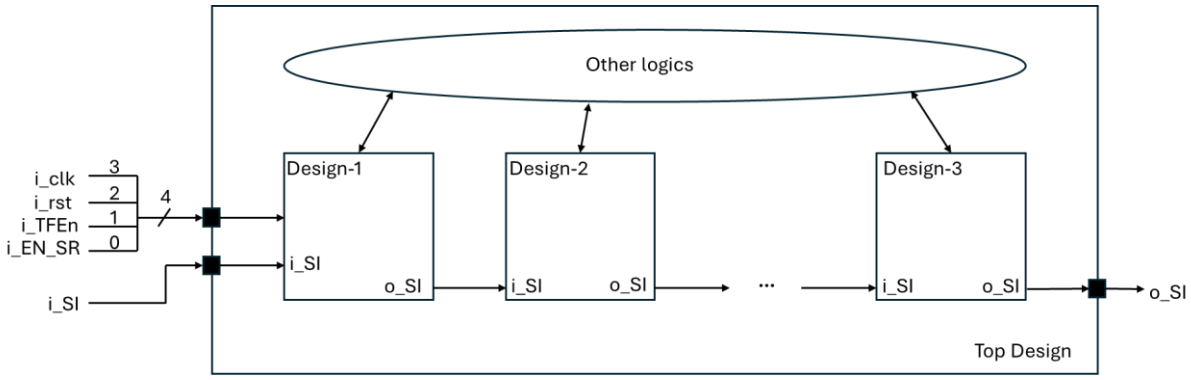

Figure 32 New ports

The reader may also ask why the script does not consider input signals. When Yosys generates a netlist it also defines wires for inputs. There might be two problems. The first one is defining a saboteur for such a signal. Figure 33 shows the case. The input signal can not be assigned in reality. This would not be a problem if the script is modified. However, the second problem may arise afterward. In the netlist, the input signal can be used in different places in the design and then all its usage has to be replaced by a new temporary signal. This can still handled by the script with proper modification however makes it much more complex. Therefore further modification is left for a newer version if it is needed.

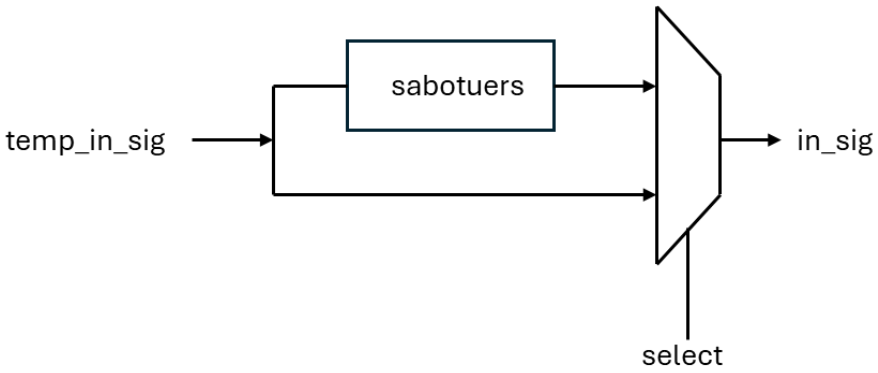

Figure 33 Saboteur for input signal

5.3.2.3. Single Event Upset (SEU)

The second case is for the SEU model. The script finds all signals for flip flops (defined as reg) and then for each signal and their enables, some logic is inserted. This is depicted in Figure 34. The logic for input D of flip flop contains enabled signal *TFEn* for fault injection framework with enabled signal *SR[x]* for target_signal. Then target_signal and the logic is XORed. The other logic for the *EN* signal includes also the same signal with the "*OR*" operation. The reason is to control the flip-flop even if *EN* is logic low. Finally, port modifications are the same as in

the first case. A difference from the first case is that *i_clk* and *i_rst* signals are not used in the second case.

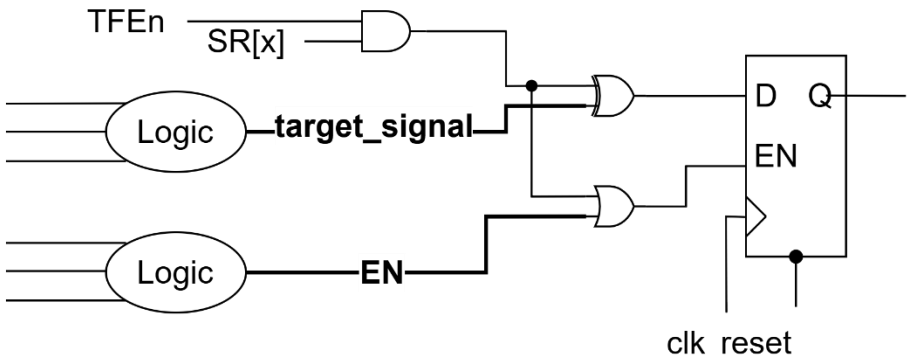

Figure 34 Logic for SEU

5.3.3. The FI Controller

The controller is hardware which sends the control signals to the target design for fault injection operation like stuck-at and transient faults. It receives corresponding signals from CPU and according to instructions by CPU, the controller prepares the target hardware. Basically, the controller is a bridge between CPU and the target block.

The controller has four internal registers: DATA, STATUS, COUNTER and COMPARATOR. CPU interacts with the controller by writing these register. Furthermore, CPU can also know the status of the controller by reading STATUS register. This is a register that both CPU and the controller modify and therefore they can communicate. In addition to these, controller takes clock and reset signals from the system.

Shift registers and saboteurs are inserted in a target design. The controller first sends the enable bits of the saboteurs to the shift register. This is the setup phase. When the CPU is sure about if the setup phase is completed, it sends the start command for the operation phase. The operation phase has two versions. The first version is for stuck-at faults. In that phase, saboteurs are enabled until the target hardware completes its operation. In the second version, the controller operates for transient faults. CPU sets corresponding registers for when to inject transient fault. When injection time is reached, the controller enables saboteurs only one clock cycle. Then it waits for other commands from the CPU. The controller modifies the STATUS register to let the CPU know that the operation is in progress for both cases.

By saying waiting, the controller has no idea when the operation of the target harware finishes. This information is known only by CPU. CPU has to send proper inputs on time to the controller. So, communication between CPU and the controller proceeds through STATUS register.

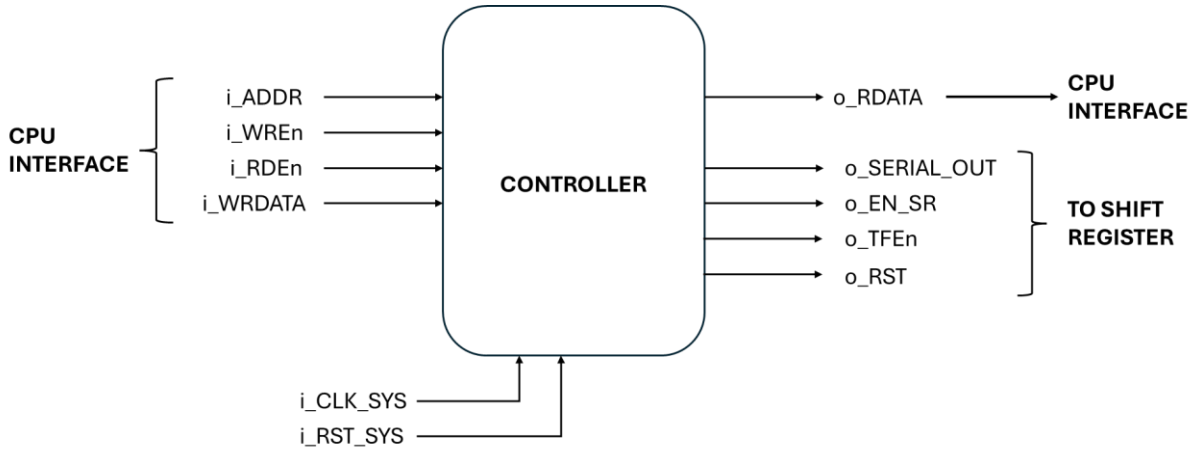

Figure 35 Controller signals

As indicated in Figure 35, there are four input signals controlled by the CPU. These signals are used to read and write to the registers. The only signal that can be read by the CPU is o_RDATA to know the phase of the controller. The rest of the outputs are connected to target hardware consisting of the shift register and saboteur.

- o_SERIAL_OUT: Enable signals for saboteurs to be sent bit by bit.
- o_EN_SR: Enable signal for the shift register. It is high while sending data via o_SERIAL_OUT.
- o_TFEn: Set enable signals for saboteurs. When low, the fault injection framework is out of use.
- o_RST: Used to reset the content of the shift register. It means disabling all the saboteurs. When the reset signal is sent by the CPU, it is forwarded to the target hardware.

To better understand the controller, the purpose of the register should be known. These registers are 32-bit registers and can be written or read with the help of two enable signals: WREn and RDEn. However, read and write operations cannot be executed simultaneously. CPU should wait for one of the operations to be done.

- DATA: It is a register that stores enable signals of the saboteurs. The content of the register is forwarded to the shift register inside the target hardware. If the length of the shift register is longer than 32, the CPU should refill the data register. For instance, if the length of the shift register is 45, the CPU should write corresponding 32 enable bits in the DATA register, and then it can write 13 enable bits. This register can only be modified by the CPU. The controller sends its content with the o_SERIAL_OUT output signal bit by bit.
- STATUS: This register can be modifed by both CPU and the controller. It stores information about type of fault [1:0] and legth of the shift register [19:4]. In addition, there are setup, start and busy bits. The controller can only modify busy bit to let CPU knows if it can write or read. Figure 36 describes this register.
    - Busy: This bit can be modified only by the controller. CPU can only read and decide the next operation. While the controller is in setup or operation state in FSM, this bit is set to 1. Then CPU can know not to modify any register inside the controller.
    - Op: Operation bit represents whether the simulation continues. After setup and start bit are set by CPU, the target hardware starts its simulation with or without fault. CPU can understand that the operation is in progress and does not try to write any register inside of the controller. If the CPU wants to start a new operation. It needs to reset the controller.
    - Shift Register Length: This lets the controller know how many bits are needed by saboteurs. The length determines the clock cycles elapsed during the setup phase. One should note that, the length does not include control bits which have to be sent to the shift register. Therefore, the actual clock cycle elapsed during the setup phase is the sum of shift register length plus 2.
    - Start: When the setup phase is completed, the CPU sets this bit to enable fault injection according to fault type or operation without fault.
    - Setup: CPU set this bit to start the setup phase. In this phase, the controller starts sending enable and control signals to the shift register inside the target block.
    - Control: Defines the type of faults. It is 2 bits.
        - 00: Stuck-at-0
        - 01: Stuck-at-1
        - 1x: Transient

The rest of the bits are not used. These bits can be used for the development of the controller.

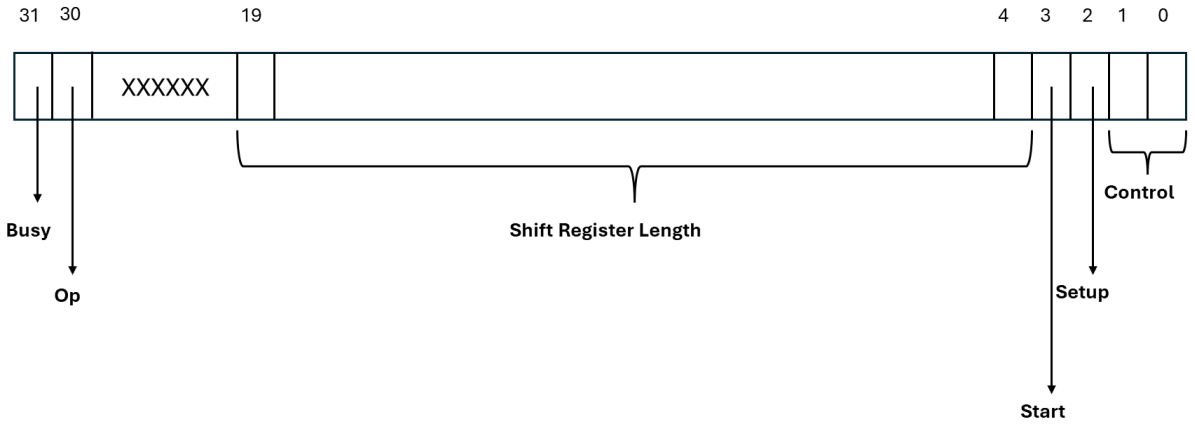

Figure 36 Status Register

- COUNTER: This stores a value of the normal counter and is used only for transient faults. It has to be reset by the CPU before transient fault injection starts. When the counter reaches a value which is transient fault injection time, the o_TFEn signal goes high in one clock cycle.
- COMPARATOR: This is set by the CPU to tell when to inject transient fault to the target hardware. When the value is matched with the counter value, the o_TFEn signal goes high in one clock cycle.

The internal architecture is described in Figure 37. When each register is written, the corresponding flag bit gets high. These flags are used in FSM. There are also internal counters for sending enable signals to the shift register. for instance CNT_SR counts till reaching length of the shift register. CNT_BIT counts up to 32 which is needed to consider if shift register length's is less than or equal to 32. On the other hand, CNT_CTRL counts up to 2 to send 2-bit control signals.

FSM controls setup, start, and progress operations. CPU has to first write in DATA and STATUS registers to initiate the setup phase. FSM executes S1 and S3 states respectively. End of this phase, the controller waits for the start bit in the STATUS register. Then output signals are given in the OPER state. The TF state is only for transient faults. When COUNTER and COMPARATOR values match, TFEn goes high 1 clock cycle. Otherwise, TFEn stays high at the end of the whole simulation for stuck-at faults.

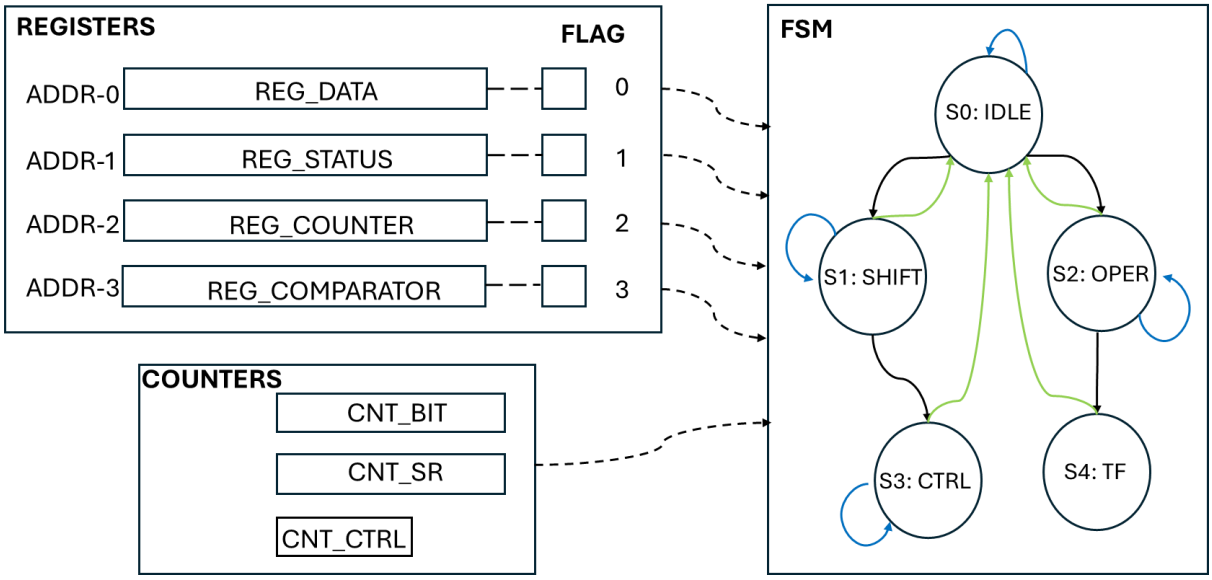

Figure 37 The internal architecture of the controller

Now, it would be better to give an example for the controller. Assuming that sa-1 fault will be injected and DATA register is set by 0xB and the length of the shift register is 91. Therefore, STATUS register content will be 0x5B5. Figure 38 represents the behavior for that configuration.

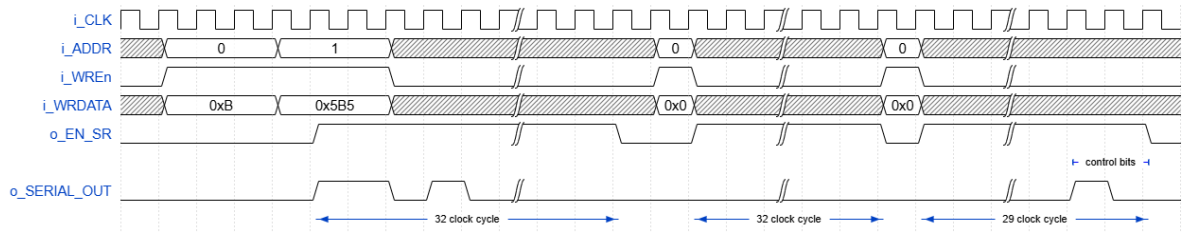

Figure 38 Timing diagram for stuck-at-fault

There are 91 enable signals and 2-bit control bits are also appended. There are a total of 93 bits and can be sent in 3 phases. The reason is that registers can hold 32 bits. CPU has to write zeros in the DATA register for the following phases. Then to start the operation, the CPU has to modify the STATUS register's bit-3 with high value. Finally, the o_TFEn signal goes high and fault injection will start. As mentioned "op" bit of the STATUS register will be 1 and modified by the controller to let the CPU know that the operation is in progress.

