Cross-Layered Reliability Analysis of Object-Tracking Algorithms to Radiation-Induced Soft Errors

By

HAO QIU

Thesis
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
in

Electrical Engineering
May, 2017
Nashville, Tennessee

Approved:

Professor William H. Robinson

Professor Richard A. Peters

ACKNOWLEDGEMENTS

I would like to thank my parents for their unconditional love and devotion on me, they also sponsored me to pursue my graduate study. My family and friends also support and encourage me throughout the entire process. I would also like to thank Dr. William H. Robinson, my advisor, who guides me in last two years. He gives me helpful advice all along the progress of this thesis, helps me finish my first academic paper, revises the expressions and even fixes grammar errors in my work. Dr. Richard A. Peters also helped me a lot in his field. I would also specially thank Dr. Lloyd W. Massengill and Dr. Bharat L. Bhuva, their wonderful lectures enlightened me to enter the field of Integrated Circuits.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Object tracking is an essential part of computer vision and is used in various applications related to security and surveillance, such as autonomous navigation and intelligent transportation systems. object tracking can be defined as a problem of assigning consistent labels to the tracked object in different frames of a video. In practical applications, tracking objects can be complex due to: 1) noise in images, 2) none rigid or articulated nature of objects, 3) partial and full object occlusions, etc. [1] To overcome these problems, the state-of-the-art detection and tracking methods tend to be computationally intensive, such as the iterative process to find the closest neighbors in the mean-shift algorithm [2], or the matrix operations involving partial derivatives in an Extended Kalman Filter (EKF) [3]. To optimize the performance of these computationally complex algorithms in practical applications and process tracking problems in real time, hardware implementations, such as field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs) are more desirable than software ones [3], [4]. The FPGA-based EKF proposed in [3] had a 3X speedup over Pentium 1.6 GHz and 13X speedup over ARM920T 200MHz. However, such implementations will face reliability issues of modern electronics, such as radiation-induced soft errors.

When energetic particles passing through the reverse biased junctions within semiconductor devices, electron-hole pairs would be generated then the excess charge could be deposited in sensitive regions, then the node voltage might be flipped. The inverting of node

voltage can cause bit-flip in storage elements, which is defined as Single-event upsets (SEU). In combinational logic, the mechanism of charge collection may lead to Single-event transients (SETs). SETs can propagate through combinational logic or get latched by storage elements and develop into SEUs [5]. There are three masking factors affecting the propagation of SET:

1) logical masking: the propagation of transient faults depends on the active logic path from struck node. For example, when a transient pulse arrives gate whose output is already determined by other controlling inputs, this transient would be masked and cannot propagate any further.
2) electrical masking; if the pulse width of a transient pulse width is much smaller than the delay of the gates along the propagation path, the transient fault can be attenuate or masked completely.
3) latch-window masking: If a transient does not arrive at the input of the storage element (i.e., a flip-flop) on its triggering, the transient will not be latched and masked. Otherwise, the transient can develop into SEU.

As the device size keeps shrinking, the operating voltage levels get reduced but the clock speed increases, the impact of electrical masking and latch window masking is decreasing. Therefore, the reliability analysis of transient faults is also necessary for evaluating the vulnerability to soft errors. Furthermore, the chance of charge sharing among multiple devices increases, then single-event multiple transients/upsets can become another concern for sub-micron integrated circuits (ICs) [6], [7].

Currently, the research on the reliability of object-tracking algorithms mainly focused on the robustness to the behavior of the tracked object, such as such as the ambiguity problem in feature-based tracking, handling occlusions between objects in multiple-objects, re-detecting and

correct tracking if the target is lost, or matching the changing appearance patterns of the object [1], [8]. However, this thesis explores how the faults in ICs affect the accuracy of object-tracking algorithms. Generally, a hardware implementation of object-tracking contains the combinational logic modules corresponding to the arithmetic operations in each step of the algorithm, sequential logic such as finite state machine (FSM) is also applied to control the flow of the algorithm. SETs or SEU can affect the response of these modules, then these SET- or SEU-induced faults may propagate through the circuit to the outputs and eventually translate into observable erroneous tracking results, where the values become inaccurate. Therefore, this work discusses the reliability of object-tracking algorithms from the perspective of hardware. When evaluating the reliability of the high-level algorithm implementation on ICs, including analyzing the accuracy of the object tracking ASIC FPGA prototype, a cross-layered reliability analysis between circuit and algorithm level is necessary [9]. especially for the electronics operated in space and avionics. [10]

In this work, fault emulation is conducted on an FPGA prototype, which implemented blob-based trackers and an edge detection-based tracker. Transient and upset faults were injected into the data bus or internal registers either between or within the specific modules in this object-tracking hardware. The emulation results show that most of the injected faults can cause degradation on the tracking accuracy. Fault injection at certain modules may even lead to the failure of the tracking system. For example, multiple upset faults can cause an error rate of 88.87% in edge detection-based tracker on X direction. The applied tracking algorithms had different responses to the same injected faults, which indicates an algorithm could be more reliable under a certain type of fault. The type of the injected faults can also alternate the accuracy of the trackers, the system behaved drastically different to set-high or set-low faults.

The thesis is organized as follows. Chapter II gives an overview of the state-of-the-art object-tracking algorithms and related hardware implementations. This chapter also introduces the research on commonly applied fault injection methodologies in detail. Chapter III describes the architecture of the emulated object-tracking FPGA prototype, applied RTL-based fault injection method, and the design of fault emulation experiments. Chapter IV provides fault emulation results and analyzes the impact on tracking accuracy for radiation-induced faults injected into various locations. Chapter V summarizes the paper and suggests future directions for this work.

CHAPTER 2

Related Works

Overview of object-tracking algorithms

Object tracking systems are typically composed of two major parts: object detection and object tracking. The detector will extract an approximate region of the moving object, then it is the tracker's task to perform object correspondence from one frame to the next to generate the tracks. A description of tracked object is required, the object could be represented by their shapes and appearance models [1].

The object can be modeled as points, primitive shapes, contour or silhouette, etc. For example, A Kalman Filter models the object as a series of points and predicts the trajectory of the moving object by the observed measurements over time. Appearance presentations are crucial for feature-based tracking, apart from more traditional feature descriptor such as probability densities and templates. Advanced feature descriptors including gradient features; texture features; spatial-temporal features are developed [11].

According to the applied object presentation methods, the selected image features and the methods of context information integration. Numerous object tracking algorithms are proposed. As the state-of-art tracking methods are computationally intensive, many studies have also been done on how to implement object tracking algorithms in hardware architectures. Some researchers modified the tracking algorithms into the ones that more suitable for ICs. Because fixed-point arithmetic is preferred in hardware. For example, Choi, D, et al. gave a full-fixed point implementation of the Kalman Filter [12]. However, some other researchers worked on

improving the performance of object tracking algorithms by taking the advantage of hardware implementations.

Though hardware implementations of object tracking algorithms benefit from the hardware acceleration. The design of such hardware architectures can more challenging than the software ones because the software implementations are more effortless and flexible. A pure hardware implementation requires specifically designed modules, optimization of the available resources and satisfying the designed architecture with clocking requirements. At the meanwhile, a hardware implementation may also exploit the inherent parallelism of the tracking algorithms, the pipelines would not speed up the processing of pixels, but help reduce the usage of memory

D. B. K. Trieu, et al. presents an implementation of the mean shift tracking on FPGA, which can handle 768×512 images with the frame rate from 50 to 130 fps [2]. An FPGA-based architecture of EKF using 32bits single precision floating-point hardware is given by Bonato, V. et al. This design can process 3× more features than the software implementation (C language) in Pentium M 1.6GHz processor. And consumes only 1.3% of the processor power [3]. More complex object tracking hardware can be built. For example, R. Serrano-Gotarredona et al. presents a massive neuromorphic system CAVIAR, which is implemented with FPGAs and ASICs. The CAVIAR can mimic the behavior of the human nervous system. The full hardware design of it allows object recognition and tracking much faster than the software one [14].

However, this thesis focus on illustrating the impact of radiation-induced faults on object tracking algorithms. Two basic object detection algorithms were built and tested: motion-based and edge detection-based tracking. The track projectile is obtained via calculating the centroid of the possible target region.

Fault injection methods categorized by abstraction levels

The methodology for fault injection is an important aspect of the reliability analysis of ICs. Categorized by the abstraction level at which faults are injected, there are three major fault injection techniques [15]:

- Device-level fault injection,
- Gate-level fault injection, and
- RTL or higher level fault injection.

As shown in Fig.1, fault injection at lower level tends to fit the physics mechanism of radiation-induced faults better, but the payoff would be a rather more time and computational consuming. The appropriate level of fault injection should be selected depending on the concentration of reliability analysis.

The device-level fault injection tools accept a realistic representation of deposited charge as input for device simulations, which in turn is used to determine the circuit level response [16]. For example, R.D Schrimpf.et al. demonstrates a multi-scale simulation approach to examine SEEs. A virtual irradiator uses MRED (Monte Carlo Radiative Energy Deposition) [16] application produces realistic nuclear events; the event-rate of as circuit can also be estimated. These events are used as input into TCAD (Technology Computer Aided Design) simulations to determine the device and circuit response. [16]. Though developing an accurate physics model is difficult, or event not practical [16], [17], injecting faults at the device level fits the physics model best among the three levels. It is predictable that such simulation methods can be far too time intensive for large ICs.