The previous timing diagram is the same for the sa-0 fault except for the control bits. In case of transient fault, counters are involved in the process. CPU puts a value on the COMPARATOR register. This is 16 in the following case. CPU has to write STATUS register for start before counting. This is illustrated in Figure 39.

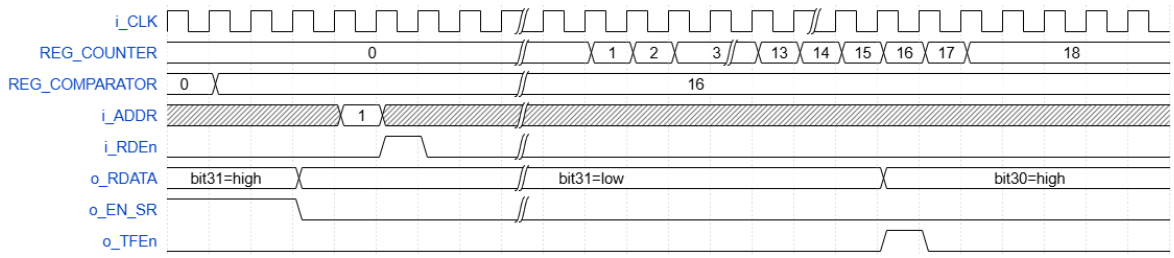

Figure 39 Timing diagram for transient fault

The values of the COUNTER and COMPARATOR register can be set at any time before deciding to start the operation. As it is seen, when the COUNTER value reaches to COMPARATOR value, o_TFEn goes high for one clock cycle. The programmer has to be careful about reseting the COUNTER register for another operation.

Chapter 6

6. Evaluation Results

This chapter presents the results of the simulations with accuracy, PSNR, and SNR values. Four images are used as a dataset as mentioned in Figure 25. There are 27 types of accelarator based on census transform. As mentioned, there are three window sizes and each have 9 kernels of census transform: 1-point, 2-point, 4-point, 8-point, 12-point, 16-point, non-redundant, full and uniform. As can be expected, area for each type of the accelarator is different and simulation times vary. By using Synopsys Design Compiler, area of the circuits are determined for conceptual insight.

Firstly, fault-free simulations are carried out and results are evaluated for the metrics. Disparty maps from the accelarators are compared with ground-truth images by Matlab. Creation of disparity maps are approximately 7 minutes at that stage. Then faults are injected. Simulations are repeated for faulty simulations. Fault lists are created intiutively and systematically. Questa simulator is used for all simulations.

Following chapters will present results as grouping results according to window sizes (5x5, 7x7 and 9x9). Then results will be given according to kernel types of the census transform under each window sizes.

6.1. Fault-Free Simulation Results

The simulations are run on Questa simulator. Three images are used: tsukuba, cones and venus. The output images and results of the evaluation metrics are given in following subsections.

6.1.1. Fault-free Results: 5x5 Window Size

As can be seen, non-redundant, uniform, and full kernel configurations provide similar accuracy results for all images. As the number of neighboring values decreases, the accuracy value drops. This is most evident in the Venus image. The accuracy drops 40% for 1-Point, 2-Point, and 4-Point versions concerning the Full configuration. Tsukuba and cones images exhibit better performance in terms of accuracy drop. The differences, on the other hand, are not pronounced enough to be visible from the images in Figure 40, Figure 41, and Figure 42.

- ***Tsukuba***

The accelerator performs the best performance for an 8-point design according to the accuracy as seen in Table 2. Non-redundant and full designs follow it. As the name indicates, the 8-Point design uses fewer neighborhoods concerning non-redundant and full designs. The worst accuracies are calculated for 1-Point and 2-Point designs and the difference can be observed in Figure 40. On the other hand, non-redundant and full designs are almost same PSNR and SNR values where full design uses all neighbourhoods inside the kernel window.

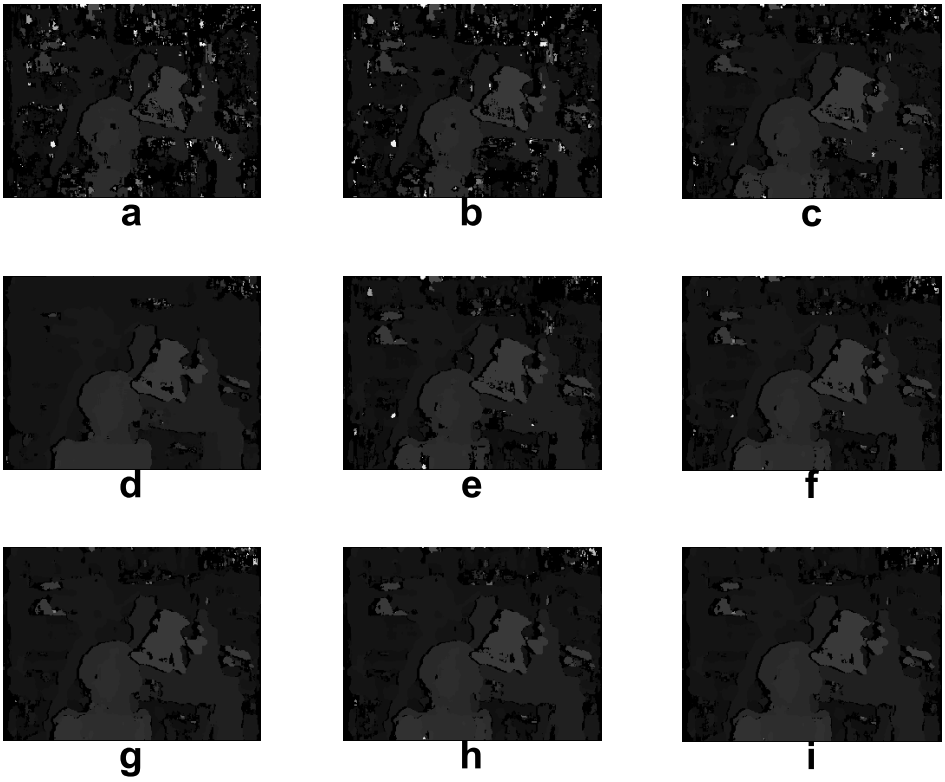

Figure 40 Tsukuba results for 5x5 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	86.68	86.99	91.60	93.19	91.58	92.08	93.10	92.27	93.10
PSNR	15.31	15.38	16.00	16.47	16.01	16.12	16.89	16.14	16.90
SNR	8.52	8.59	9.21	9.69	9.22	9.33	10.10	9.35	10.11

Table 2 Accuracy, PSNR, and SNR values for Tsukuba images, 5x5 window size

- ***Cones***

There are slight changes for the Cones image across all metrics. The accelerator produces almost the same disparity maps as seen in Figure 41. The exceptions are 1-point and 2-point images where some flickering is present in the images.

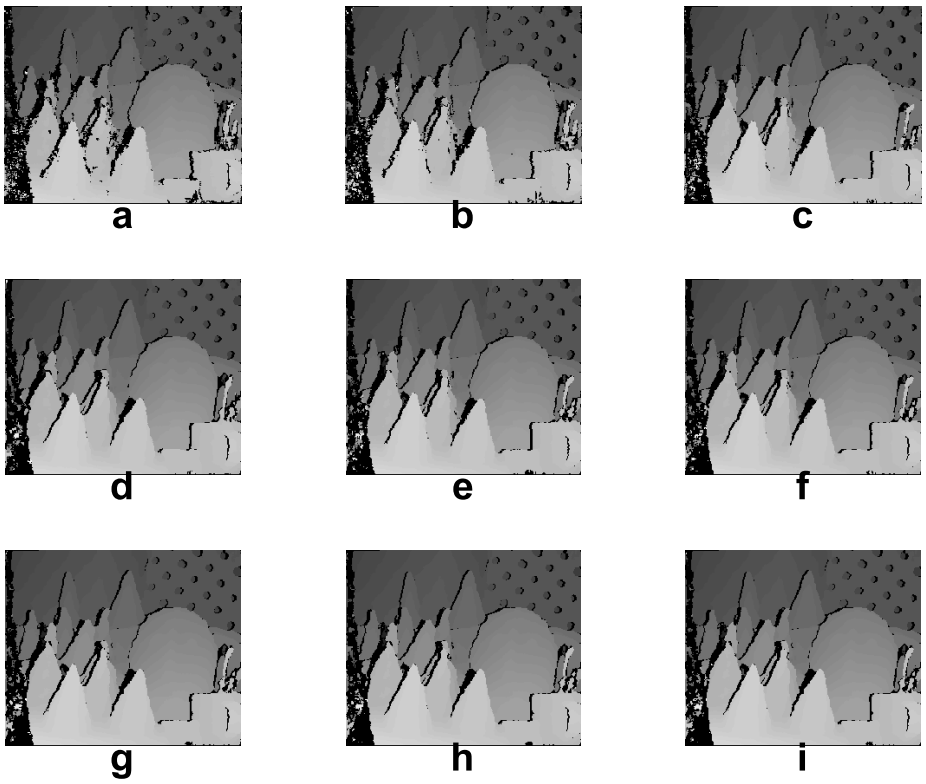

Figure 41 Cones results for 5x5 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	85.50	86.30	88.46	88.37	88.59	88.67	88.58	88.55	88.59
PSNR	13.75	14.04	14.69	14.73	14.78	14.79	14.78	14.73	14.79
SNR	8.42	8.72	9.36	9.41	9.45	9.47	9.45	9.41	9.46

Table 3 Accuracy, PSNR, and SNR values for Cones images, 5x5 window size

- ***Venus***

Despite full design having the best metrics, 8-point design performs almost the same with fewer neighborhoods. As noticed, this is the case also in the Tsukuba image. 1-point, 2-point, 4-point, and 12-point designs generate the worst disparity maps as seen in Figure 42. There are also differences between non-redundant and uniform designs that have the same number of neighborhoods. It can be said that a selection of the neighborhoods also affects the results for the same number of neighborhoods.

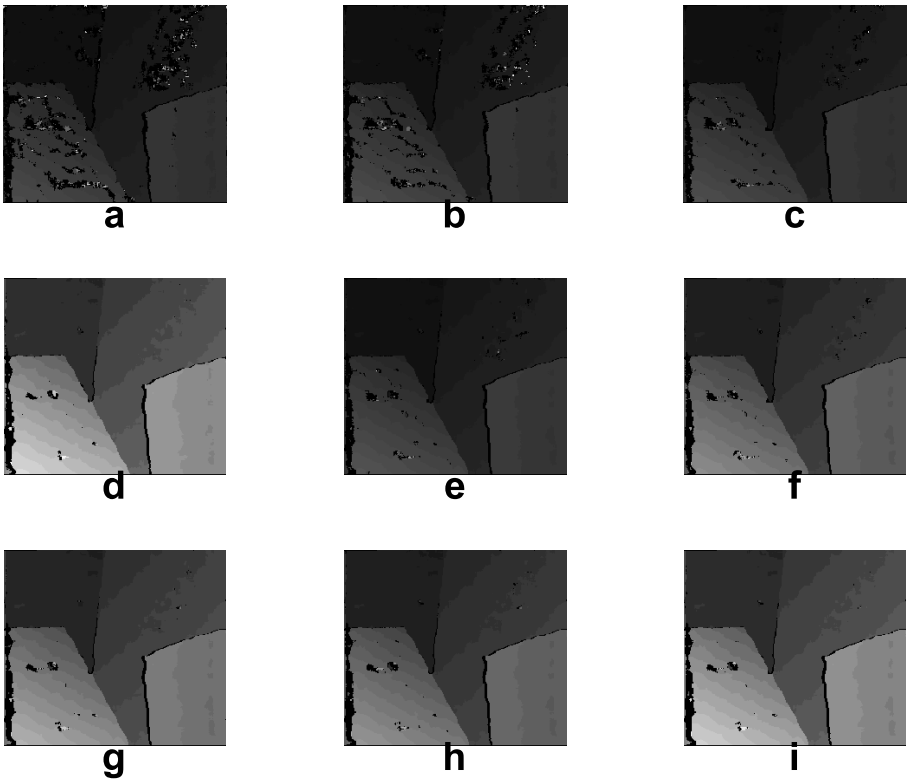

Figure 42 Venus results for 5x5 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	57.56	57.72	57.80	97.29	58.21	78.08	95.57	84.57	97.65
PSNR	16.46	16.74	16.92	25.71	17.32	21.10	24.07	21.82	25.76
SNR	5.95	6.22	6.41	15.20	6.80	10.58	13.56	11.31	15.25

Table 4 Accuracy, PSNR, and SNR values for Venus images, 5x5 window size

6.1.2. Fault-free Results: 7x7 Window Size

- ***Tsukuba***

The accelerator act as same except for 1-point and 2-point designs. This is similar to 5x5 design for tsukuba image. Non-redundant design provides the best numbers and 1-point has poor results.

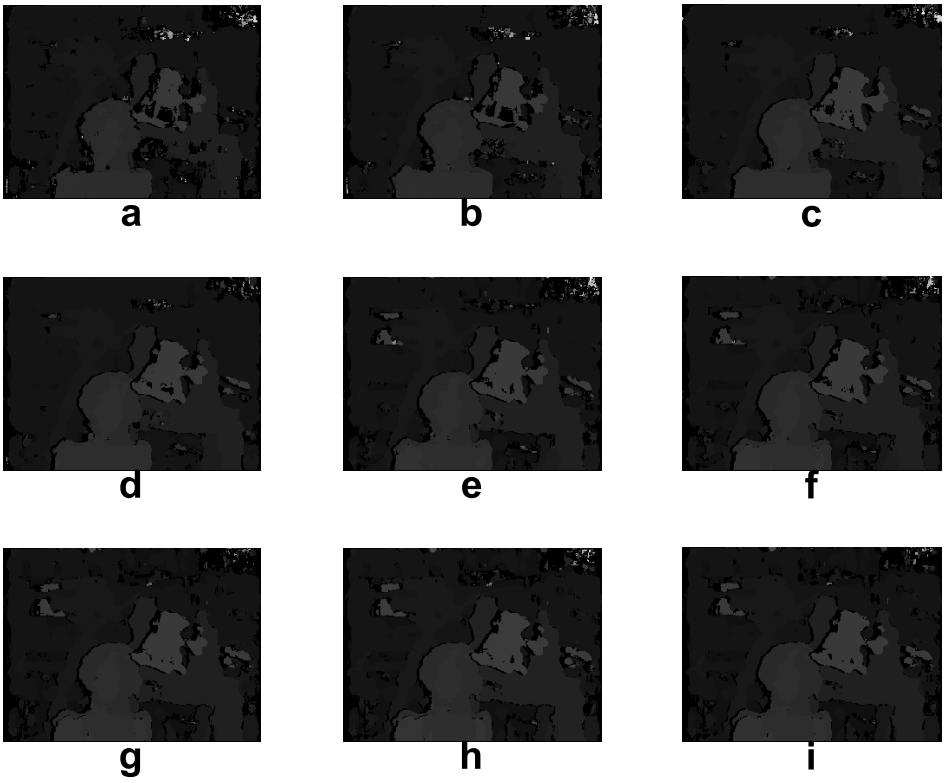

Figure 43 Tsukuba results for 7x7 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	2-Point	4-Point	8-Point	12-Point	16-Point	Non-Redundant	Uniform	Full
Accuracy	87.12	87.44	93.15	93.19	93.12	93.15	93.20	93.16	93.10
PSNR	15.58	15.70	16.32	16.47	16.86	16.49	16.54	16.51	17.52
SNR	8.79	8.91	9.53	9.69	10.07	9.70	9.75	9.72	10.74

Table 5 Accuracy, PSNR, and SNR values for Tsukuba images, 7x7 window size

- ***Cones***

As for the 5x5 window size, the results are similar to each other. 4-point has the highest accuracy values, on the other hand, the 16-point design has the best values for PSNR and SNR. Here, a 2% accuracy drop is ambiguous for cones. Generally speaking, there are no substantial differences.

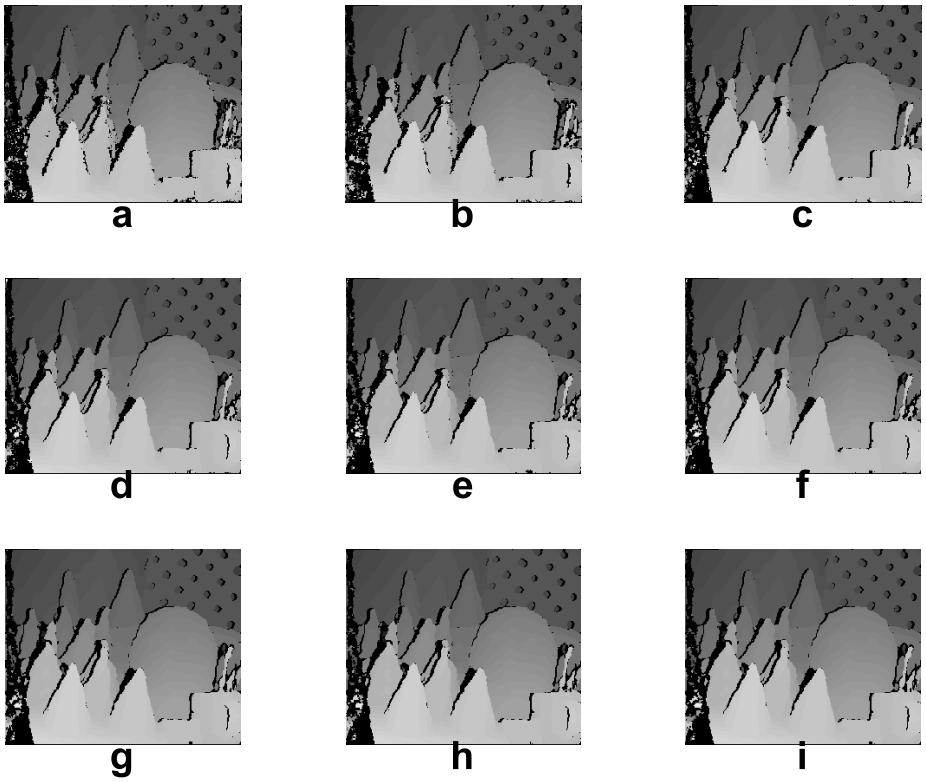

Figure 44 Cones results for 7x7 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	86.08	86.51	88.57	88.37	88.53	88.56	88.42	88.44	88.41
PSNR	14.01	14.17	14.72	14.73	14.77	14.78	14.75	14.76	14.76
SNR	8.69	8.84	9.39	9.41	9.44	9.46	9.43	9.43	9.43

Table 6 Accuracy, PSNR, and SNR values for Cones images, 7x7 window size

- ***Venus***

Significant changes occur when the number of neighborhoods is less than 8 points. Above 8 points, the accelerator almost behaves the same. The 16-point design has the best accuracy and is non-redundant, and the full design has the highest PSNR and SNR values.

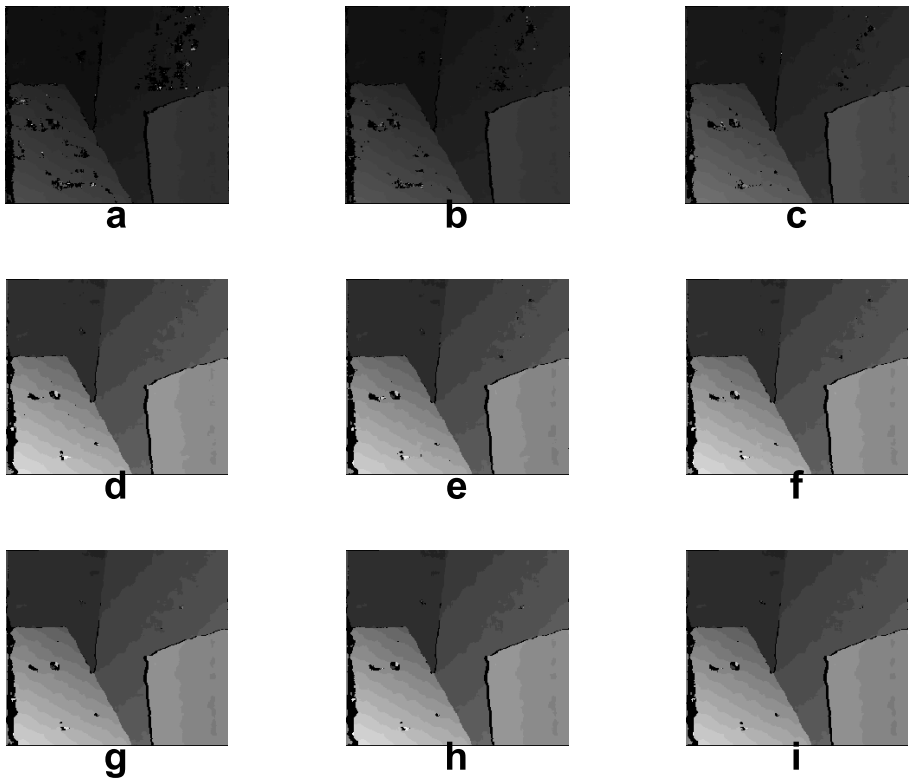

Figure 45 Venus results for 7x7 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	57.40	58.59	62.00	97.29	97.69	97.13	97.65	97.03	97.65
PSNR	16.82	17.52	19.14	25.71	25.78	25.66	25.85	25.59	25.85
SNR	6.30	7.00	8.63	15.20	15.27	15.15	15.34	15.08	15.33

Table 7 Accuracy, PSNR, and SNR values for Venus images, 7x7 window size

6.1.3. Fault-free Results: 9x9 Window Size

- ***Tsukuba***

When the number of neighborhoods is higher than 2 points, the disparity maps are nearly the same. In terms of SNR, there are at most 1.89 drops between 1-point and 12-point designs. The 12-point design has the highest PSNR value.

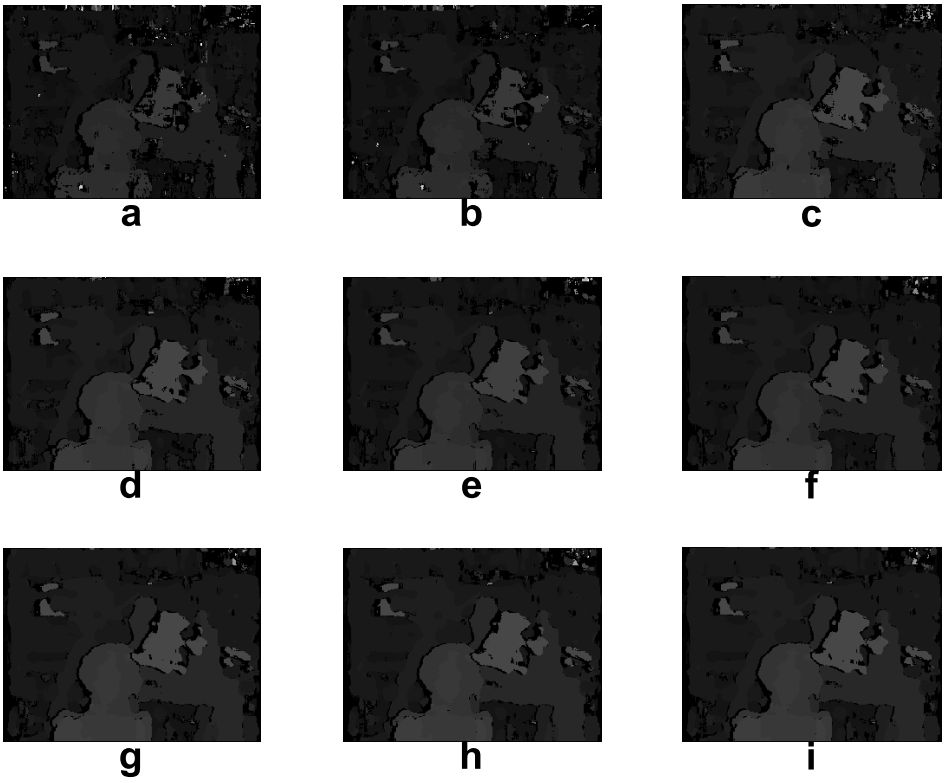

Figure 46 Tsukuba results for 9x9 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	2-Point	4-Point	8-Point	12-Point	16-Point	Non-Redundant	Uniform	Full
Accuracy	88.51	88.81	93.04	93.15	93.06	93.15	93.14	93.11	93.13
PSNR	15.71	15.77	16.52	16.57	17.60	16.78	17.56	17.53	17.53
SNR	8.92	8.98	9.73	9.78	10.81	9.99	10.77	10.74	10.74

Table 8 Accuracy, PSNR, and SNR values for Tsukuba images, 9x9 window size

- ***Cones***

The accuracy values fall in the range of 88 except for 1-point and 2-point designs. In terms of PSNR, rising trend, from 1-point to full, drops remarkable for non-redundant design. This is also the case for 5x5 and 7x7 designs, however, the drop is not significant. Lastly, uniform and full design have the strongest SNR value.

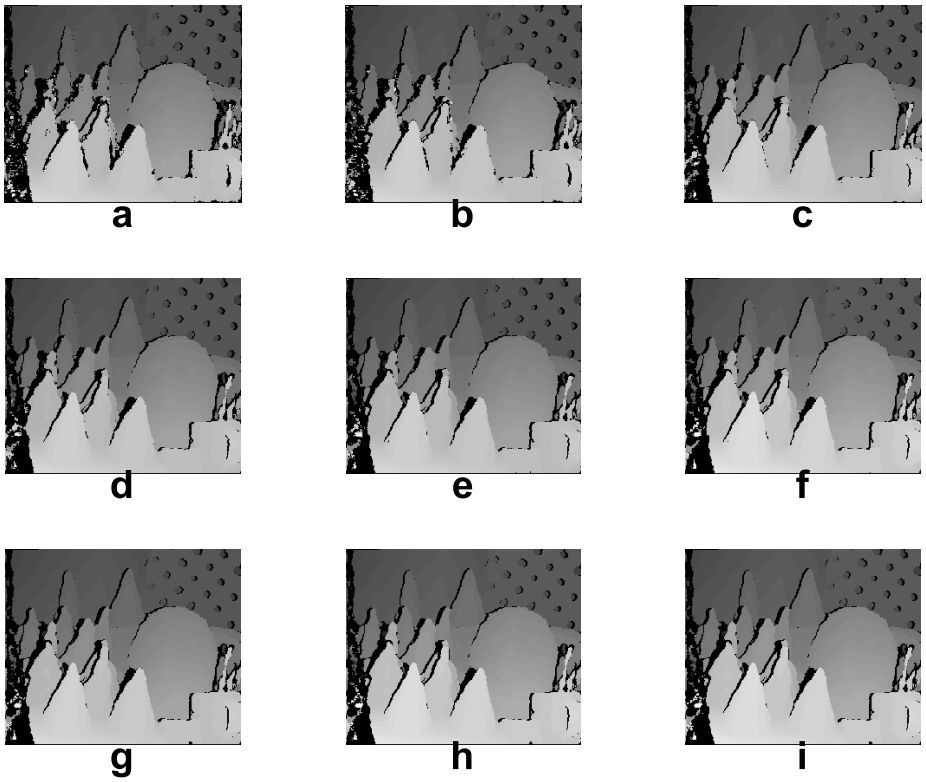

Figure 47 Cones results for 9x9 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	85.72	86.14	88.24	88.00	88.05	88.60	88.82	88.73	88.67
PSNR	13.91	14.03	14.70	14.61	14.67	15.10	14.79	15.13	15.12
SNR	8.58	8.71	9.38	9.29	9.34	9.78	9.47	9.80	9.80

Table 9 Accuracy, PSNR, and SNR values for Cones images, 9x9 window size

- ***Venus***

There is no remarkable difference between designs with same number of neighbourhoods which is slightly better than full design. The variations for the images are observable for 1-point, 2-point, and 4-point designs.

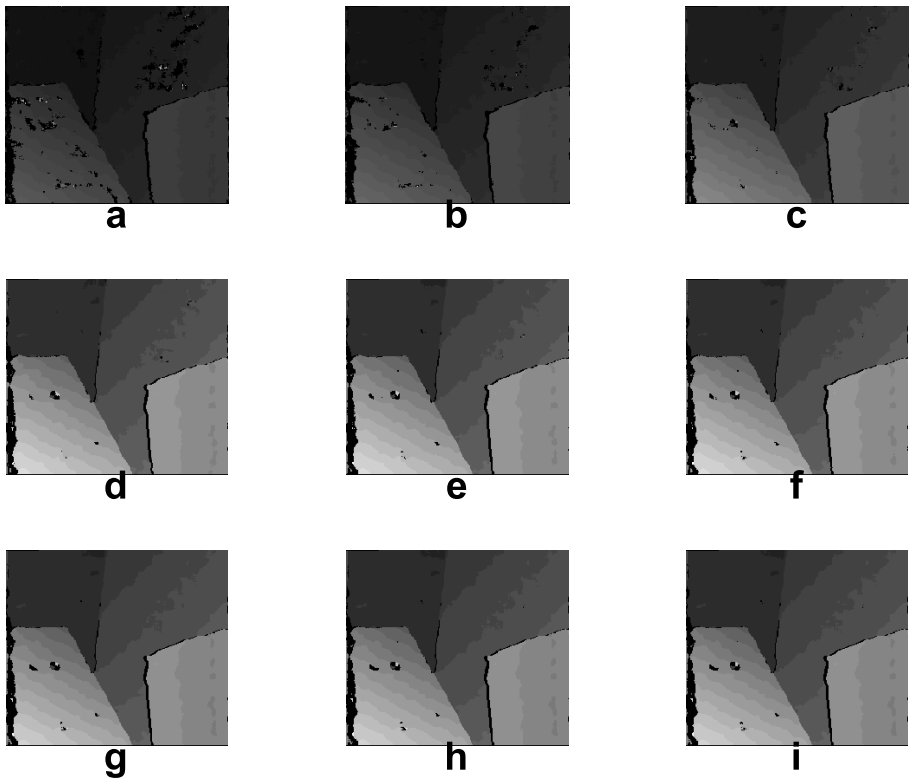

Figure 48 Venus results for 9x9 window. (a) 1-Point. (b) 2-Point. (c) 4-Point. (d) 8-Point. (e) 12-Point. (f) 16-Point. (g) Non-redundant. (h) Uniform. (i) Full.

	1-Point	**2-Point**	**4-Point**	**8-Point**	**12-Point**	**16-Point**	**Non-Redundant**	**Uniform**	**Full**
Accuracy	58.26	59.73	74.23	96.54	96.51	97.01	97.61	97.61	97.60
PSNR	17.44	18.37	20.93	22.92	22.86	25.61	25.84	25.83	25.83
SNR	6.92	7.86	10.42	12.41	12.34	15.10	15.33	15.32	15.32

Table 10 Accuracy, PSNR, and SNR values for Venus images, 9x9 window size

6.2. Faulty Simulation Results

The stuck-at faults are injected in the outputs of the *Disp_Comp*s in Figure 23. There are 63 comparator modules. The experiments are carried out for the uniform design with a 7x7 window size.

Graphs are organized according to averaging accuracy, PSNR, or SNR values concerning the bit positions. Since the output *o_data_C* is 12 bits and *o_data_D* is 7, a total of 19 bits start from 0. For instance, the accuracy values of 63 components at the bit-0 position are computed and then plotted. This is the same for other metrics. In addition, the golden metrics are indicated as a dotted line. Furthermore, stuck-at-0 and stuck-at-1 results are presented separately.

- ***Tsukuba***

The accuracies are depicted in Figure 49 and Figure 52 for stuck-at-0 and stucck-at-1 respectively. All disparity maps from the stereo vision accelerator suffer from the accuracy reduction. The best accuracy under the stuck-at faults is obtained at bit position 15 with 88.84% for both fault types. Remember from Table 5, the accuracy for fault-free disparity map is 93.16%. The sharpest fall is observed for bit-17 that is 6^{th} bit of the output *o_data_D*. The accuracy is 85.28% for both fault types. A slight drop regarding to a trend of accuracy line occurred at bit-11, the most significant bit of the output *o_data_C* with 88.18% for both cases.

A similar trend is followed by PSNR and SNR metrics. As noticed, the accuracy when the fault is injected in bit position 16, is better than bit position 17. However, bit-16 seems the most corrupted according to these metrics. This outcome can be seen in Figure 55. The disparity map is most corrupted when the fault is injected at bit position 16.

The behavior of the accelerator under the fault is the same for all components except for the last component, 63. The same bit positions behave differently when the fault (*sa0* or *sa1*) is injected as seen in Figure 56. This case can be confirmed by Table 11.