An alternative way is gate-level fault injection. Based on the response of CMOS devices to particle strikes, transient pulses can be modeled as high or low pulses injected to the fan-out of

any logic gate to mimic the radiation-induced faults. There are many benchmark sets published based on the gate-level model, such as ISCAS'85, ISCAS'89, and ITC'99, etc. [18]. Hence the gate-level fault injection can be carried flexibly and conveniently if gate-level netlist is available. Gate-level fault models can also be utilized together with the layout information. As shown in B.T Kiddie's work, according to the placement information of logic cells obtained from layout, set-high or set-low faults can be injected into the fan-out of gates [19].

Considering the complexity of object-tracking hardware, the device-level or gate-level fault injection methods are not suitable for this work, because these techniques will require significantly more time for fault injection because of the enormous size of VLSI systems [20]. Injecting faults at RTL and higher levels is more feasible; for example, the results in [15] presented a 500× speedup of RTL-based fault injection than with a gate-level method.

Modeling VLSI using Hardware description languages (HDLs) at RTL is a typical practice in modern digital design [21]. When the RTL fault list of a module is treated as a representative fault sample of the collapsed gate-level stuck-at fault or bit-flip fault. RTL fault injection method will be a much more effective way to perform fault emulation/simulation to ICs. [20] If we approximate the fault model of RTL to higher abstraction levels, it is also possible to inject faults to higher levels such the architecture level, and even software fault injection. While coupling the high-level response with lower-level faults does not give a mapping of the realistic environment. Many studies are carried on RTL fault injection techniques to compare and analyze the discrepancy between higher level and lower level.

C.D Yount and D.P. Siewiorek developed an RTL fault model and emulated on a microprocessor, their experiments presented that over 97% of gate-level transient faults injected using gate model can be reproduced with RTL model [15]. P.A. Thaker, et.al proposed that

stratified fault sampling technique for VLSI, experimental results shown that the stratified RTL coverage of tested system was estimated to be within 0.6% of the gate-level coverage [20]. The comparison between RTL fault injection and traditional radiation-hardness assurance (RHA) tests was also carried. In fault injection verification of IBM POWER6 platform was performed by P.Kudva. et al. The very small difference of flip rate between RTL-based fault injection and Proton Irradiation test indicates fault injection can be an alternative way to validate circuit reliability. [22]

Fundamentals of Fault emulation algorithms

To design a fault injection test or campaign, there are four aspects need to be covered in general: 1) The fault model; 2) the fault injection locations for each fault; 3) the input vector set or test patterns; 4) the operational timing of the faults with respect to the execution of the design under test (DUT) [23]. Once the fault model is selected, the rest of factors will define the size of the test space. One can simplify the fault injection campaign by constraining the test space. For example, in this thesis, the input test vectors predefined by the input test video to object tracking system rather than an exhaustive input set, the timing of the faults is also set to be a constant in each experiment.

The basic fault injection algorithm in this work is shown in Figure.1. Following this algorithm, the fault detection method is used to determine whether the injected faults cause incorrect outputs. As shown in Figure.2, a fault-free golden copy is used along with a checker to detect the error outputs of DUT.

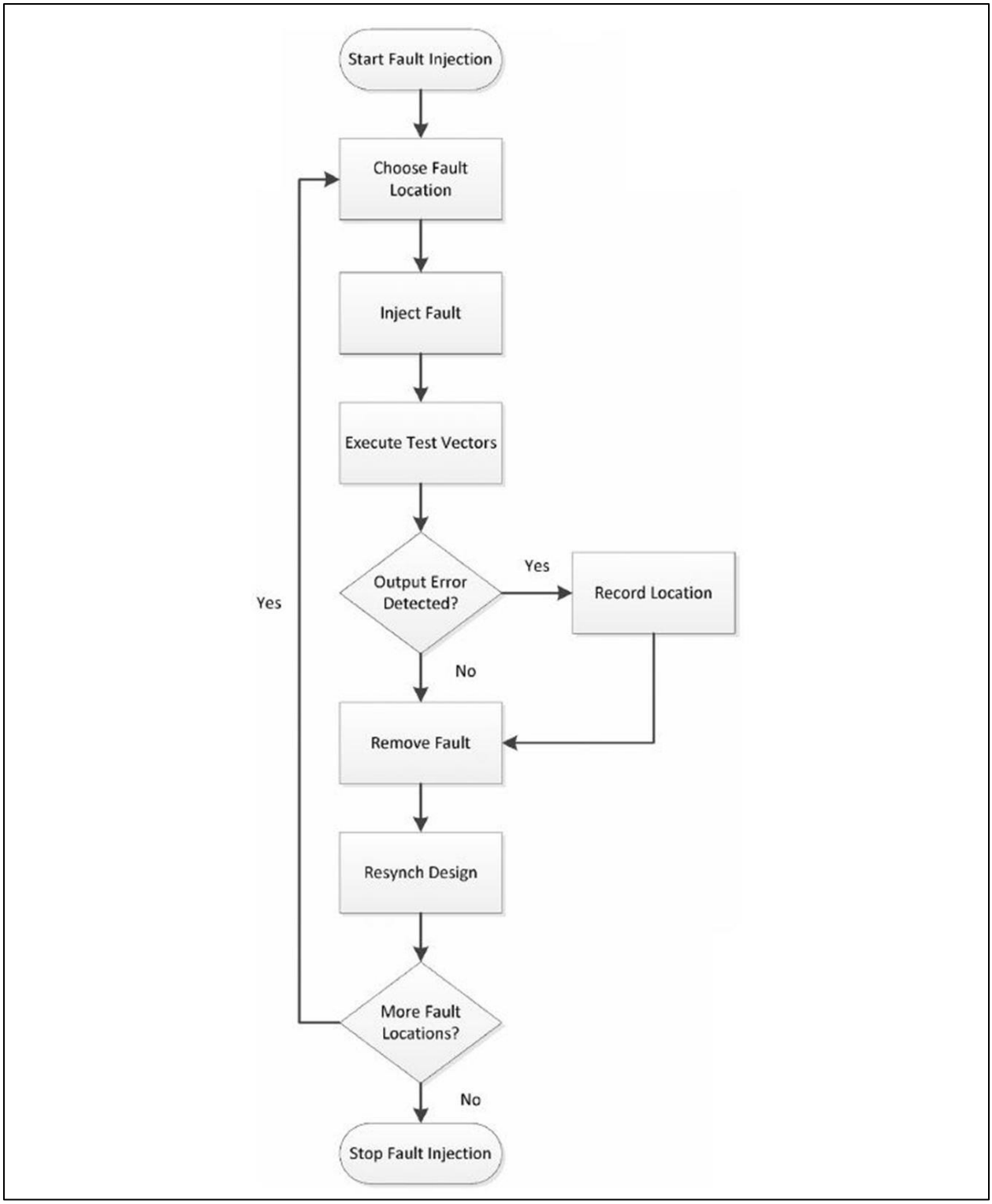

Figure 1: Flowchart of basic fault injection algorithm [23]

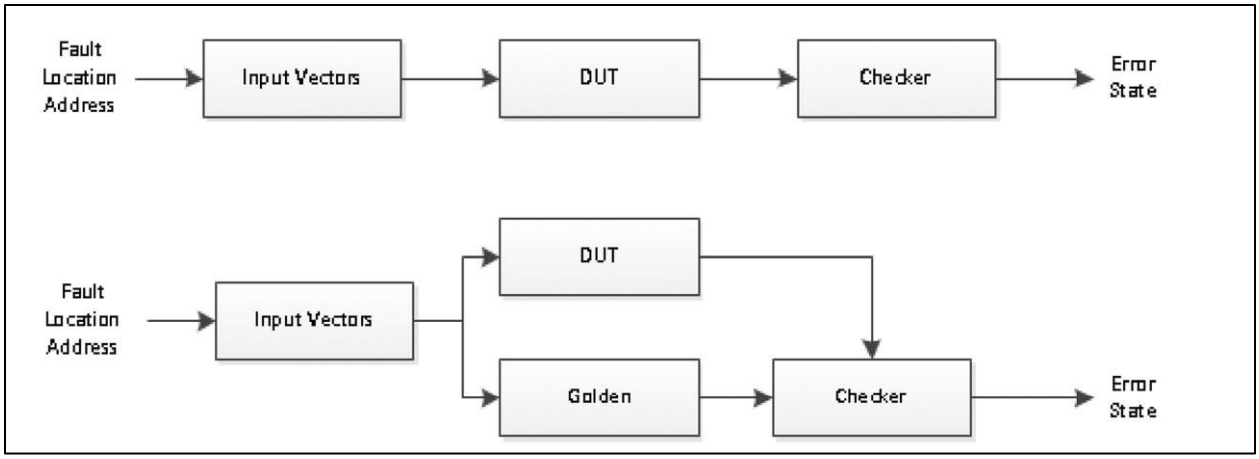

Figure 2: Flowchart Fault detection methods [23]

SRAM-based FPGA emulation methods

In this work, the implementation of object-tracking algorithm is prototyped in an SRAM-based FPGA board. For FPGAs, injecting faults into configuration memory is a common method. One of the most powerful tools to access configuration memory is the Joint Test Action Group (JTAG) boundary scan port, which is implemented by nearly all manufacturers as a standard. Most FPGA manufacturers also provide other tools to perform optimized fault emulation to their devices [24]. For example, a configuration RAM based Fault Injection IP Core allows evaluating system response for SEU is provided by Altera [25]. M. A. Aguirre, et al. present FT-UNSHADES, a fault injection tool on emulated hardware based on Xilinx Virtex FPGAs. The design uses partial reconfiguration techniques to provide fast fault injection emulation [26]. F. Ghaffari, et al give a fault injection scheme combining Partial Dynamic Reconfiguration and Isolation Design Flow, which enable faults can be injected into a specific region of DUT [27].

Another fault injection method for FPGAs is treating FPGA as a dummy device, faults are injected into the design prototyped on board rather than the internal structures such as

reconfiguration memory and Look Up Tables (LUT) of FPGAs. Shirazi and S.G. Miremadi presented an FPGA-based fault injection tool with good controllability and observability, the design is based on RTL fault injection, faults can be injected into any locations base on the description of DUT [28].

While many injection techniques for FPGAs are proposed based on dynamic partial reconfiguration [24], for our work, we restrict our analysis to the actual implementation of the object-tracking algorithm, instead of analyzing errors within the configuration bits of the FPGA, hence the method based on circuit structures with good controllability is preferred.

Overview of cross-layered reliability analysis

Radiation-induced faults at device level may propagate through hardware layer to the application layer and reach the system outputs [9]. Digital designers should also balance the growing reliability concern with other essential design aspects. However, the research of cross-layered reliability analysis is still immature compared to research that focuses on a single layer. The current practice is to rely either on more traditional fault injection campaigns or on simplified models, such as vulnerability factors. [29]

The architectural vulnerability factor (AVF) of a microarchitecture and the program vulnerability factor (PVF) of a program could give a coarse and conservative estimation of system reliability with a reasonable computational time. The Bayesian statistical models for reliability estimation is a novel method, but computing the conditional probability can also be difficult [29]. Gate-level fault injection is a traditional but time-consuming technique that can provide more precise reliability estimation.

Chapter 3

Methodology

Hardware implementation of object tracking algorithm

In this thesis, two basic object detection algorithms implementations were built and tested using FPGA. The object tracking algorithms are simplified based on the following constraints imposed.

- *Th*e camera is fixed or relatively stationary,
- The background will not change abruptly, and
- There is only one moving object to be tracked.

The tracking algorithms implemented here are a blob-based tracker and an edge detection-based tracker. As shown in Fig.5, the input image pixels I(x,y) are converted from RGB into Grayscale first, then I(x,y) would be subtracted by corresponding background image B(x,y). Once the object is extracted, the new image O(x,y) would be filtered by an approximated Gaussian Filter or Sobel Operators respectively. A centroid calculator is used to obtain the location (X or Y coordinate in an image) of the tracked object from filtered image. A full-hardware implementation of these algorithms is used to perform fault injection, the hardware design is coded in Verilog HDL and synthesized to Altera FPGAs.

A. Overview of the object-tracking FPGA prototype

The FPGA-based object tracking system has three main partitions: (1) Blob-based object tracking core modules (2) Edge detection-based object tracking core modules, and (3) the other modules that support video input and output, including clock generators, frame buffers, memory

controllers, VGA controller, etc. The block diagram of the high-level design is shown in Figure.3.

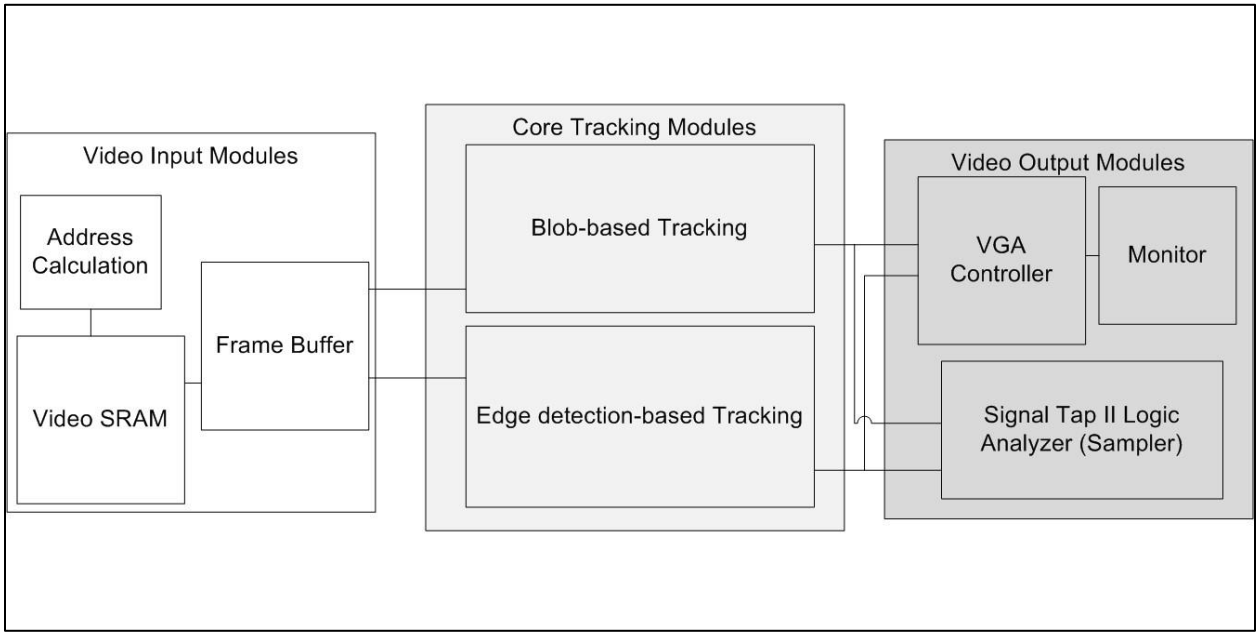

Figure 3: Block Diagram of FPGA prototype

The Video-input modules feed pixel values into the object detection modules and VGA controller, then the information of object locations obtained via core tracking and centroid calculation modules would be sent into Video-output modules. The original video and the cursor representing the object location can be drawn on the monitor via VGA DisplayPort.

B. Design of Supporting modules

In this work, core tracking algorithm modules are the main concern for reliability analysis, so the supporting modules, such will not be considered in fault emulation. But the design of important supporting modules would be introduced briefly in this section.

The VGA controller is configured to a display area of 640×480 with a frame rate of 60fps. Then the pixel rate can be obtained by multiplying the number of pixels in horizontal or vertical scan line and frame rate, which is 25.2M pixels/s (i.e. the pixel clock is 25.2MHz). Most of the sub-modules are synchronized with the pixel clock. Hence the 25.2MHz pixel clock is also the main system clock.

25.2 MHz main system clock of the object tracking FPGA prototype is generated by Altera Phase-Locked Loop (PLL) IP Core. Several manually scripted counters were implemented to generate clocks that cannot be obtained via PLLs such as frame clock; memory read/write clocks.by dividing the on-board 50MHz oscillator clock source [30].

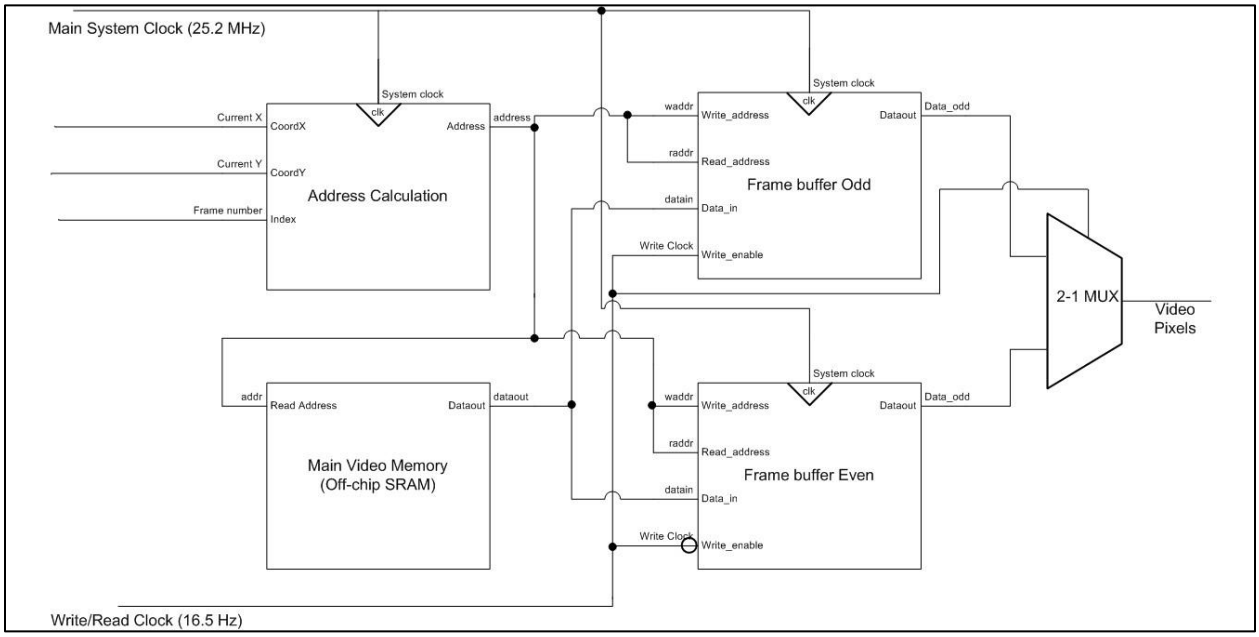

Figure 4: Schematic of Memory hierarchy for feeding video pixels

A simple memory hierarchy is built in this prototype, which includes the main video memory (on-board SRAM) [31] and the frame buffers (on-chip M4K blocks). The block diagram of video input memory hierarchy is shown in Figure.4. Because size of the on-chip RAM on DE2 board is rather small (483840 bits), the 512Kb SRAM is used as main memory to store the test video sequence, the video was converted into memory initialization file (Mif file) from standard video formats and write into SRAM at the beginning of emulation via a DE2 Control Panel facility. Two on-chip memory banks are implemented as frame buffers, which allow one memory bank to be accessed to display while the other one is busy with the write stage. Altera Embedded Memory IP Cores [32] are utilized to build frame buffers.