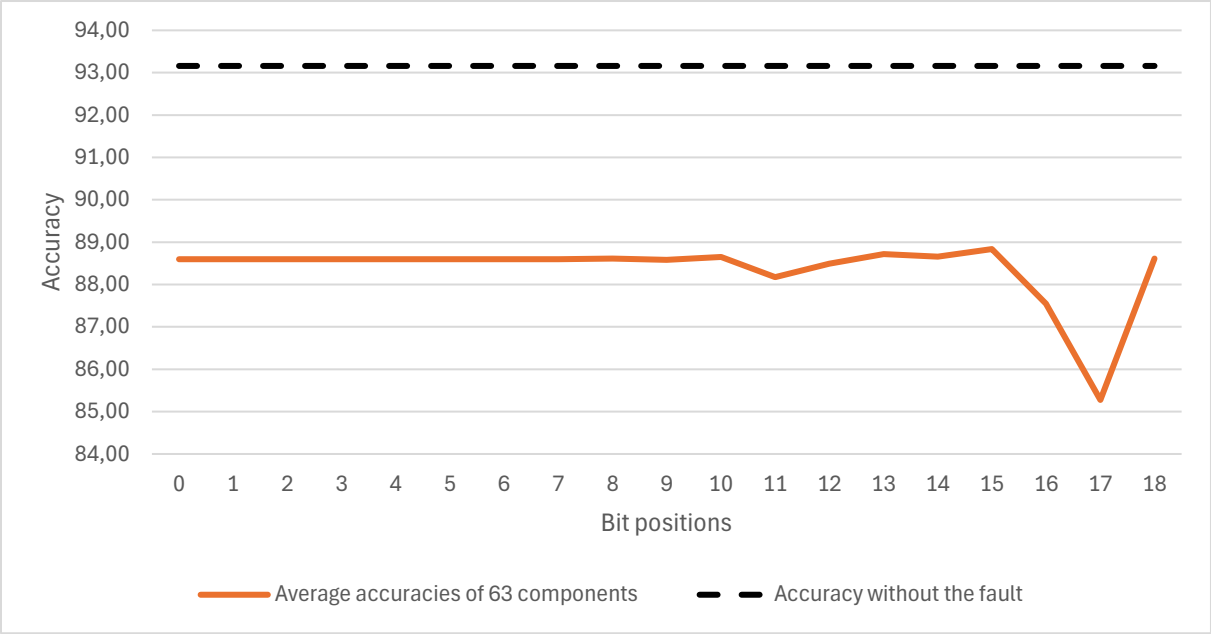

Figure 49 The stuck-at-0 accuracy results of Tsukuba for uniform and 7x7 window size

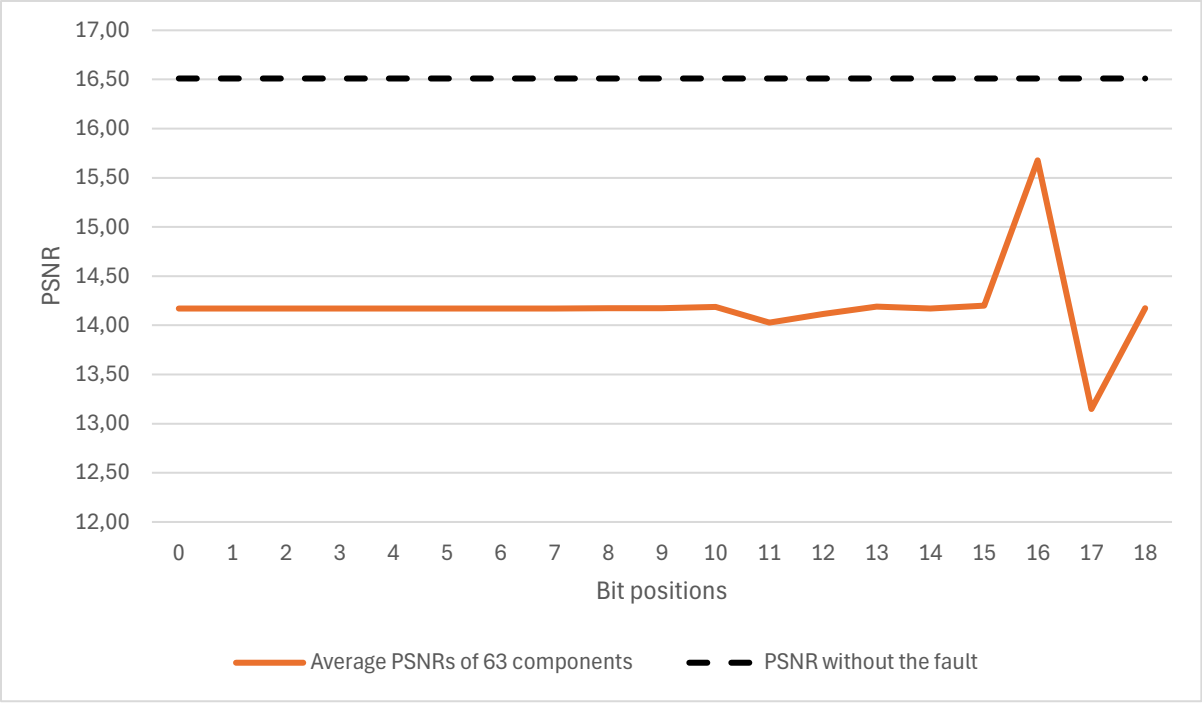

Figure 50 The stuck-at-0 PSNR results of Tsukuba for uniform and 7x7 window size

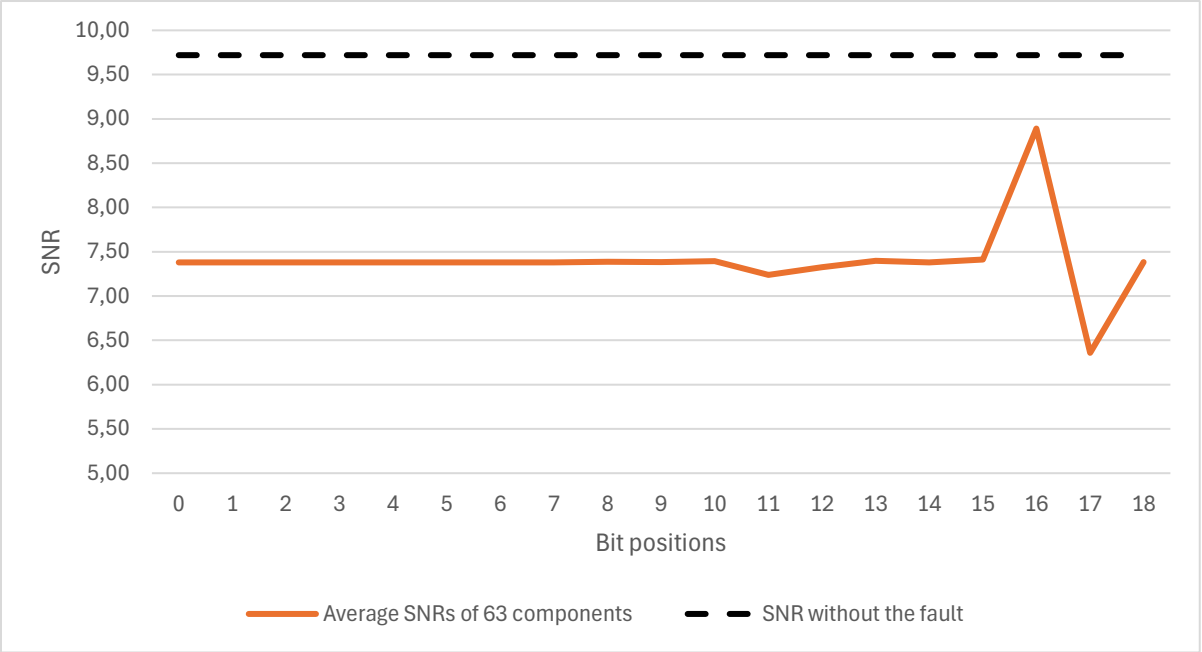

Figure 51 The stuck-at-0 SNR results of Tsukuba for uniform and 7x7 window size

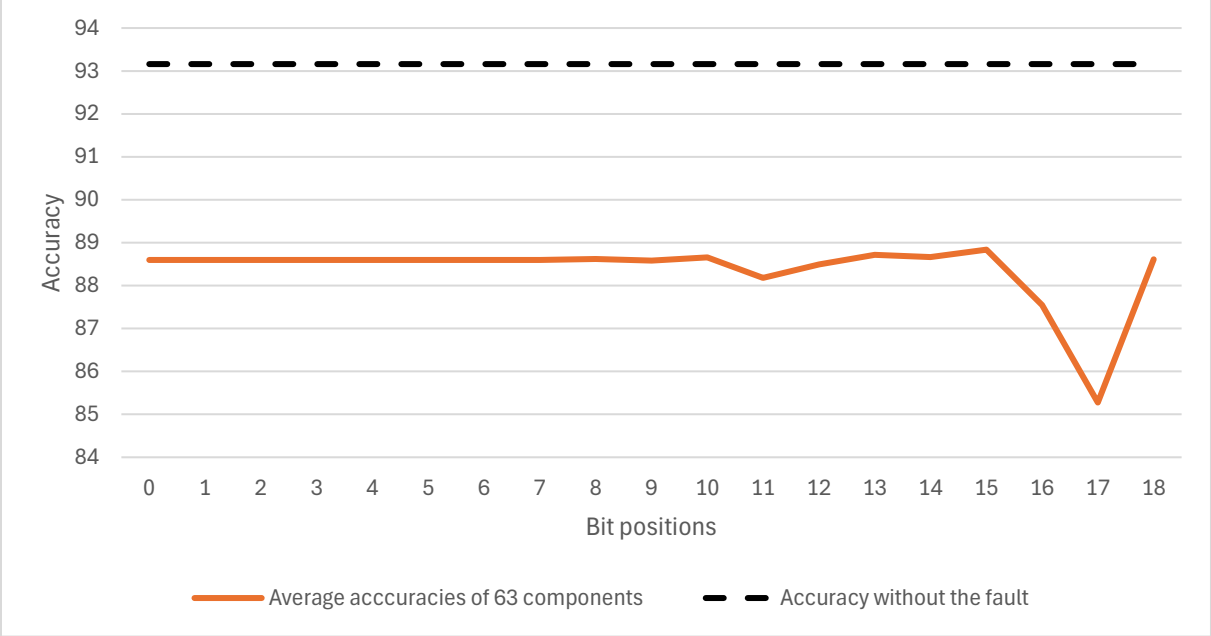

Figure 52 The stuck-at-1 accuracy results of Tsukuba for uniform and 7x7 window size

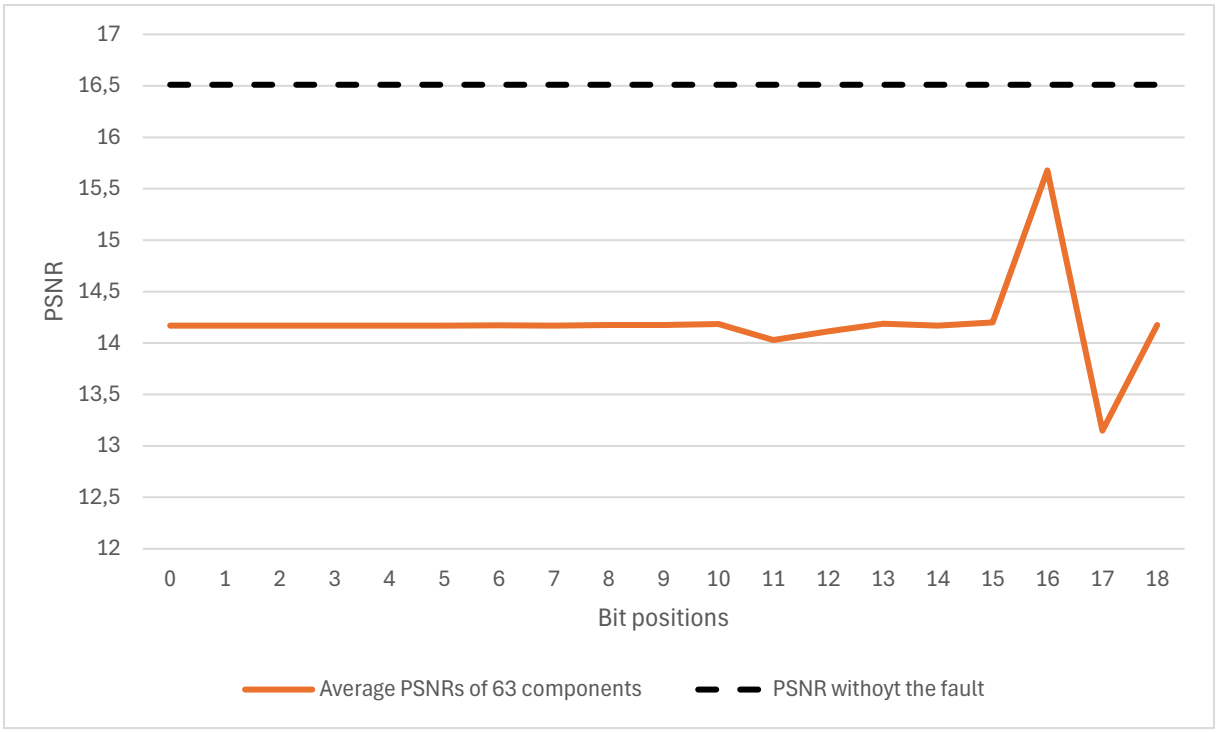

Figure 53 The stuck-at-1 PSNR results of Tsukuba for uniform and 7x7 window size

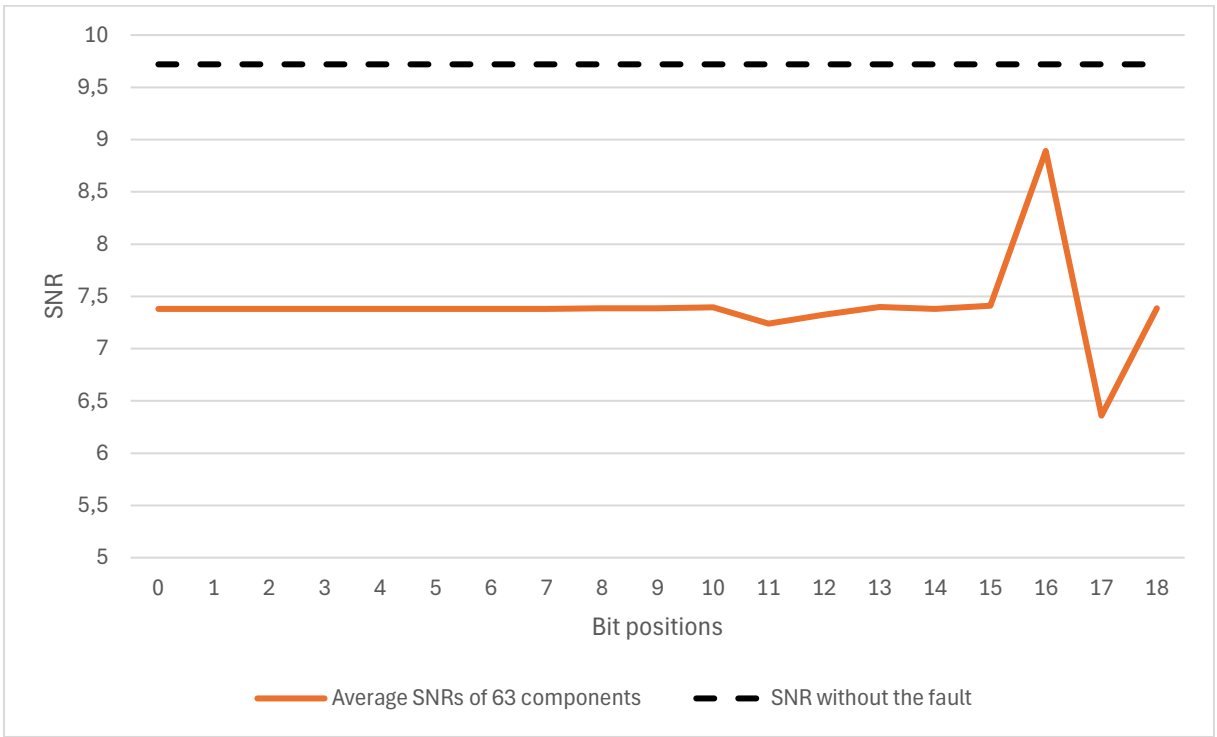

Figure 54 The stuck-at-1 SNR results of Tsukuba for uniform and 7x7 window size

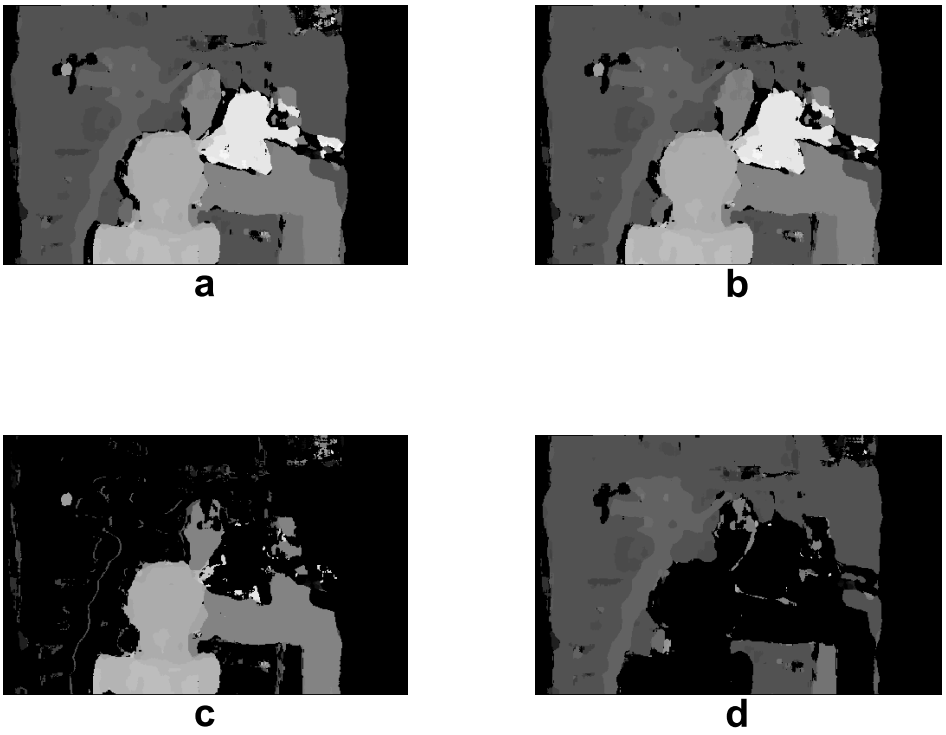

Figure 55 Faulty Tsukuba for component 33. Stuck-at-1 fault is injected at: a) Bit-11, b) Bit-15, c) Bit-16, d) Bit-17.

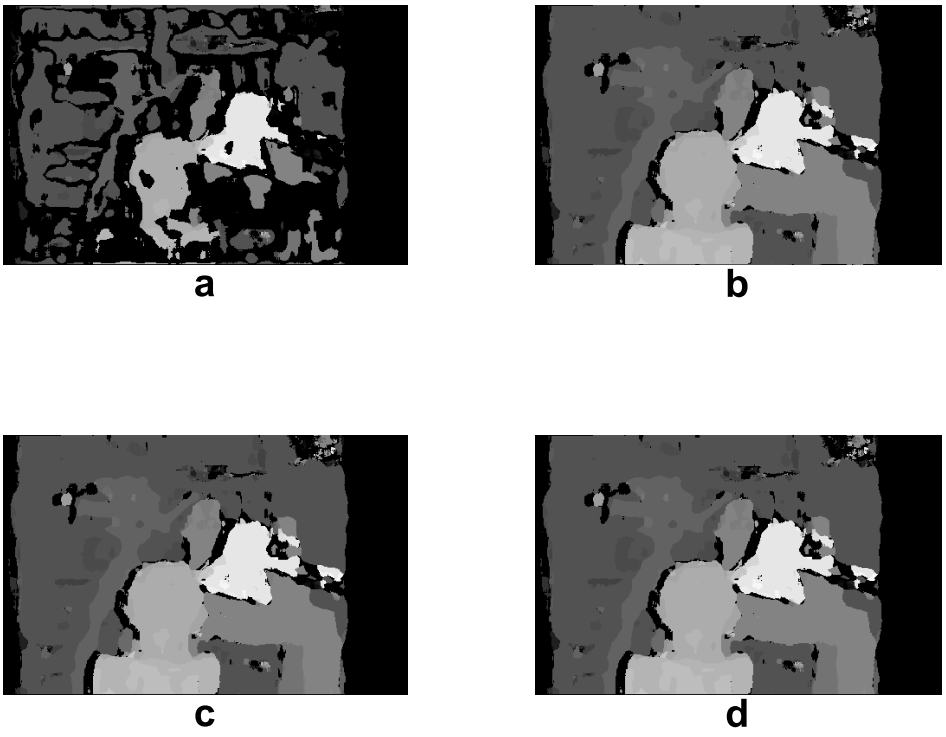

Figure 56 Faulty Tsukuba for component 63. Stuck-at-1 fault is injected at: a) Bit-11, b) Bit-15, c) Bit-16, d) Bit-17.

BIT	ACCURACY	PSNR	SNR
11	87,48	14,39	7,6
15	88,6	14,17	7,38
16	88,6	14,17	7,38
17	88,6	14,17	7,38

Table 11 The metrics for component 63, *sa0*, *sa1*, and bit positions 11, 15, 16, and 17

- ***Cones***

As seen in Figure 57 and Figure 60, for *sa0* and *sa1* respectively, the accuracies drop dramatically at bit locations 11, 17, and 18 which correspond to the most significant bit of output *o_data_C* and the last two bits of output *o_data_D* respectively. The accuracies are 76.86, 62.39, and 73.54 percent respectively. In addition, the bit locations 10, 12, and 16 have a slight decrease with accuracies of 76.86, 87.99, and 77.73 percent. The accuracy without any fault is 88.44% from Table 6.

As noticed, these locations match with the upper half of the signals for all 63 components. Other metrics also follow the same trend at the bit locations. Figure 63 and Figure 64 show the images for the bit positions mentioned above for components 33 and 63. The fault effect appears on the images as an example. Finally, the comments are valid for both *sa0* and *sa1* fault types. On the other hand, the metrics are better than fault-free results on some bit positions. Figure 65 shows the comparison.

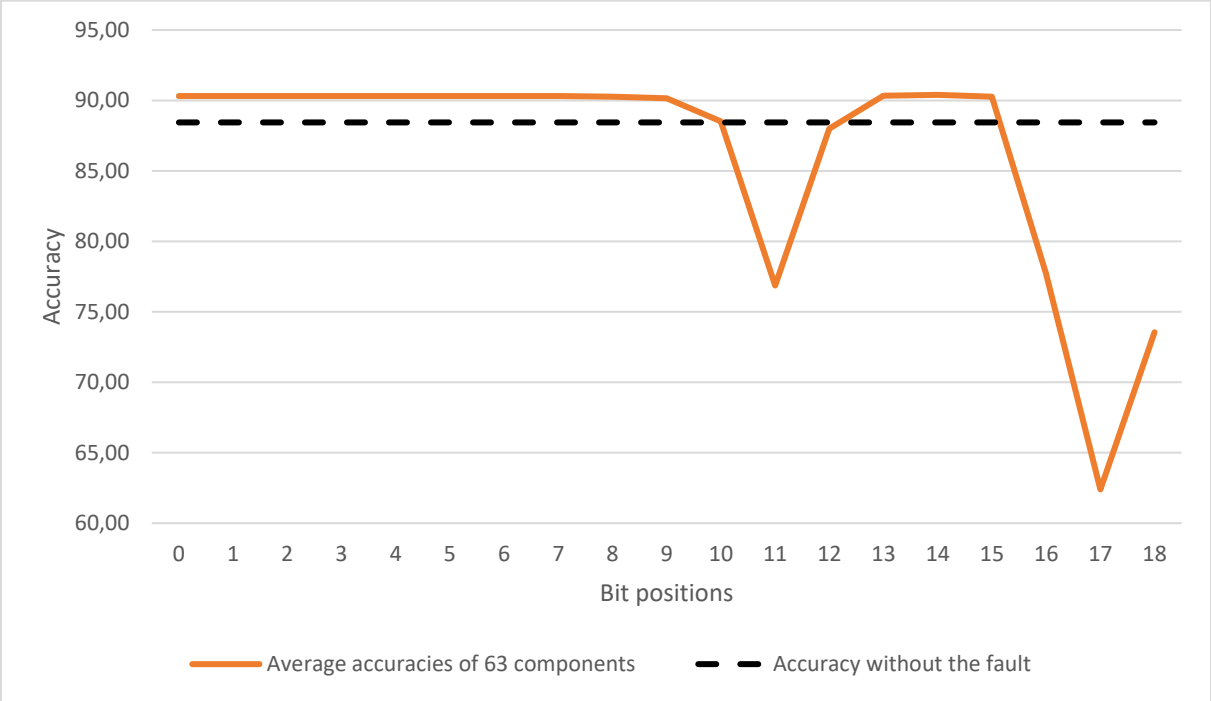

Figure 57 The stuck-at-0 accuracy results of Cones for uniform and 7x7 window size

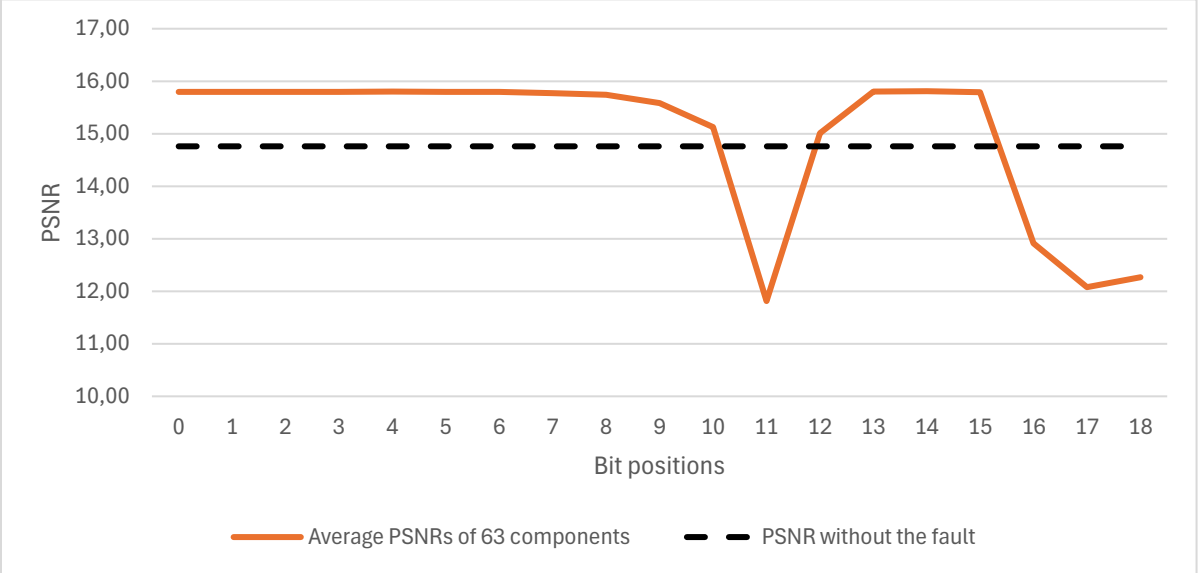

Figure 58 The stuck-at-0 PSNR results of Cones for uniform and 7x7 window size

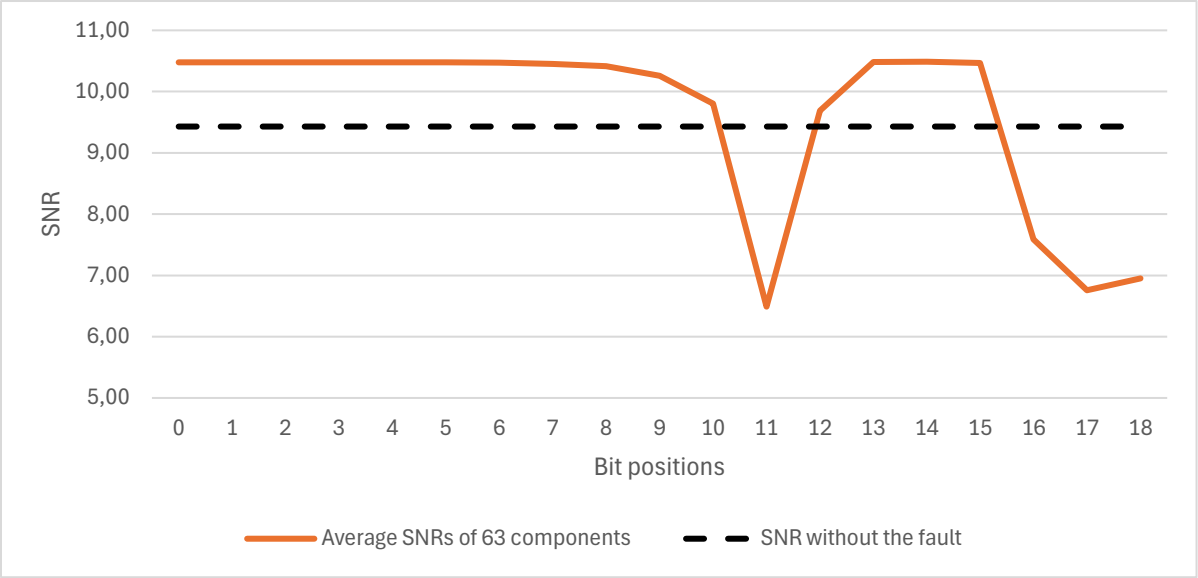

Figure 59 The stuck-at-0 SNR results of Cones for uniform and 7x7 window size

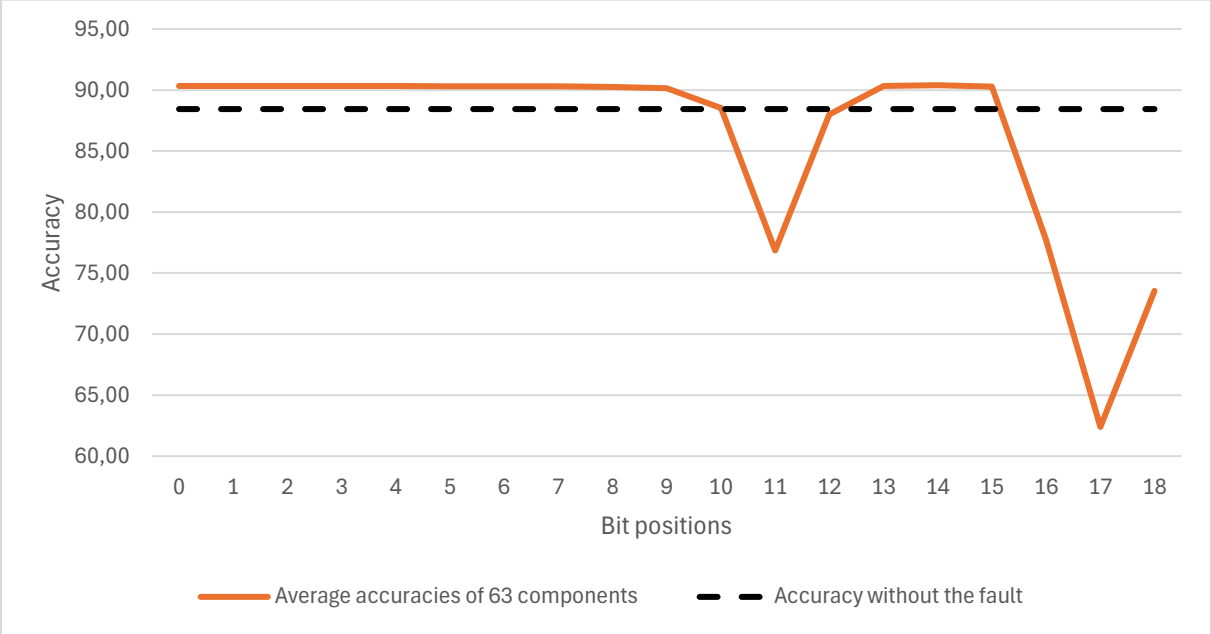

Figure 60 The stuck-at-1 accuracy results of Cones for uniform and 7x7 window size