The address of this memory structure is a concatenation of address indicating current coordinates on VGA display and the index defining the current frame number. The address partitioning is shown in Figure. 5.

As shown in the address partitioning, the address is fed to main memory in full data width to take the specified frame, the lower part of the address is fed to frame buffers to draw images correctly via VGA. Because the write/read operations on frame buffer are interleaving, the same address can be connected to write/read address ports.

The video output modules are built based on the VGA controller module available at [33], the frame buffers output bank will feed pixels directly to it, then the test video can be displayed. Two different colored 4×4 squares cursors are added to the original module and used to indicate the results of implemented trackers, which are drawn over the displayed frames.

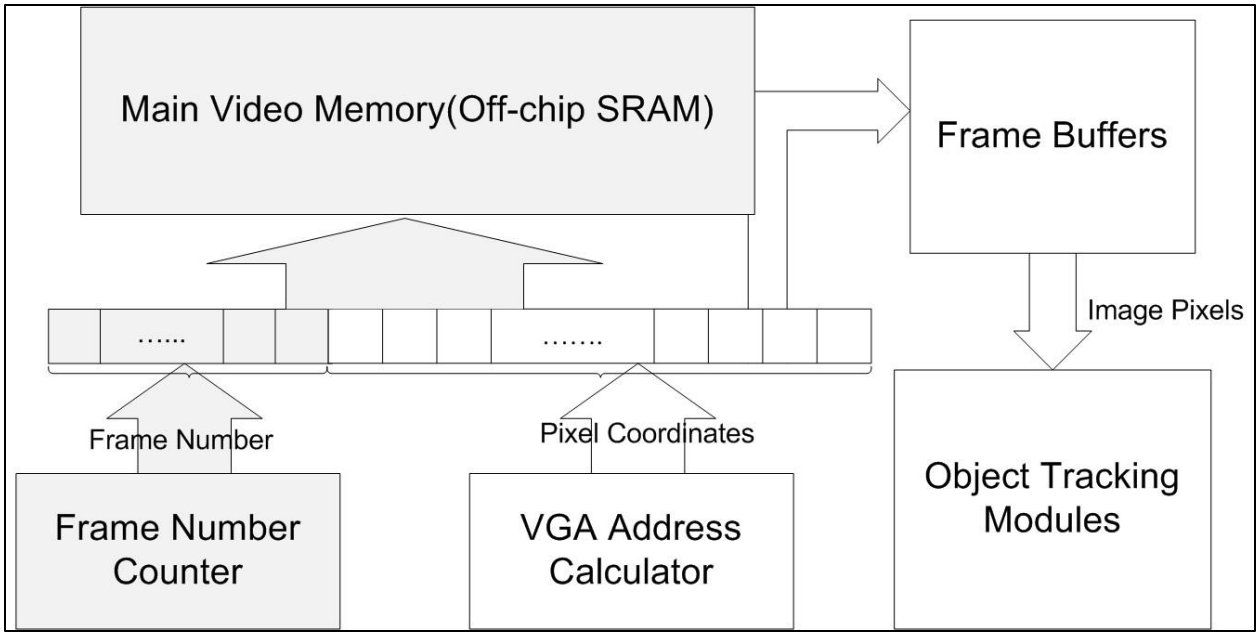

Figure 5: Partitioning and connecting of memory addresses

C. Core modules of object tracking FPGA prototype

The blob-based object detection system consists of five major components: (1) RGB to Grayscale (Rgb2gray) conversion; (2) Object extraction; (3) Noise Filtering; (4) Thresholding. Edge detection-based tracking shares the preprocessing modules with the blob-based tracker, but the pixels will be filtered by 3×3 Sobel operators.

a. Preprocessing Modules

1. RGB to Grayscale Conversion

Video frame pixels are converted from true color RGB to grayscale intensity values first, the standard NTSC RGB to Grayscale formula is applied here:

$$Intensity_{grayscale} = 0.2989 \times Red + 0.5870 \times Green + 0.1140 \times Blue \quad (1)$$

To avoid computationally intensive fractional number operations, Equation.1 is approximated by right shift (division by 2) registers and adders. Then RGB2Gray module is built based on Equation.2

$$Intensity_{approximated} = Red \gg 2 + Red \gg 5 + Green \gg 1 + Green \gg 4 + Blue \gg 4 + Blue \gg 5 \quad (2)$$

2. *Object extraction*

As shown in Equation.3, the coarse object region is extracted by taking the absolute value of the subtraction between the intensity values of current frame pixels and the predefined background image pixels.

$$Object\_image(x, y) = |Intensity\_currentframe(x, y) - Intensity\_background(x, y)| \quad (3)$$

A ROM built with Embedded Memory IP Core is used to store background information. Apart from the subtractor, a multiplexer (MUX) is employed to obtain the absolute value of the extracted object image, where the most significant bit (MSB) of the difference is the selector. The block diagram of preprocessing modules is shown in Figure. 6

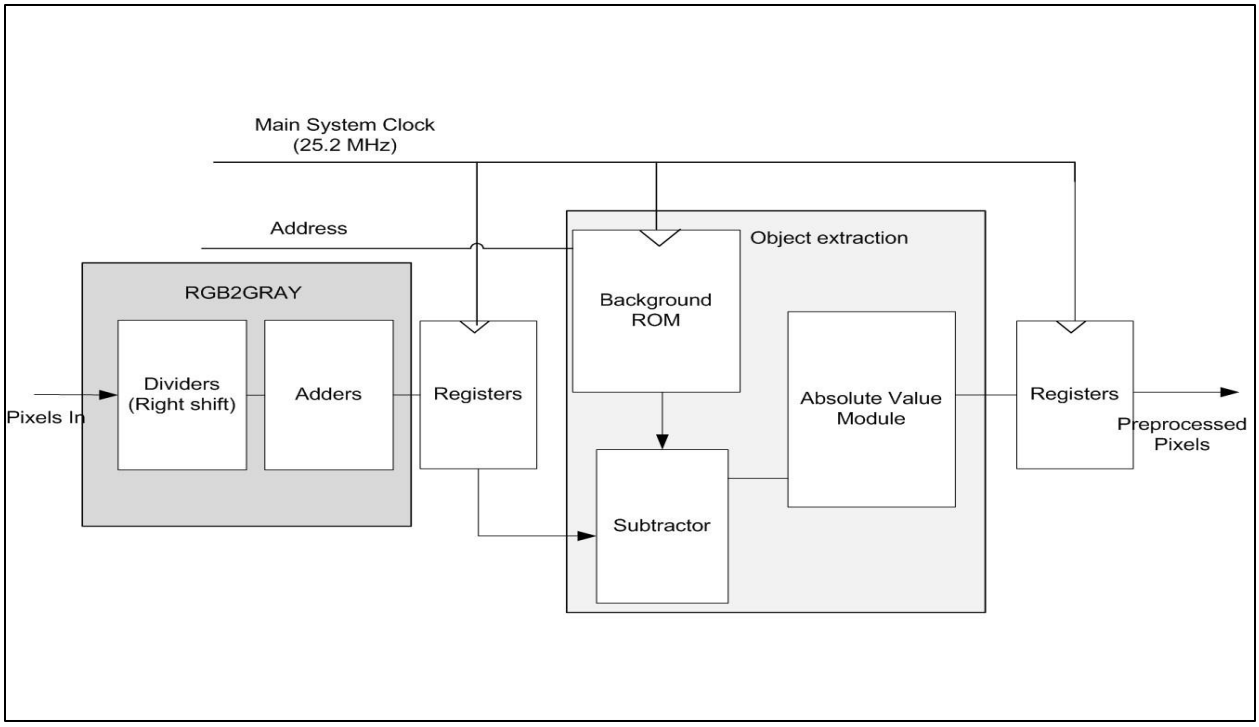

Figure 6: Block diagram of Preprocessing Modules (Rgb2Gray & Object extraction)

b. Spatial Filters

Extracted object image would be filtered by Gaussian Filter or Sobel Operators. In designed FPGA prototype, the video frame pixels are processed sequentially and individually in most of the modules. Hence to perform matrix convolution, apart from the filter modules, a data feeder is required.

1. Data feeder (3-line buffer)

A 3-line buffer built with Altera RAM-based Shift Register IP Core [34] is utilized to convert the flow of individual and sequential pixels into 3×3 matrices. The 3-line buffer is a large shift register holding the information of three entire lines of the VGA display. As shown in

Figure.7, three taps correspond to three elements in the same column of the “sliding window” are registered.

Because the active horizontal line length is 640 pixels, the distance between each tap of the shift register is set to be 640. Apart from the existing taps, six additional registers were deployed as buffers to store the image pixels from taps, a 3×3 matrix will be available. At each positive edge of the main system clock, the matrix is then fed into next module and convoluted with filter operators, which realizes the commonly seen sliding neighborhood operations in high-level languages.

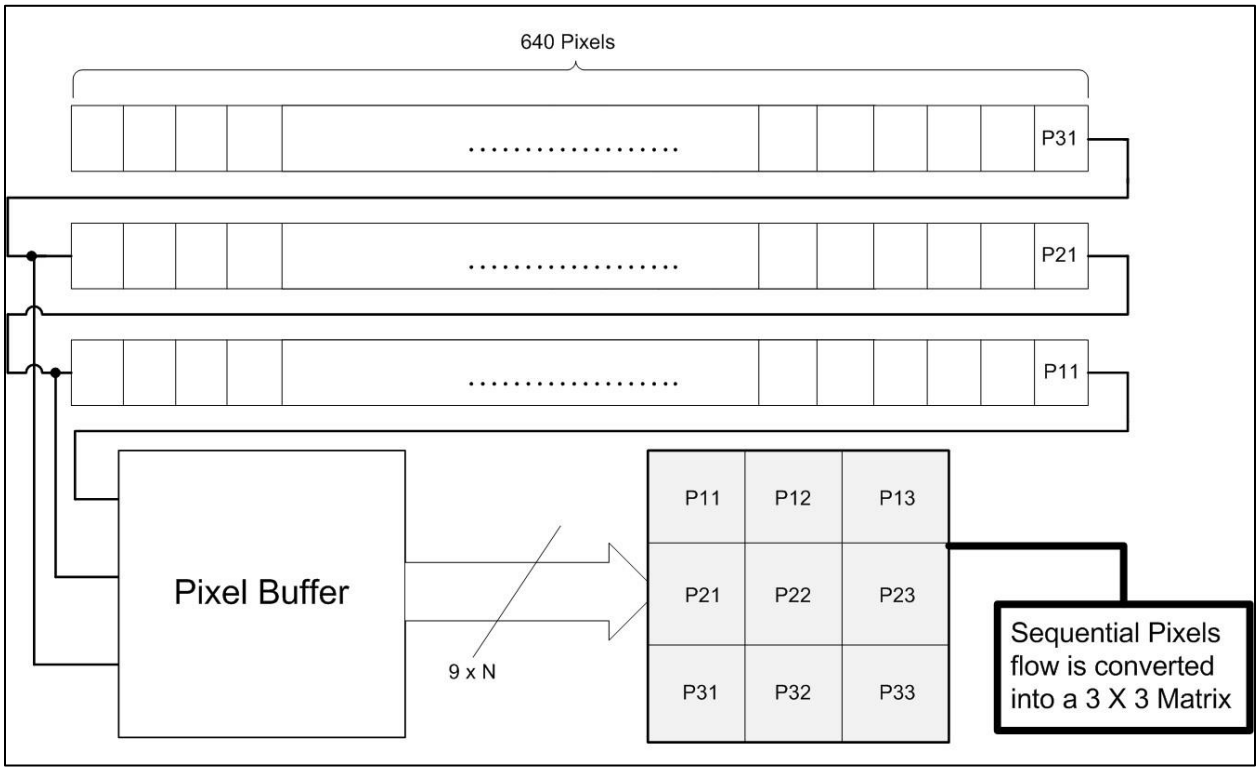

Figure 7: Schematic of Data feeder (3-line buffer)

2. Gaussian Filters

In blob-based tracking, a 2-D Gaussian Filter is applied to smooth the extracted object image. The Gaussian Filter deployed here is an approximated 3×3 Gaussian kernel (simplified into a mean filter), the formula of the filter is shown in Equation. 4:

$$G(x,y) = \begin{bmatrix} P11 & P12 & P13 \\ P23 & P22 & P23 \\ P31 & P32 & P33 \end{bmatrix} * \left(\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right) \quad (4)$$

A combinational logic circuit is built and the matrix convolution was broken into arithmetic operations to each element within the matrix. An adder will sum up these convolution results to obtain the intensity value of smoothed pixel. The schematic of Gaussian Filter is shown in Figure.8.

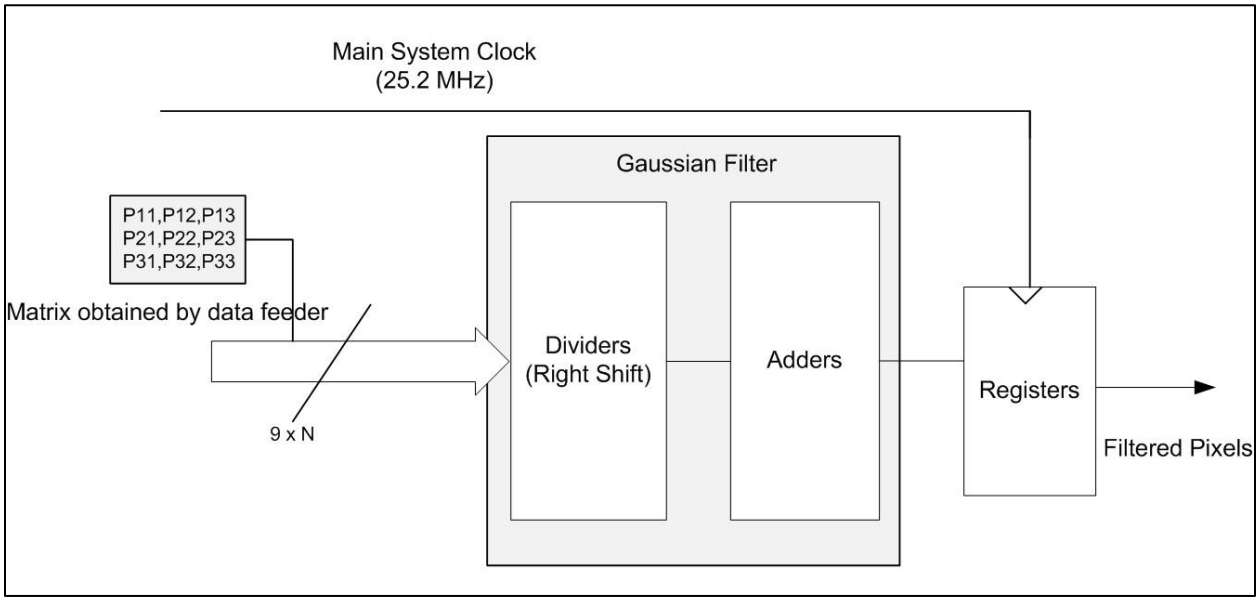

Figure 8: Block Diagram of Gaussian Filter Module

3. Sobel Operators

The Sobel Operator is applied in the edge detection-based tracker, it uses two 3×3 kernels to convolve with the extracted object image, one in the horizontal direction and another one in the vertical direction, respectively. The final intensity value calculation is simplified into an OR operation between horizontal and vertical results processed by threshold units. The edge detector computations are shown in Equation. 5:

$$S(x,y)_x = \begin{bmatrix} P11 & P12 & P13 \\ P23 & P22 & P23 \\ P31 & P32 & P33 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S(x,y)_y = \begin{bmatrix} P11 & P12 & P13 \\ P23 & P22 & P23 \\ P31 & P32 & P33 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (5)$$

$$Edge(x,y) = S(x,y)_x \mid S(x,y)_y$$

Like Gaussian Filter implementation, the matrix convolution is broken into arithmetic operations, the schematic of Sobel Filter is shown in Figure.9, Sobel operators also includes built-in threshold units, the edge could be detected via logic OR operation the convoluted results of horizontal and vertical operators.