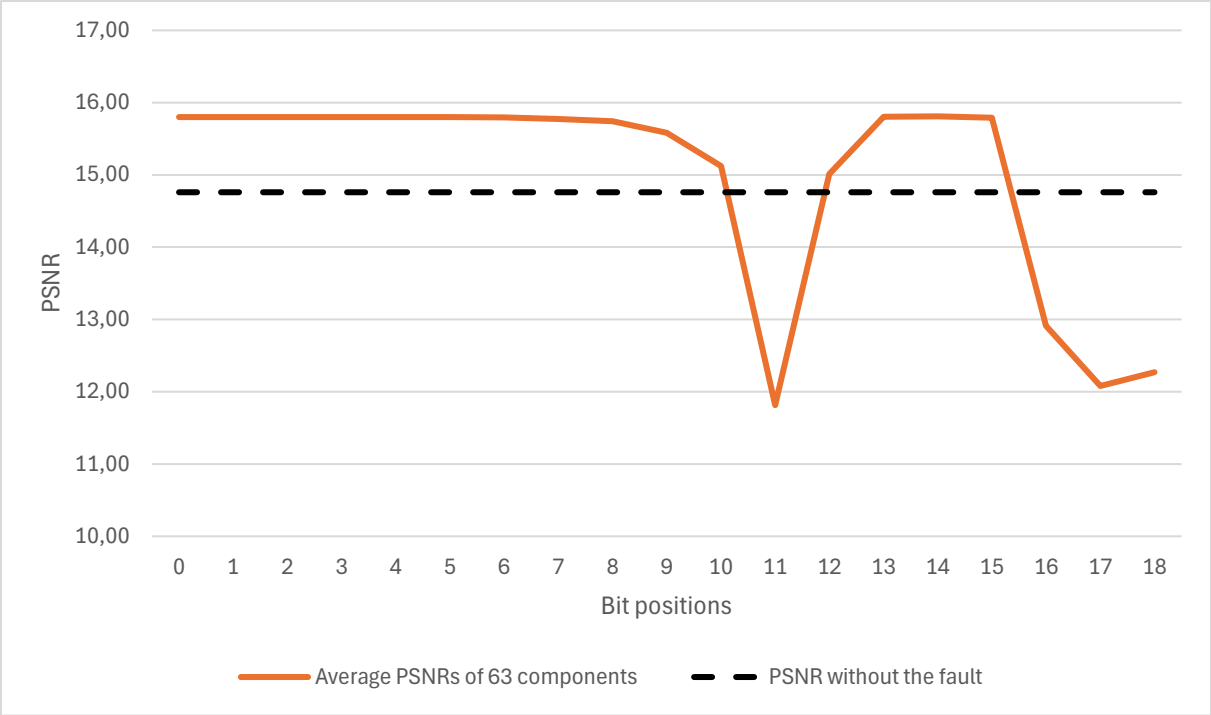

Figure 61 The stuck-at-1 PSNR results of Cones for uniform and 7x7 window size

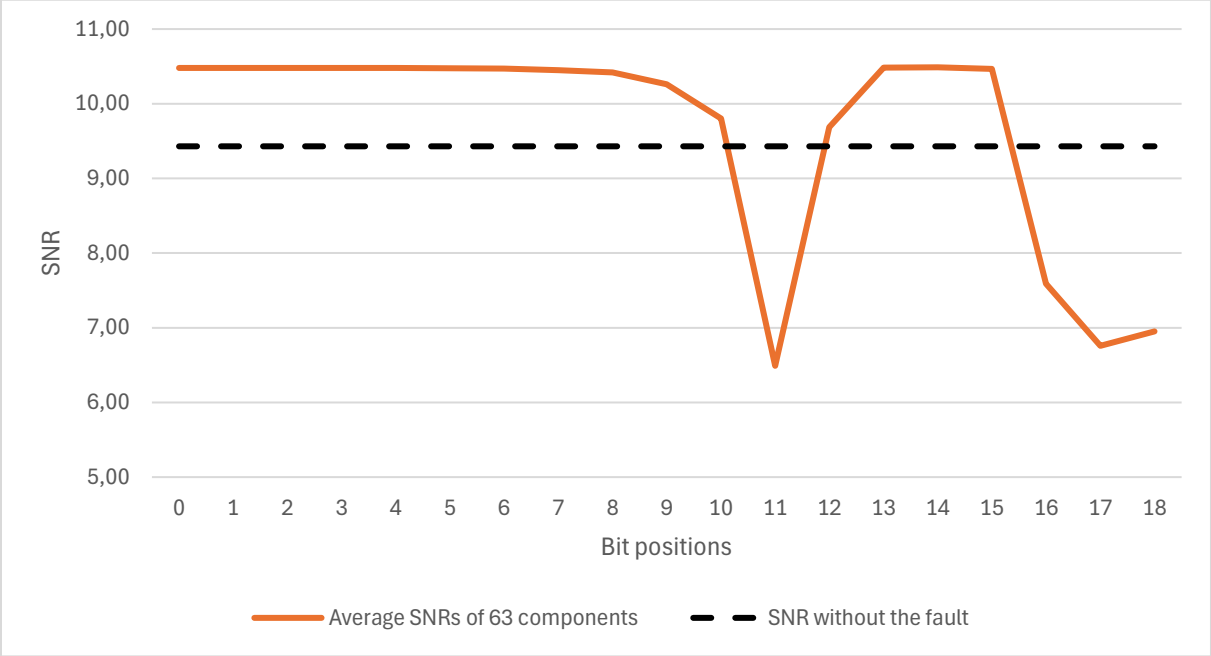

Figure 62 The stuck-at-1 SNR results of Cones for uniform and 7x7 window size

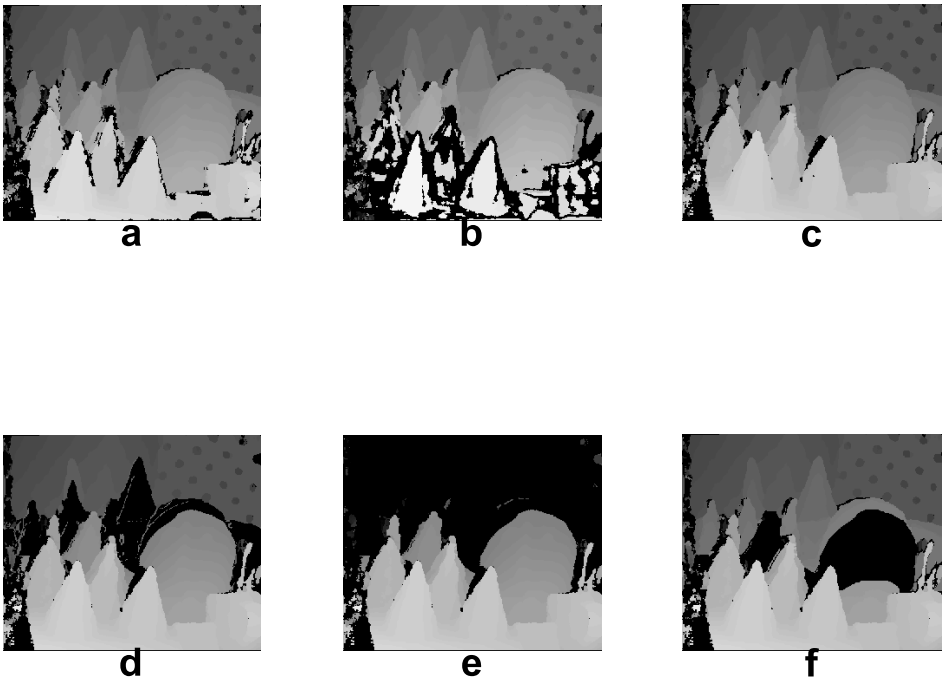

Figure 63 Faulty cones for component 33. Stuck-at-0 fault is injected at: a) Bit-10, b) Bit-11, c) Bit-12, d) Bit-16, e) Bit-17, and f) Bit-18.

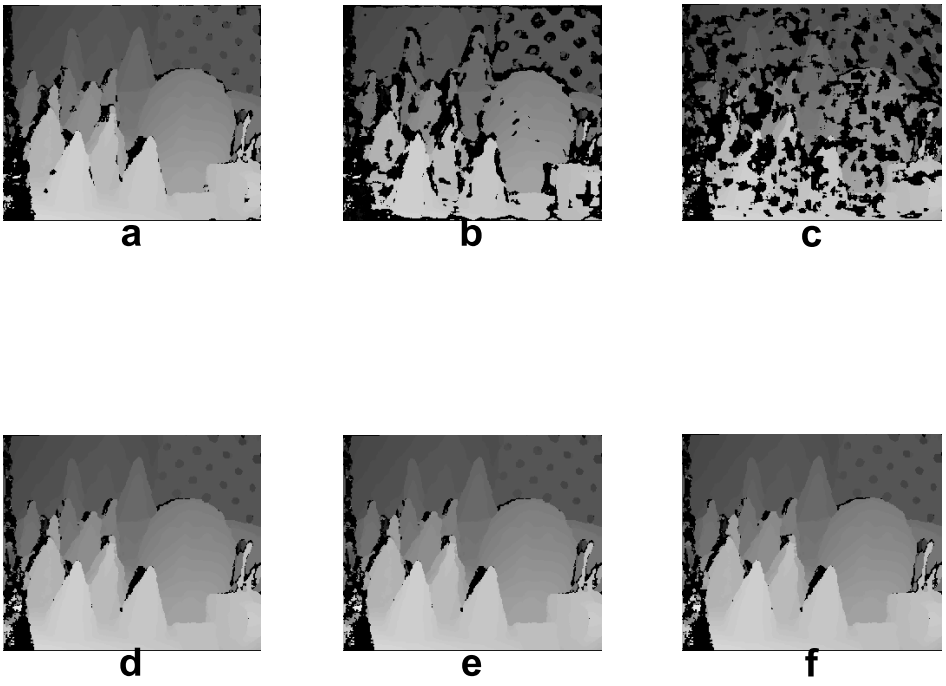

Figure 64 Faulty cones for component 63. Stuck-at-0 fault is injected at: a) Bit-10, b) Bit-11, c) Bit-12, d) Bit-16, e) Bit-17, and f) Bit-18.

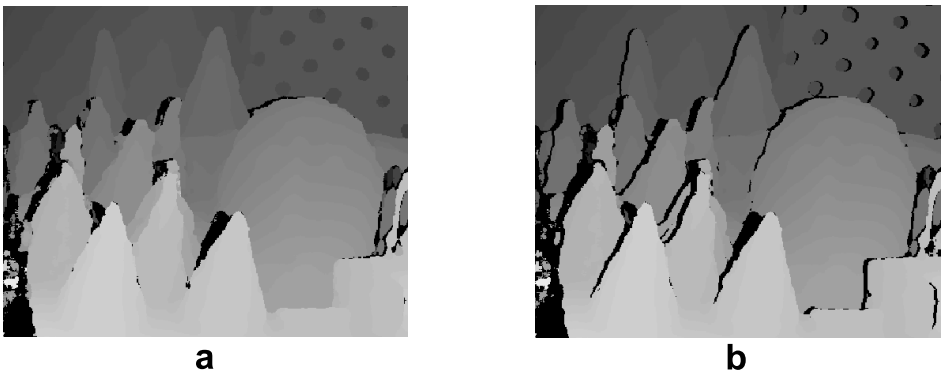

Figure 65 Cones images: a) sa0 fault at the 0th-bit position for component 30. b) Fault-free image

- ***Venus***

The behavior of the stereo vision accelerator under the fault injection is also maintained for the Venus image. As seen in Figure 66 and Figure 69, there are significant drops in bit locations 16 and 17 with accuracies of 81.89 and 45.56 percent respectively. The golden accuracy is 97.03 percent as given Table 7. The phenomenon is followed by also PSNR with 19.03 and 16.10 percents and SNR with 8.52 and 5.59 percents. The effect of both fault types is same on the output disparity maps. The golden values for PSNR and SNR is 25.59 and 15.08 percent respectively *for* sa0 and *sa1*. Lastly, as expected, there is a small degradation at bit position 11. Figure 72 shows the images for the mentioned bit positions. The degradation can be seen accordingly.

In terms of accuracy, the accelerator performs almost a good performance except the bits mentioned above. However, the PSNR and SNR metrics indicate that the difference is slightly larger. Figure 73 reveals that the difference, 1.61%, is observable.

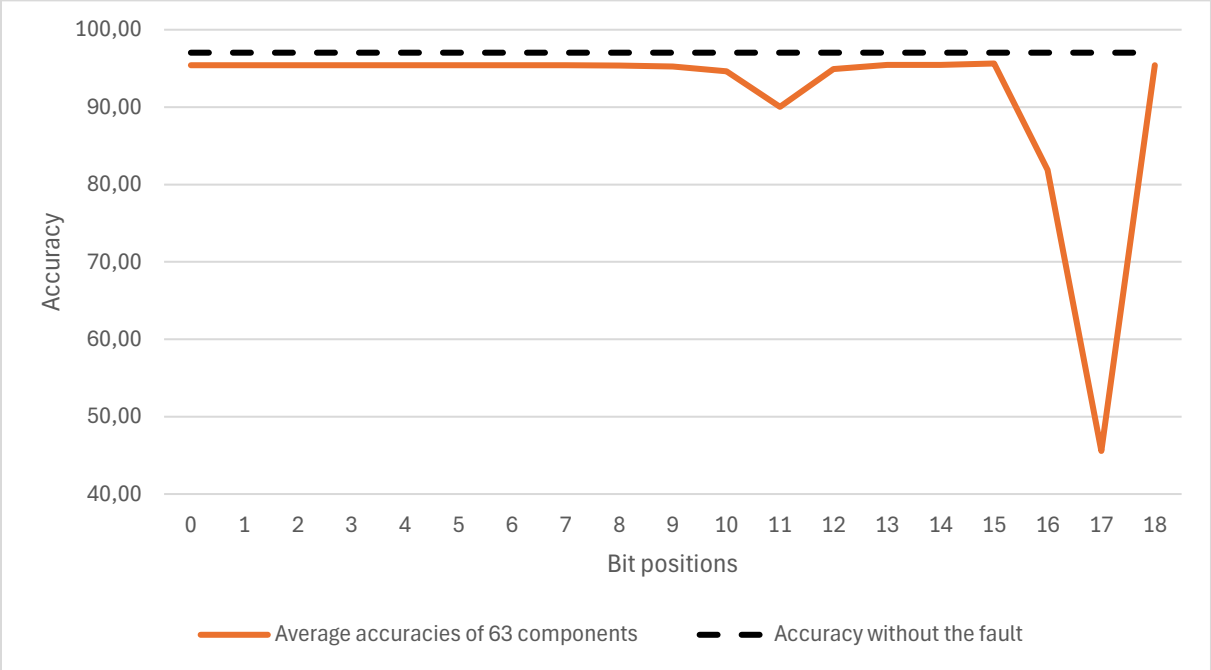

Figure 66 The stuck-at-0 accuracy results of Venus for uniform and 7x7 window size