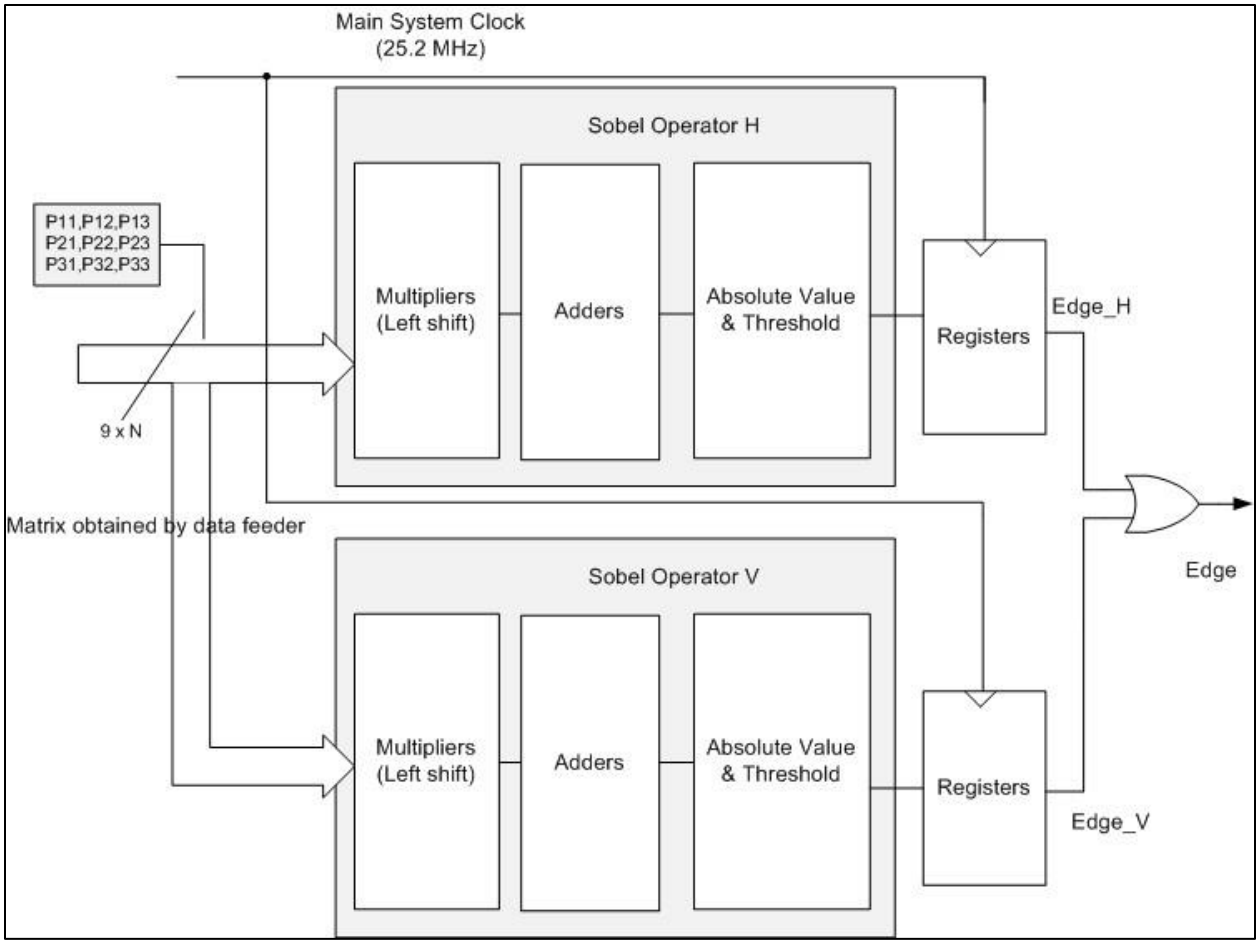

Figure 9: Block Diagram of Sobel Operators and built-in threshold modules

c. Threshold units

Threshold units are deployed at the output of each filter to check the values of filtered image pixels. If the intensity value of a pixel is greater than the threshold, the RGB value of this pixel will be set to all “binary 1s” (white). Otherwise, it will be “0” (black). The RGB pixels would be binarized using threshold units. According to the distribution shown in the grayscale histogram of the filtered test sequence, the threshold of Gaussian Filter is “0011111111”, which is compared with filter output via a comparator.

The threshold units for Sobel edge detectors are slightly different, because of the negative elements in kernels, the filtered intensity value can be smaller than zero. To avoid unnecessarily signed number arithmetic, the absolute values of the Sobel filter output are sent to threshold unit. Similar to the object extraction module, the absolute value is taken by a MUX whose selector is the MSB of incoming pixel. As the Sobel Operators approximate the derivatives, the thresholds are set to be "1" in both horizontal and vertical directions.

d. Centroid Calculation Module

The final output results of object tracking system can be obtained via centroid calculation modules, the X and Y coordinates of tracked object are fed into the cursor drawers of VGA controller.

The image pixels processed threshold are either be all "1s" or "0", where all "1s" pixels indicate the object region. Then the center of mass can be calculated by averaging the coordinates of leftmost and rightmost non-zero pixels for X axis, or top and bottom for Y axis. The algorithm implemented here is shown in Figure. 10.

However, the coordinates of center changes as new frames arrive, additional reset blocks are required. According to the timing of VGA, the current frame ends when vertical sync signal (Vsync) goes low and the new frame will be displayed at next active Vsync. Hence whenever the Vsync is "0", the intermediate values (leftmost, rightmost, top, bottom) are reset to the center of the display region. When Vsync is "1", the calculation of the centroid would be carried.

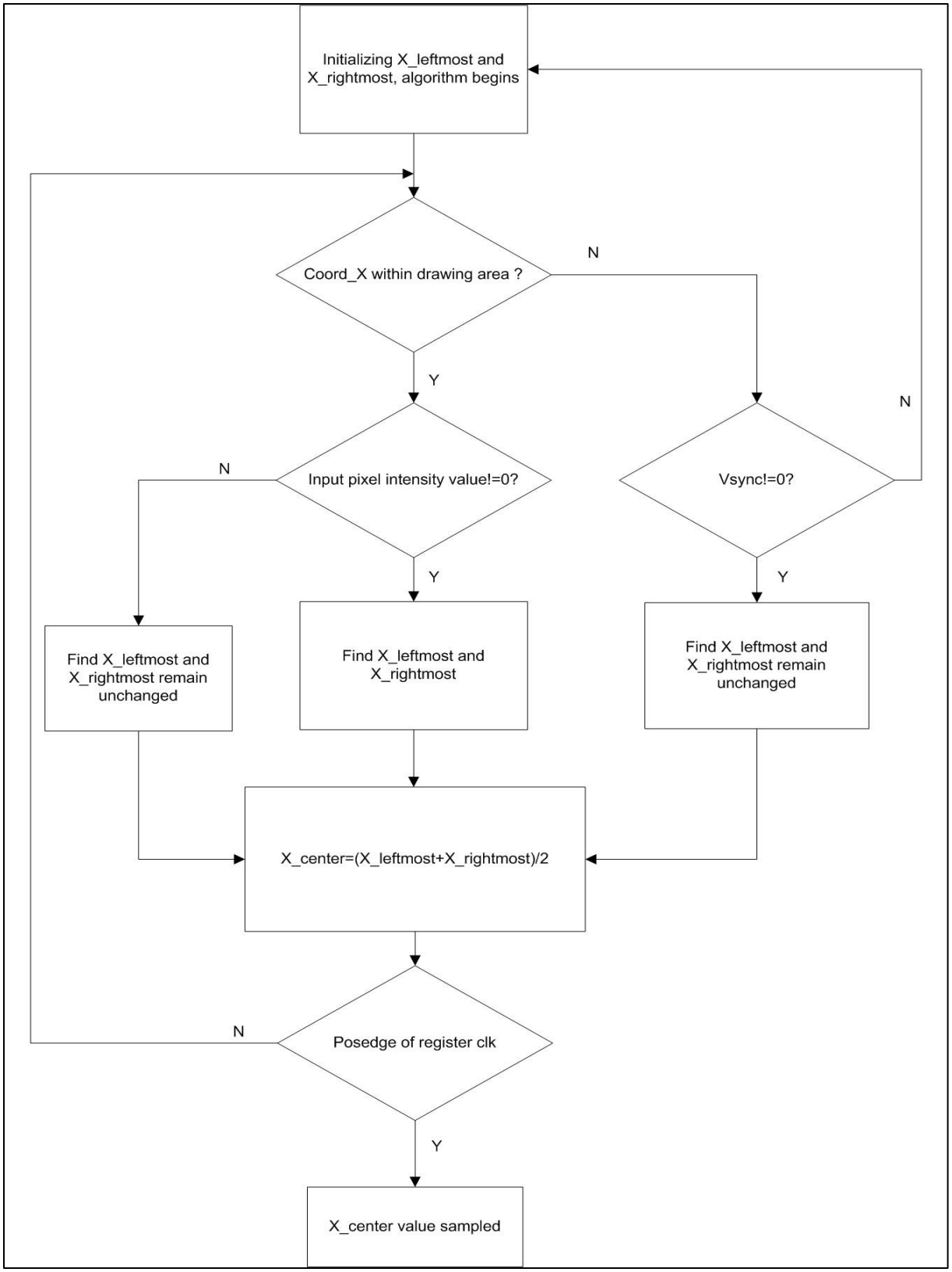

Figure 10: Flow chart of implemented centroid calculation algorithms

RTL-based fault injection on the FPGA prototype

As introduced in chapter II, fault emulation requires specified fault models, and a test space defined by test patterns, operation cycles, and fault locations. Because the test patterns are predefined by the video sequence stored in SRAM. Here we need to setup the faults models, faults locations and related hardware structures. The scheme of the fault injection on object tracking FPGA prototype is shown in Figure. 11.

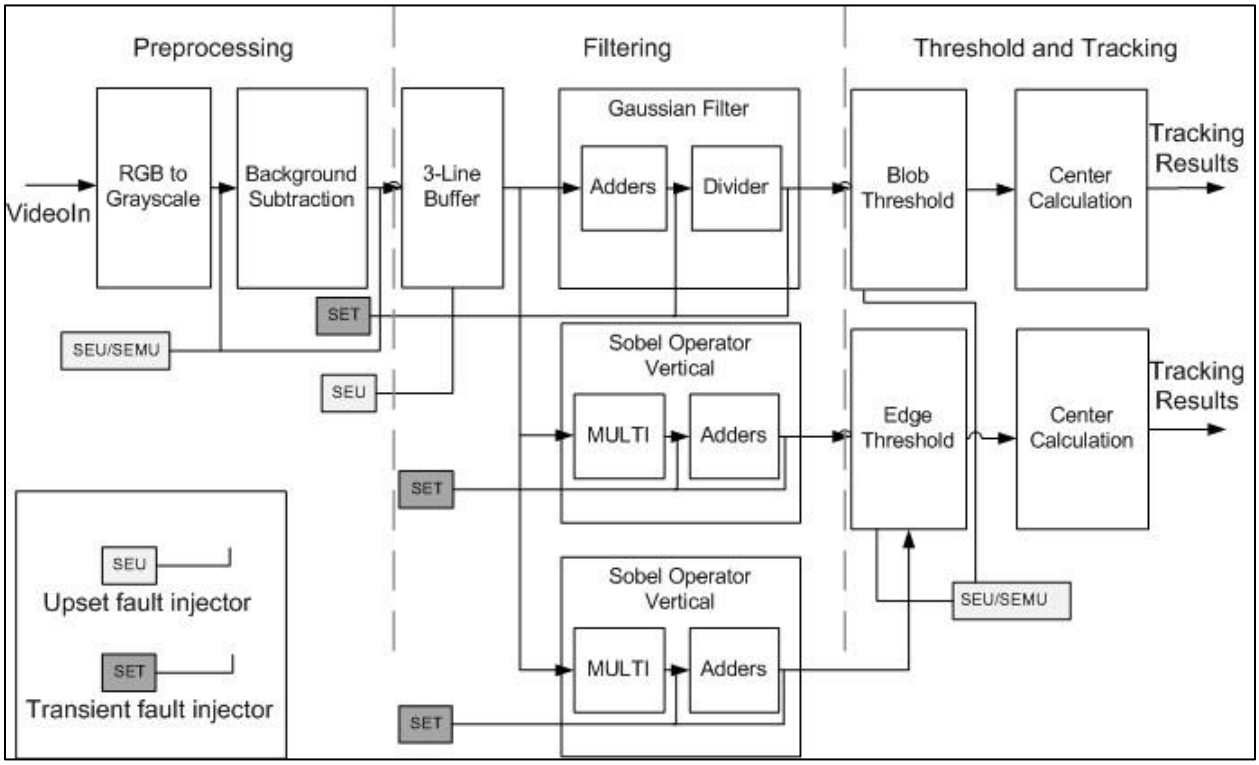

Figure 11: Overview of fault injection scheme

A. Fault models

Though there are many techniques for injecting faults into the specific locations of DUT. Two main types of faults are often applied in examining the SEEs induced faults: Set-at fault (set-high or set-low) and Bit-Flip fault. Set-at faults can model transients but bit-flip faults mimic upsets. Because the reversed biased p-n junctions tend to be more vulnerable to SEEs, for CMOS technologies, if the sensitive drain of the OFF NMOS transistor is struck by energetic particles(n-hit), a high-low-high pulse may appear at the output node, whereas for strikes at the OFF PMOS transistor may lead to a low-high-low pulse. Then set-high and set-low models can mimic the mechanisms of SETs more precisely.

In this work, these fault models are utilized and additional fault insertion lines and gates are connected to fault injected nodes. As shown in Fig.12, the structures of three types of faults injected are:

- Set-high fault: OR gate, another input is set to high;
- Set-low fault: AND gate, another input is set to low;
- Bit-flip fault: Inverter.

A 4-1 MUX is deployed at each fault injected node, a selector determines whether the node is fault-free or not and which type of fault is injected into the faulty node. The pulse width of set-high or set-low faults is set to be one frame clock period, then set-at faults develop into stuck-at faults here. Note that the transient faults were exaggerated in this work to analyze the impact of lower-level transient faults on system outputs.

Though the fault injection structure given here introduces significant area overhead to DUT, possibly changes the timing behavior of the high-speed circuits. Besides, if the pulse width of inserted transient might be masked by additional gates by the electrical masking. In this thesis, the area requirement is not a concern at all, and the fastest clock in this system is 25.2Mhz, the additional delay introduced by MUX and Gates is not large enough to cause synchronization problems on DUT. Because functional reliability test on the object tracking prototype, the pulse widths of transient faults are configured to be unrealistic ones, which last for one frame clock cycle. Hence the electrical masking will not be a trouble.

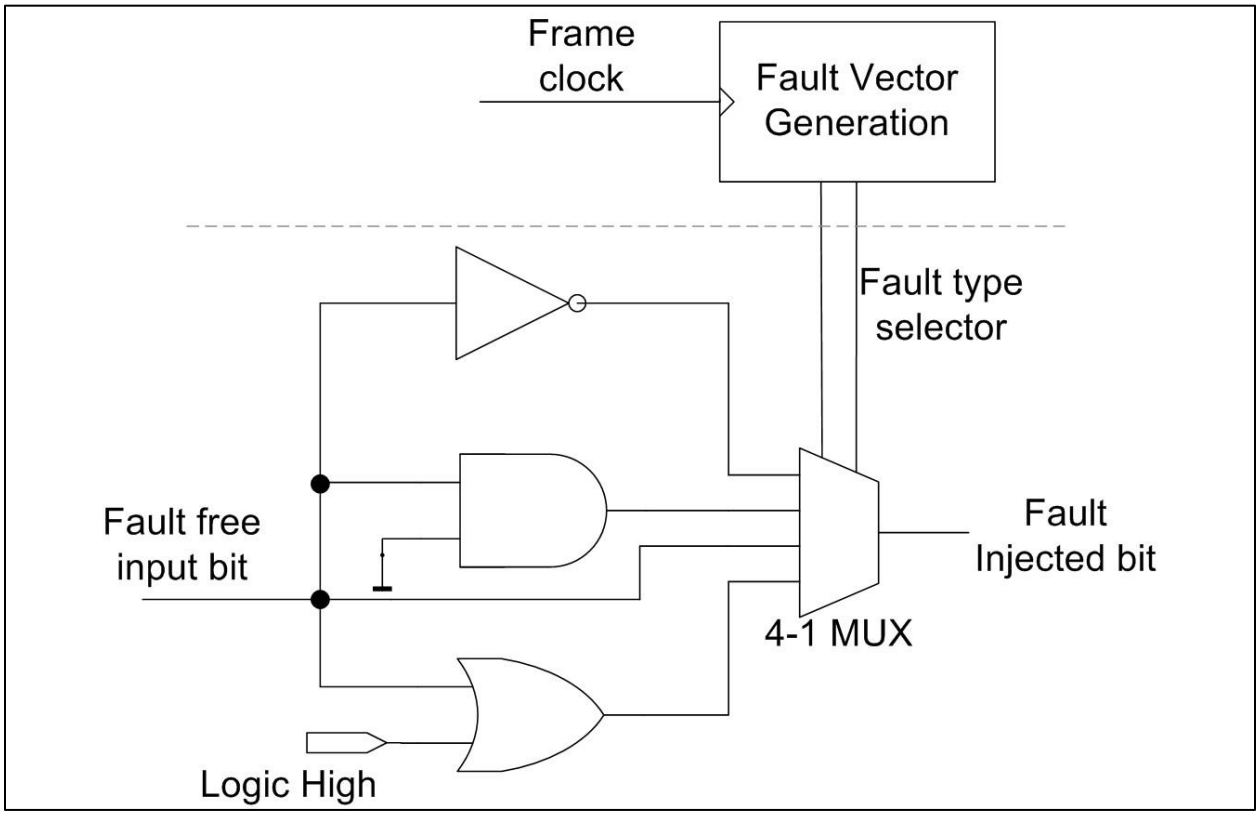

Figure 12. Schematic of applied fault injector

For the DUT with a strict area or timing requirements, the nodes to which transient faults are injected should be forced to VDD or GND voltage level, and the pulse width of the transients also need to be defined. In this way, the problems caused by extra inputs and related additional structures will not exist, but more efforts should be made in design such a fault injection scheme.

B. Fault locations

Fault injections are carried on core modules of the object tracking prototype, then the DUT can be abstracted to various stages according to the function of each main tracking algorithm module. Each module can also be further abstracted into more detailed RTL steps based on its Verilog description. Once the functional stages are set up, the data to which faults are injected can be fully localized.

By understanding the mechanisms of radiation-induced faults, a certain module should be injected with its corresponding faults. Usually, SET faults should be injected into combinational logic functional stages, while SEUs would be injected into registers or memories. The stages into which faults are injected into are listed in Table. I

Table I. Fault injection locations and corresponding fault models

Tracking Algorithm	Main Module	Functional Stage	Fault Model
Blob-based Tracking	Rgb2Gray	Output Registers	SEU/SEMU
	Object Extraction	Output Registers	SEU/SEMU
		Output Registers	Random Upsets
		Internal Subtractor	SET
	3-Line Buffer	Output Registers	SEU/SEMU
	Gaussian Filter	Internal Adder	SET/SETH/SETL
		Internal Divider	SET/SETH/SETL
	Threshold	Threshold Registers	SEU/SEMU
Edge detection-based Tracking	Rgb2Gray	Output Registers	SEU/SEMU
	Object Extraction	Output Registers	SEU/SEMU
		Output Registers	Random Upsets
		Internal Subtractor	SET
	3-Line Buffer	Output Registers	SEU/SEMU
	Vertical/Horizontal Sobel Operators	Internal Adders	SET/SETH/SETL
		Internal Multipliers	SET/SETH/SETL
	Threshold	Threshold Registers	SEU/SEMU

Though there are combinational logic blocks in RGB2Gray and object extraction modules, only the upsets fault at the output registers of these preprocessing modules are considered. The 3-line buffer (data feeder) are a series of registers and SEU would be the only concern. For the spatial filters modules, the arithmetic blocks are the issues, then transient faults are injected.