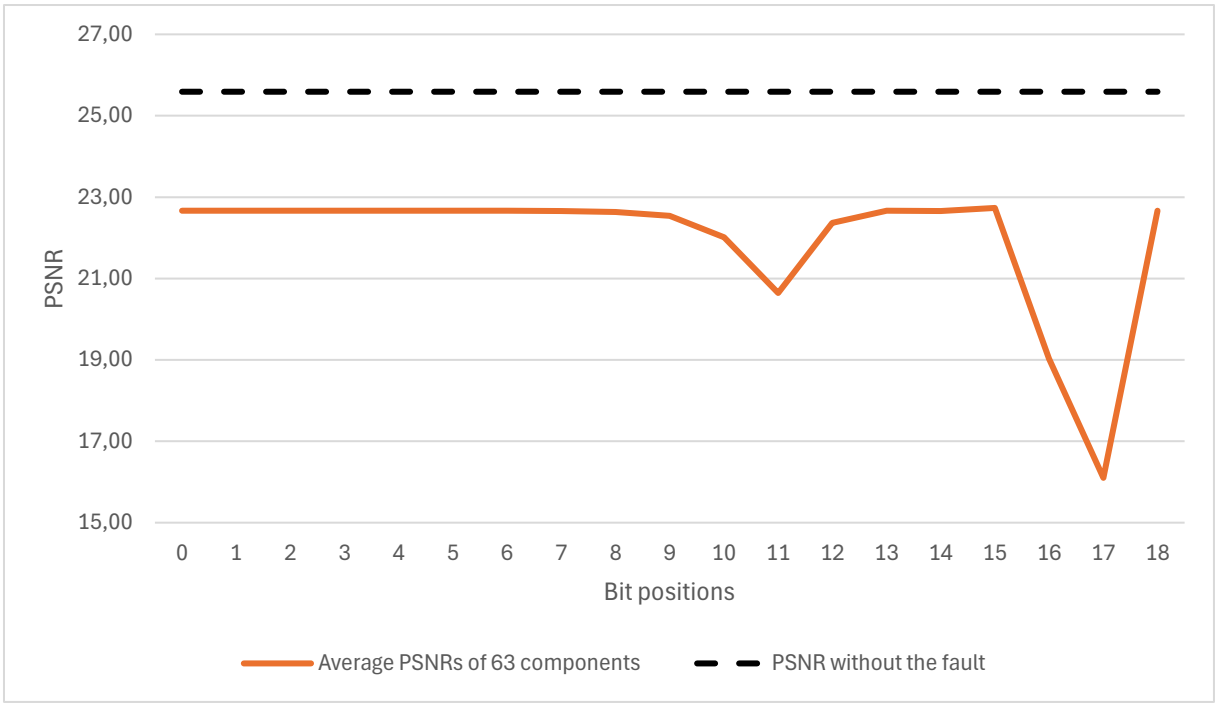

Figure 67 The stuck-at-0 PSNR results of Venus for uniform and 7x7 window size

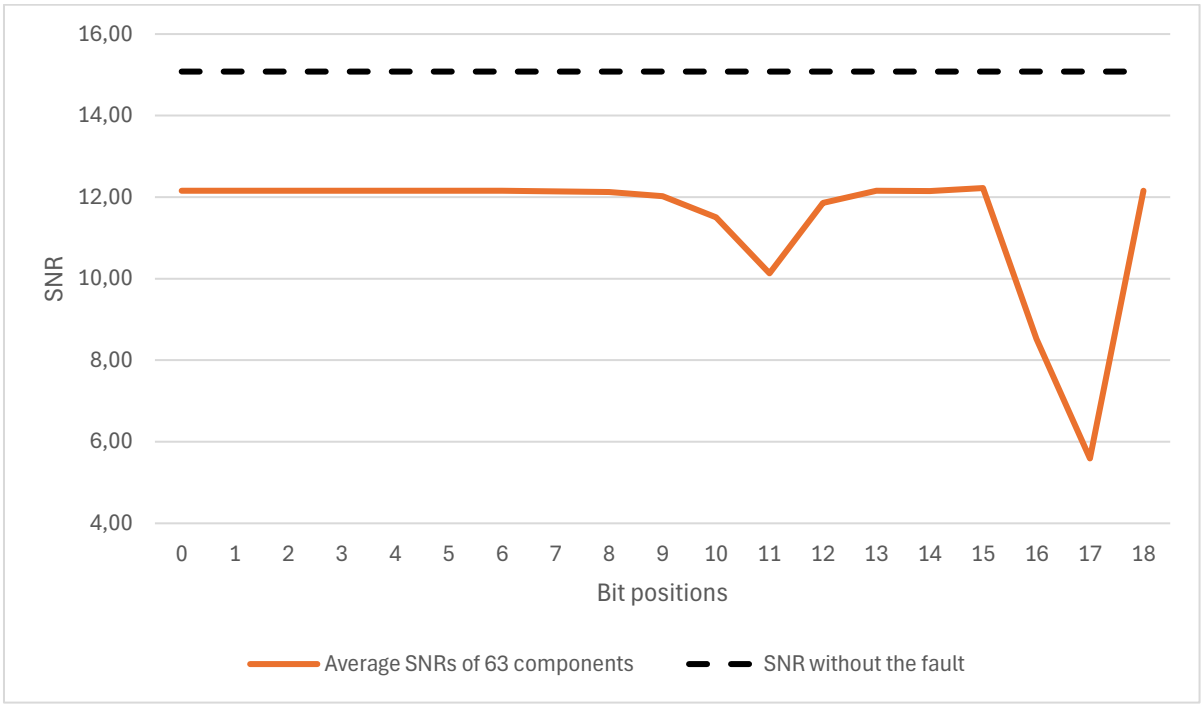

Figure 68 The stuck-at-0 SNR results of Venus for uniform and 7x7 window size

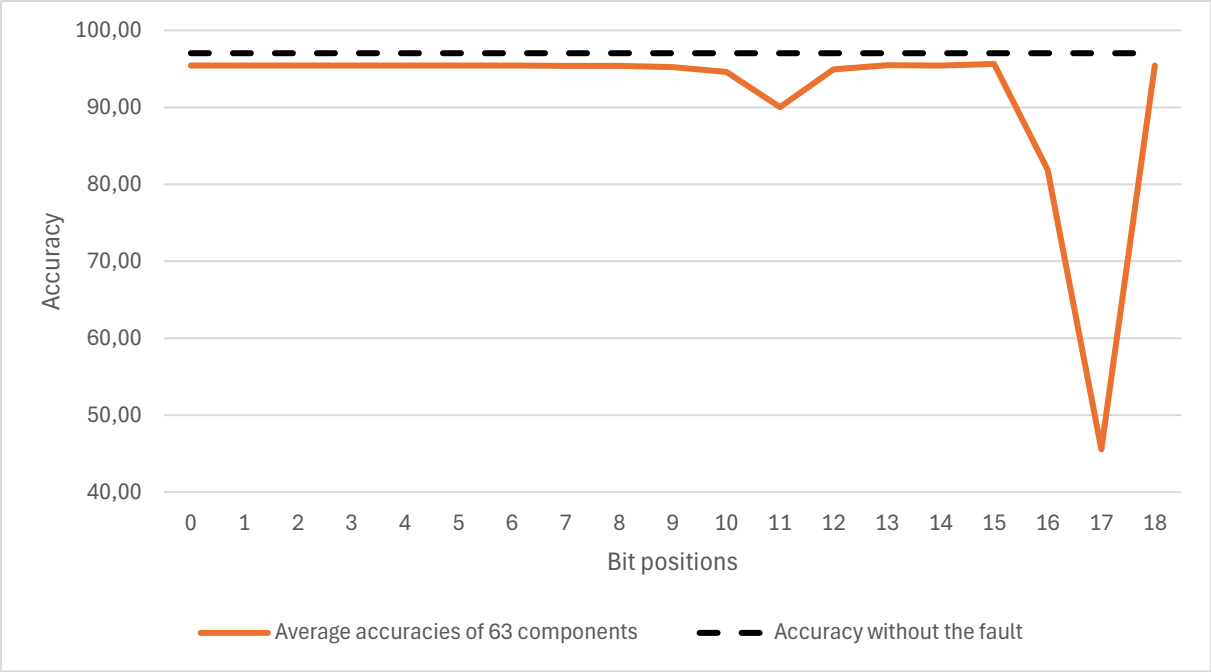

Figure 69 The stuck-at-1 accuracy results of Venus for uniform and 7x7 window size

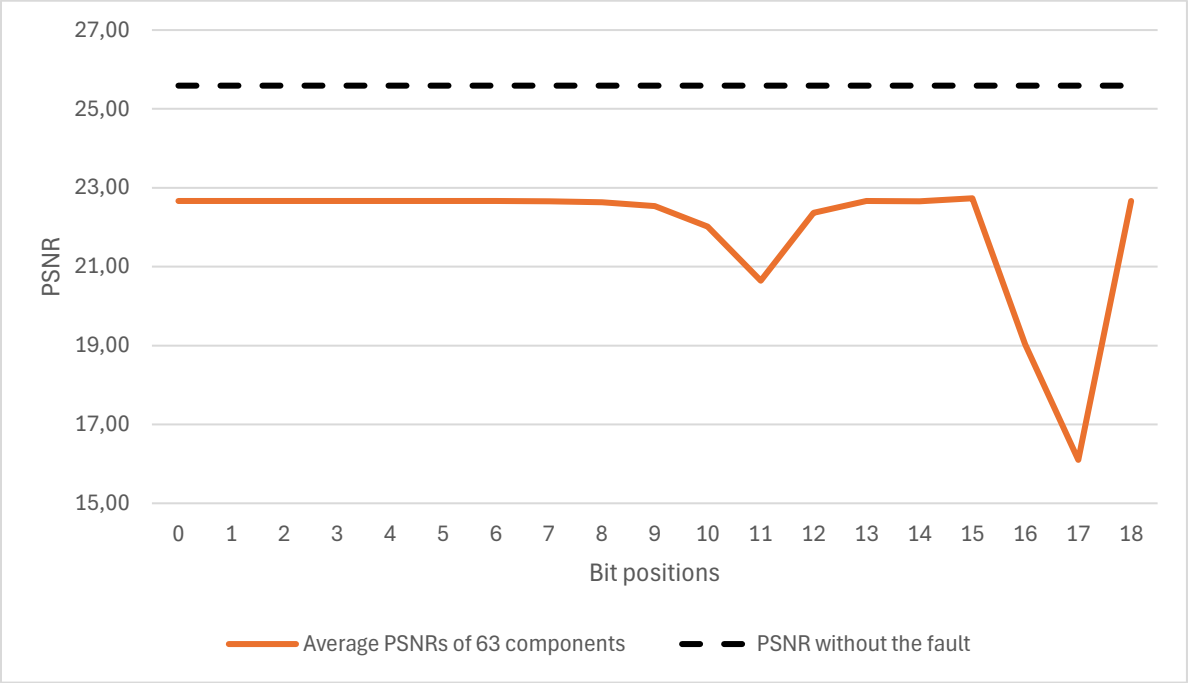

Figure 70 The stuck-at-1 PSNR results of Venus for uniform and 7x7 window size

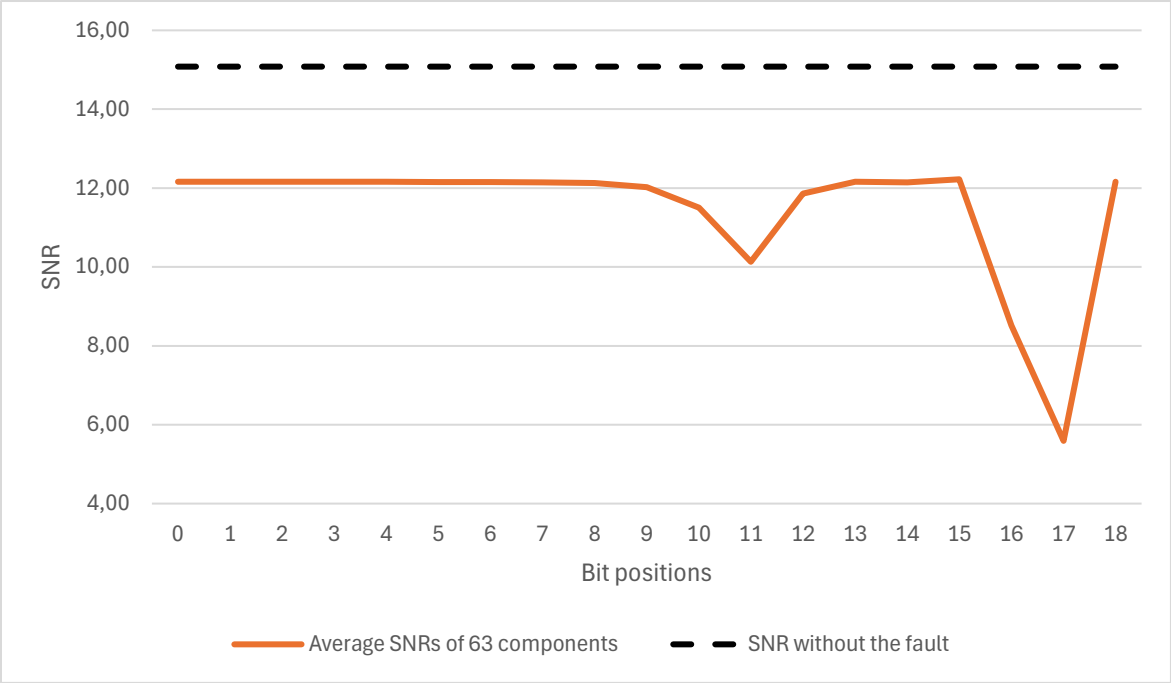

Figure 71 The stuck-at-1 SNR results of Venus for uniform and 7x7 window size

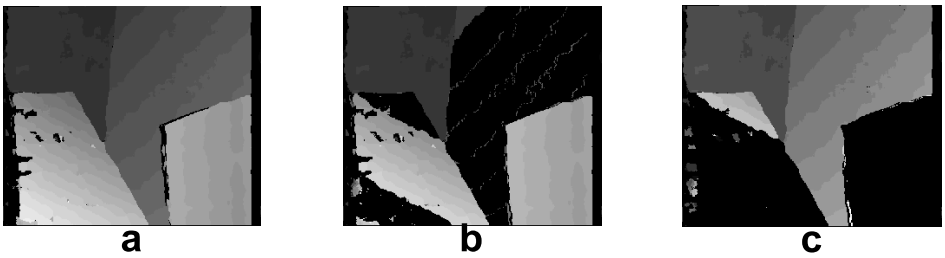

Figure 72 Faulty venus for component 33. Stuck-at-0 fault is injected at: a) Bit-11, b) Bit-16, and c) Bit-17.

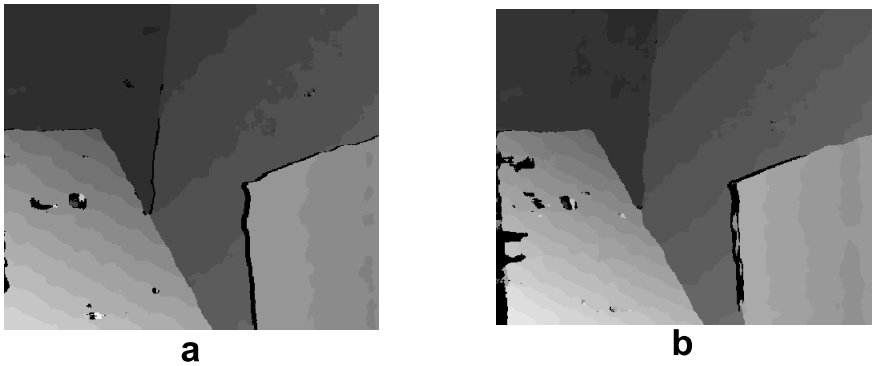

Figure 73 a) Golden disparity map of Venus (97.03% accuracy). b) When a fault is injected at component 10, bit position 1 for *sa0* (95.42% accuracy).

Until now, bits in the fault list, 19 bits, are focused in terms of the fault effect for each image. This provides which bit position affects the disparity map the most. The following analysis illustrates an average effect of the fault over components. As mentioned, there are 63 comparator modules cascading each other to determine the disparity level. To observe which comparator modules are the most critical in terms of fault propagation or compensation, comparisons across all images and comparators are aimed to be illustrated. The comparisons are made by separating *sa0* and *sa1* fault types. The metrics can be observed for each fault type.

From Figure 74 to Figure 79, it can be observed from all metrics that there are noticeable and expected decreases for all images. The fluctuations over the metrics are similar for the images at the same components. For instance, ripples can be observed for components number 8, 19, 30, and 41. When the final peak is reached among these components, there are no significant changes or fluctuations observed in the metrics. As can be noticed, the stereo vision accelerator can, generally speaking, mitigate the effects of the fault for the components mentioned above.

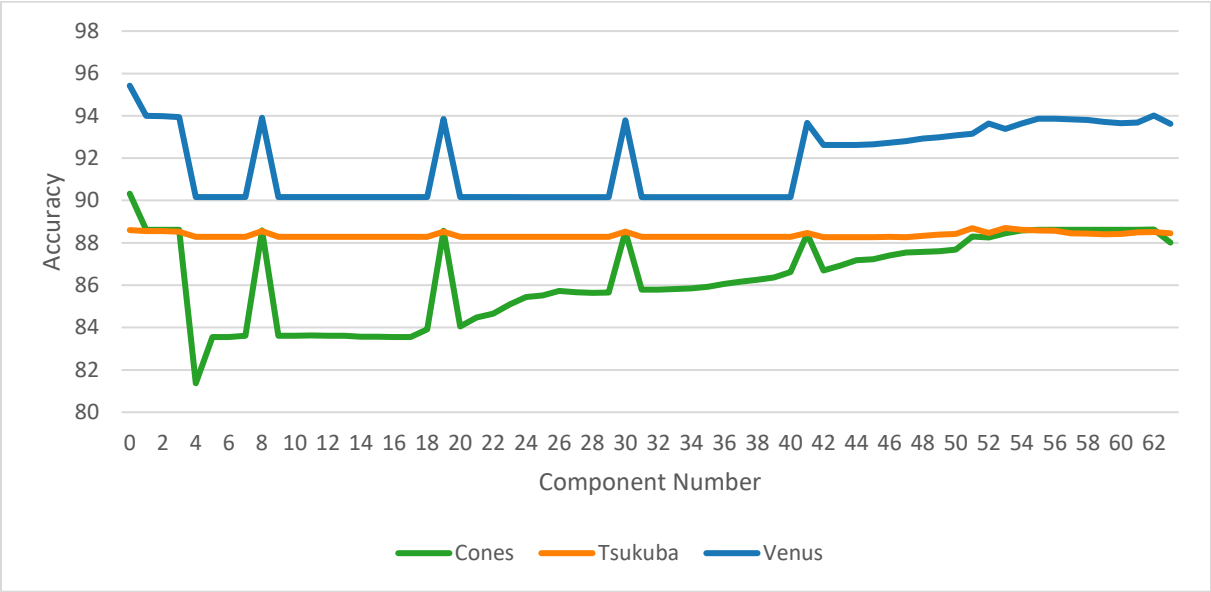

Figure 74 Accuracies for Cones, Tsukuba, and Venus images based on the number of comparators for SA1.

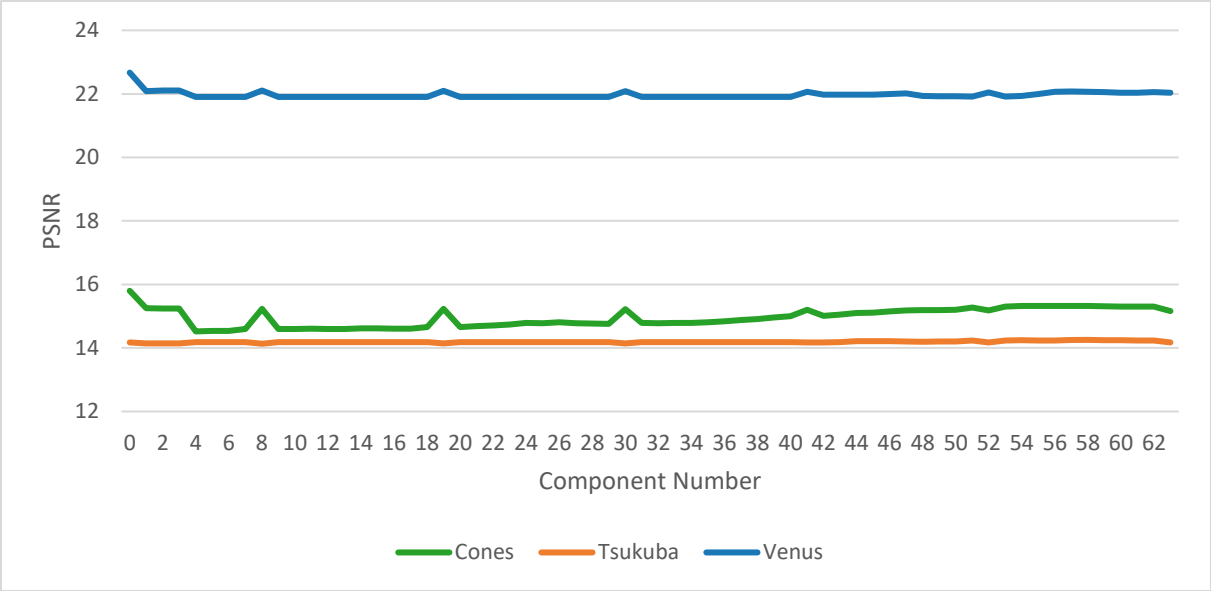

Figure 75 PSNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA1.

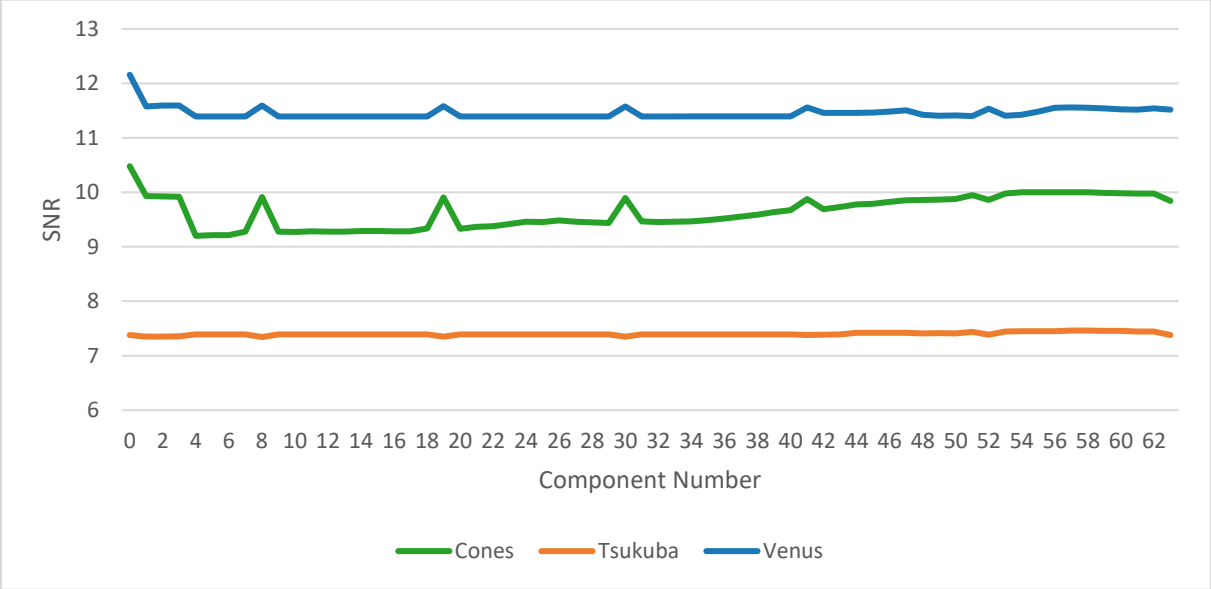

Figure 76 SNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA1.

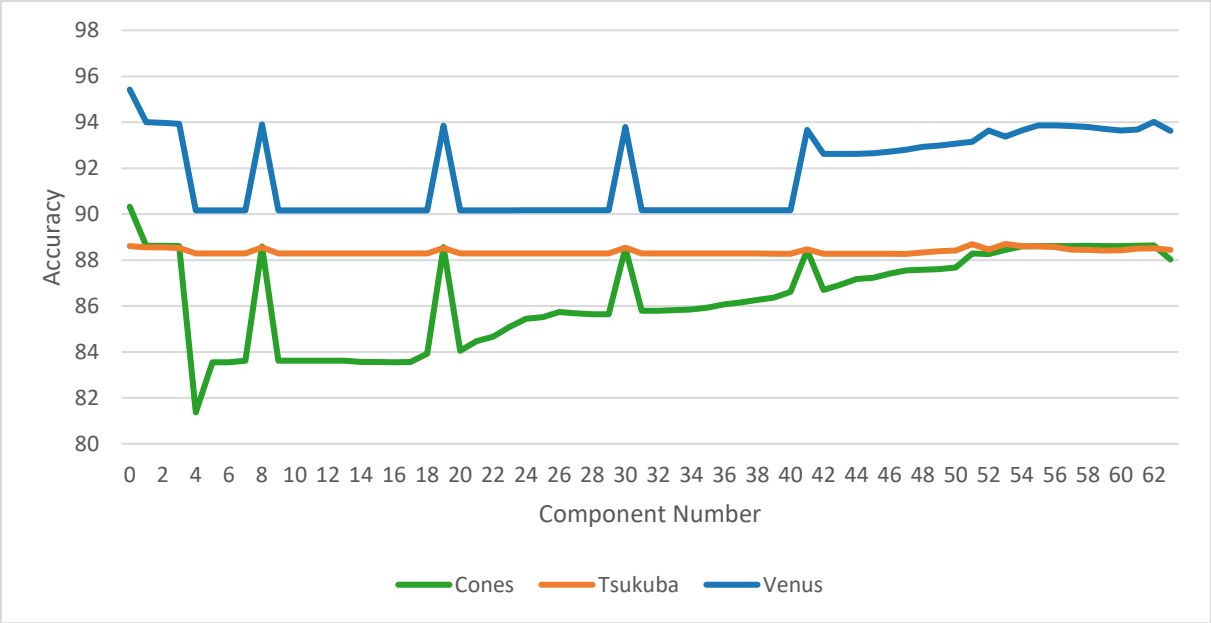

Figure 77 Accuracies for Cones, Tsukuba, and Venus images based on the number of comparators for SA0.

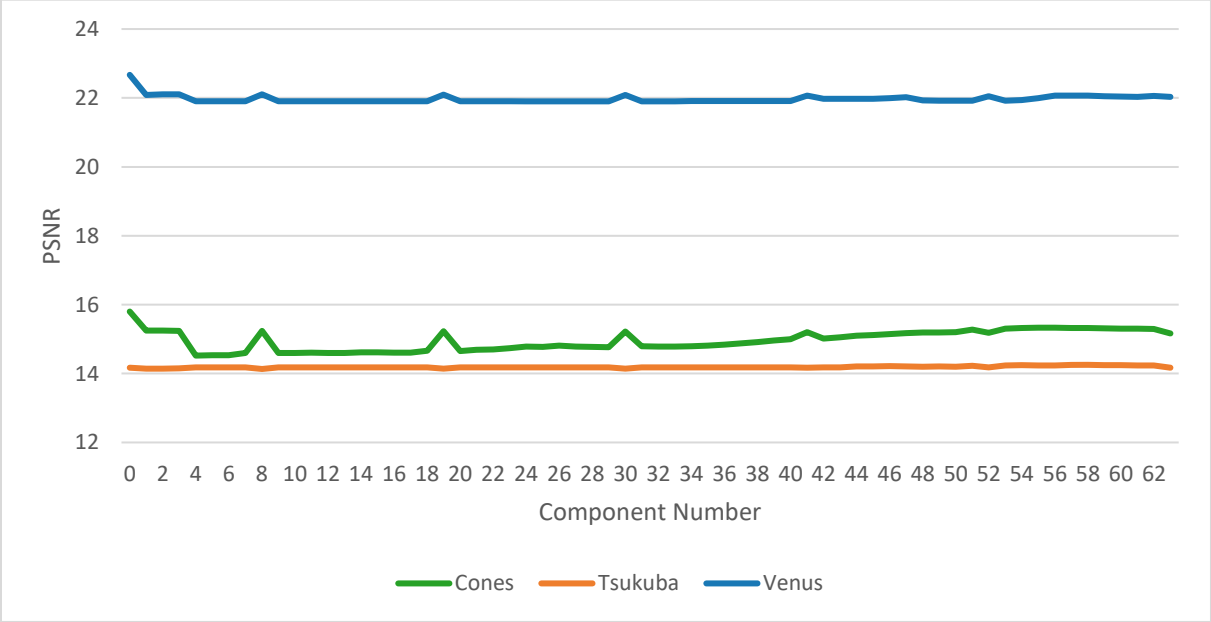

Figure 78 PSNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA0.

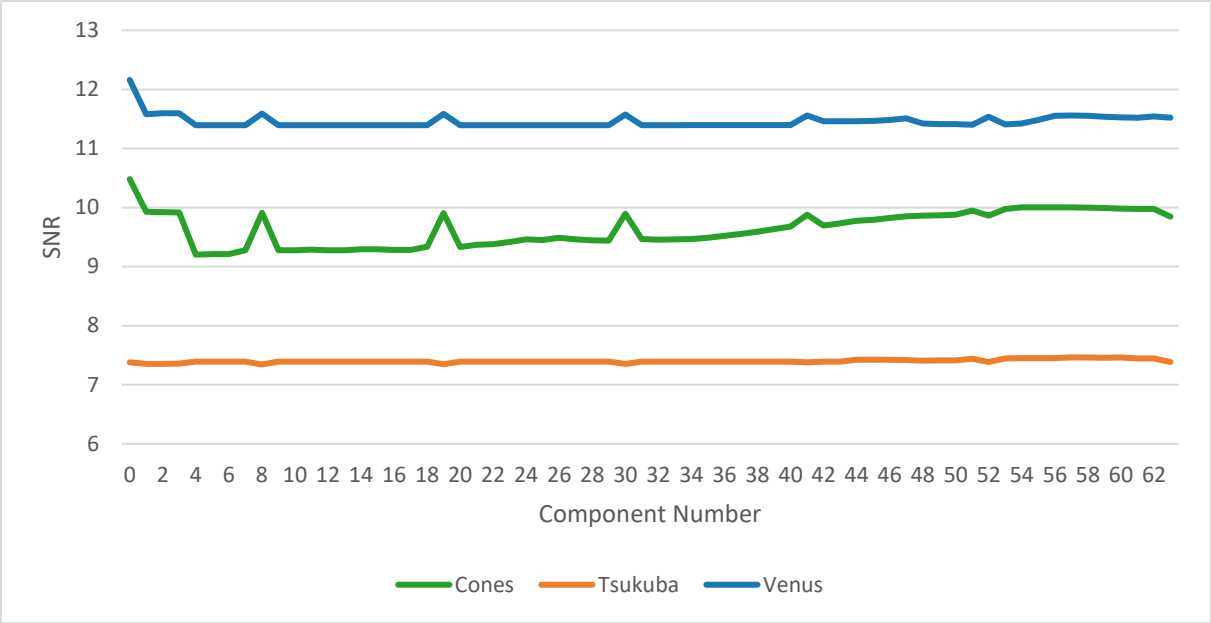

Figure 79 SNRs for Cones, Tsukuba, and Venus images based on the number of comparators for SA0.

Chapter 7

7. Conclusion and Future Work

Stereo vision accelerators have computationally heavy arithmetic units that can include sign operations. These accelerators can also be used in safety critical applications like autonomous cars. Therefore, their reliability is an important issue under abnormal conditions. The abnormalities may be caused by environmental effects or simply faults on physical electronics. The thesis investigates the effect of such errors and evaluate the reliability of the stereo vision accelerator based on census transform.

Simulation-based methods and analysis are implemented with Tsukuba, Cones, and Venus images. In chapter 6.1, fault-free simulation results are provided. The simulations show that the metrics vary according to images, window sizes, and neighborhood configurations. The accelerator, generally speaking, performs better with $7x7$ and $9x9$ window size, but there is not much difference between them. The results also indicate that more neighborhoods in the configurations can perform better in terms of the metrics. The variations, on the other hand, depend on the images.

The stuck-at faults are, then, injected for the reliability evaluation. The outcomes are presented in chapter 6.2. The average accuracy drop considering all images is 4.05%. A drop for PSNR and SNR is 1,91%. However, that amount of drop can be observed in the images in Figure 43. The observation is that when the fault appears in the upper half of the output signal, it is more likely to affect the accuracy, PSNR, and SNR values. If the signal is a part of a comparison unit, it is understandable. Because the magnitude comparison is influenced by the most significant bit position. Another result is that the accelerator can mitigate the adverse effect depending on the depth of the cascade architecture. For instance, the accelerator can partially recover the accuracy after every 11 consecutive connections of the comparator in Figure 74. The architecture also includes an *LRCC* module which enable double-check of the disparity levels. Despite the presence of the *LRCC* module, the effect of the fault can be seen in some cases.

The issue with the simulation-based method is the excess of time required. It also depends on the image size and the number of neighbors in the configuration. Considering that the processing time for the Tsukuba image for uniform design with a $7x7$ window size is 7

minutes in Questa, it becomes evident that implementing a faster method for fault injection would be more appropriate. The emulation-based method is the second option in the thesis. The controller module is developed and tested for an integration of the stereo vision accelerator in the FPGA.

7.1. Future Work

The future work covers an integration of the accelerator with the FPGA environment. However, the development of the controller facilitated progress in the subsequent phases. The future works include the integration of the accelerator with the stream data interface as in Figure 30. The aim is to establish a setup that runs all simulations and stores the results with just one command.

User needs to select fault type, location that fault is injected and then all statistics and images will be generated. By that way designers can analyze their designs under fault and adjust their designs as needed to ensure more secure operation.

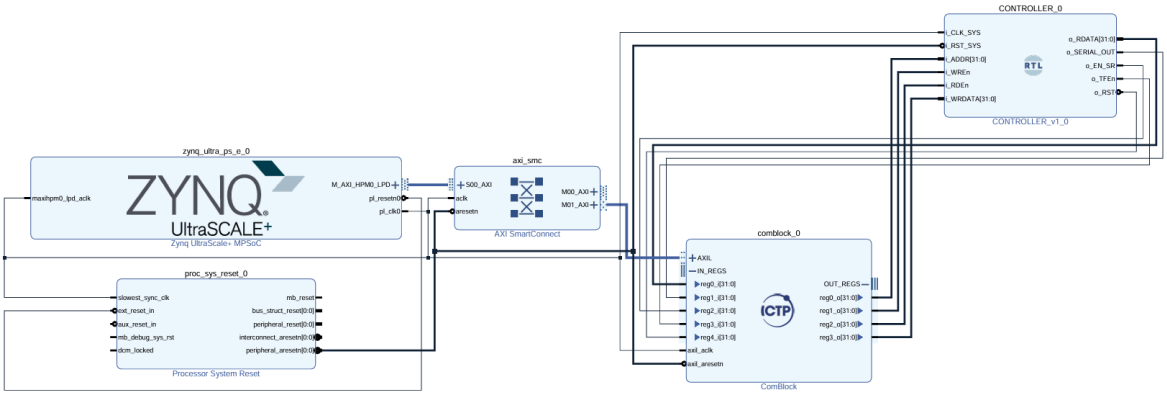

Figure 80 Example connection of the controller with the ComBlock

As shown in Figure 80, the controller has already been connected to the ComBlock. Here, ComBlock has 5 inputs and 4 output registers, each 32 bits. Four output pins are connected to the inputs of the Controller and outputs of the Controller are connected to the inputs of the ComBlock. Clock and reset signals are globally connected for both IPs. The next phase is to integrate the accelerator with the controller and then the stream interface.

In conclusion, this thesis focused on the reliability evaluation of the stereo vision accelerator. Stuck-at faults were injected and the statistics were gathered. The resilience of the

uniform design with a 7x7 window size was analyzed for the fault injection process. Besides the simulation-based method, the contribution to the emulation-based method is provided.

Bibliography

[1] W. S. Fife, "Improved Stereo Vision Methods for FPGA-Based Computing," Brigham Young University - Provo, 2011.

[2] R. Szeliski, Computer vision: Algorithms and applications, Springer Science & Business Media, 2010.

[3] "NVIDIA Jetson Nano System-on-Module," NVIDIA Corporation, 2014.

[4] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, no. 1, p. 13, 2004.

[5] B. Dipert, "How to build a custom embedded stereo system for depth perception," 10 August 2022. [Online]. Available: https://www.edge-ai-vision.com/2022/08/how-to-build-a-custom-embedded-stereo-system-for-depth-perception/.

[6] R. Klette, "Rectification of Stereo Image Pairs," in *Concise computer vision: An introduction into theory and algorithms*, Springer Science & Business Media, 2014, pp. 235-236.

[7] R. Klette, "Matching, Data Cost, and Confidence," in *Concise computer vision: An introduction into theory and algorithms*, Springer Science & Business Media, 2014, pp. 287-289.

[8] A. Garg, "Medium," 22 February 2022. [Online]. Available: https://medium.com/analytics-vidhya/distance-estimation-cf2f2fd709d8.

[9] D. Scharstein, P. Ugbabe and R. Szeliski, "2001 Stereo datasets with ground truth," 2011. [Online]. Available: https://vision.middlebury.edu/stereo/data/scenes2001/.

[10] S. K. Bukasa, L. Claudepierre, R. Lashermes and J.-L. Lanet, "When fault injection collides with hardware complexity," *Lecture Notes in Computer Science,* pp. 243-256, 2019.

[11] M. S. Reorda, "The stuck-at fault model," Politecnico di Torino, 2022.

[12] M. L. Bushnell and V. D. Agrawal, "A Glossary of Fault Models," in *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, Springer Science & Business Media, 2006, p. 66.

[13] M. L. Bushnell and V. D. Agrawal, "Faults," in *ESSENTIALS OF ELECTRONIC TESTING FOR DIGITAL, MEMORY AND MIXED-SIGNALVLSI CIRCUITS*, 2006, pp. 259-260.

[14] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001),* pp. 131-140.

[15] T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, D. DiSabello and M. C. Herbordt, "Achieving high performance with FPGA-based computing," *Computer,* vol. 40, no. 3, pp. 50-57, 2007.

[16] "GitLab," 2024. [Online]. Available: https://gitlab.com/ictp-mlab/hyperfpga-bsp.

[17] "GitLab," 2024. [Online]. Available: https://gitlab.com/ictp-mlab/core-comblock.

[18] S. Merrouche, M. Andric, B. Bondžulic and D. Bujakovic, "Objective image quality measures for disparity maps evaluation," *Electronics,* pp. 1-3, 2020.

[19] "GitHub," [Online]. Available: https://github.com/YosysHQ/yosys.

[20] L. A. Mesa, J. Guerrero-Balaguera, E. D. Castañeda, M. Sanchez and W. Pérez-Holguín, "An integrated environment for the reliability assessment of CNNs accelerators implemented in FPGAs," *2024 IEEE 25th Latin American Test Symposium (LATS),* 2024.