C. Fault vector generation and related structures

To perform massive fault emulation, additional modules should be designed to control the types and locations of the faults. Considering the applied fault model, the faulty bits should also be specified by fault injection controller. The controller is actually a generator that feeds fault injectors with a series of vectors (i.e. fault vectors) whose bits indicate the necessary information of faults to be injected. The fault vector is concatenated by a vector representing the fault injected module (fault location), and another vector indicating the injected bits (faulty bits) together with the type of fault. The partitioning of fault vector and fault injector scheme is shown in Figure. 13.

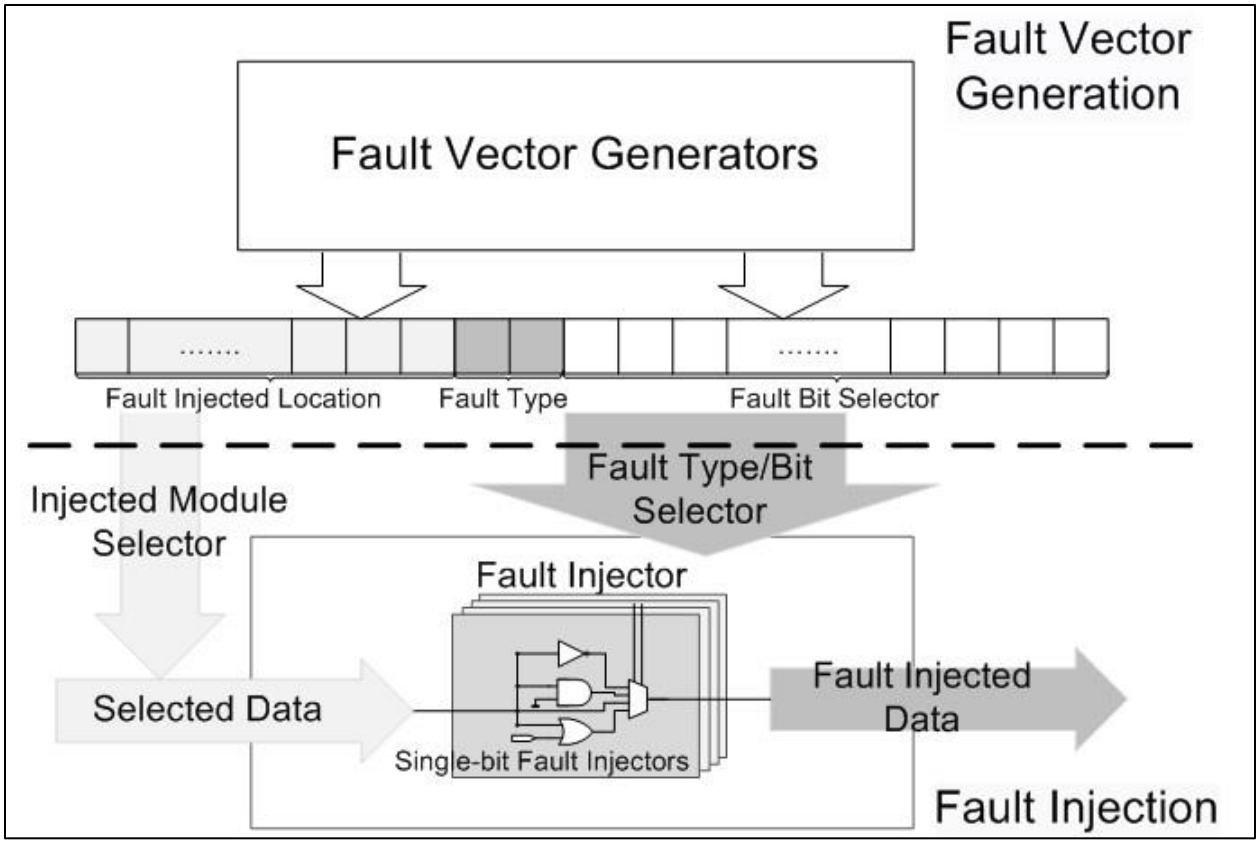

Figure 13: Partitioning of fault vectors and fault injector scheme

A fault vector generator is deployed to feed fault vectors into the fault injector. Here we only inject a certain type of fault at one functional stage in one frame, and the fault will not recover until the frame ends. As a new frame comes, a new fault will be injected as soon as the current fault is removed. Hence the clock of the fault injection vector generation is determined by the framerate of test video sequence. Then, the fault vector generator can be simplified into a pseudo-random number generator synchronized with frame clock.

The transient faults appeared at the fan-out of combination logic are determined by various of factors. For example, if a particle strike causes a low-high-low transient at the output of a gate within a multiplier, this transient may propagate through the circuit and lead to

multiple-bit transients at the fan-out of the multiplier. Hence the SEE induced transient faults end to distribute randomly among the fan-out data bits, which can be intimated by a linear-feedback shift register (LFSR) [35]. A Galois LFSR with a predefined seed is built, the LFSR generates an exhaustive set of fault vectors, an N-bit LFSR with a maximum-length polynomial can cover all possible N-bit fault vectors.

Bit-flip faults are more likely to happen on one bit or adjacent bits in the registers, then a shift register containing consecutive "1s", which indicates the faulty bits, can be used to control injection of the upset faults. For example, given a 10-bit register, 2-bit SEMU can be inserted by initializing the content of the shift register to be "1100000000", which indicates the 2-bit upsets appear at physical adjacent flip-flops (i.e. flip-flops D[9] and D[8]) currently. The emulation results show that a pseudo-randomness can be achieved via shifting the fault vector.

D. Fault Emulation and Reliability Evaluation

In fault emulation, experiments were carried on the DUT with fault injection hardware. A golden copy of the design is used as a reference to check errors [23]. The object tracking prototype and fault injectors were synchronized with the main system clock. During each experiment, a test video was sent into the DUT and played iteratively, and fault vectors were automatically generated and fed into the fault injector. The fault vector generator was synchronized with the frame clock, which enabled that a certain type of fault is injected into a specific location in each frame.

On the output side, Altera Signal Tap Logic Analyzer [36] is used to monitor the tracker outputs: Coordinate X and Coordinate Y of both the blob-based tracking and the edge detection tracking results. As the tracking results were sampled at selected triggers and logged in a list file.

Here the trigger is set to be the positive edge of the frame clock. Then, the contents stored in the list file were compared with ones in the golden copy, and the next experiment would be executed.

A threshold is set to determine whether the difference between the output of DUT and its counterpart in the golden reference is enough to be an error. Here the radius of the tracked object is known, the value of the threshold can be set to be a specified proportion of the radius

In this work, the reliability of the DUT is evaluated by determining the error rate of the tracking results, which can be obtained by:

$$\text{Error rate} = \frac{\sum \text{number of observable errors}}{\sum \text{number of fault emulation sampling points}} \quad (6)$$

A secondary metric is also provided if the discrepancy between error rates induced by different faults is trivial. The geometric mean of tracking results error (unit: pixels) is defined by:

$$\text{Average Error} = \sqrt[\#\,observable\,faults]{\prod_{i=1}^{n}(|Coord\_\text{gold}_\text{i} - Coord\_error_i| - Threshold)} \quad (7)$$

Where Coord_gold represents the values of the tracking result of gold copy; Coord_error means the values of the erroneous tracking result (defined as observable errors) of the DUT.

Once the log lists of the tracking results of DUT and Gold copy are obtained, the evaluation process is conducted with Java scripts or other high-level programming languages based on the proposed metrics.

CHAPTER 4

Experiment and Result Analysis

Experiment configurations

The fault emulation experiments were carried on Altera DE2 development and education board, equipped with a Cyclone II EP2C35F672C6 FPGA. The object-tracking algorithms implemented with the FPGA prototype design were used as a benchmark for cross-layer evaluation. The design was described in Verilog and synthesized in Altera Quartus II 13.0 sp1 software. A flow chart of Fault Emulation Experiment is shown in Figure. 14.

A 19-frame, 80×45 resolution, 12-bit (4-4-4) RGB video is used as the test sequence in the experiments. As introduced in Chapter III, the main system clock (i.e. VGA pixel clock) is 25.2 MHz, and the frame rate is 33 fps (i.e., a 33 Hz frame clock). The test video is played iteratively throughout the fault emulation.

The display area on the monitor is set to a size of 80×45, if the coordinate of the pixel is out of the region, the RGB value of these pixels would be set to all "0s" (black). The frame number partition in the address is obtained via counter counts from 0 to 18, which is synchronized with 33Hz frame clock. The pixel location partition of the address is calculated by:

$$Address_{Pixel\ location} = Coord_X + Coord_Y \times 80 \quad (8)$$

Where $Coord_X$ and $Coord_Y$ indicating the locations of current pixel are acquired from VGA controller. Because the width of test video is 80 pixels, the address will be increased by 80 when VGA is scanning the next line.

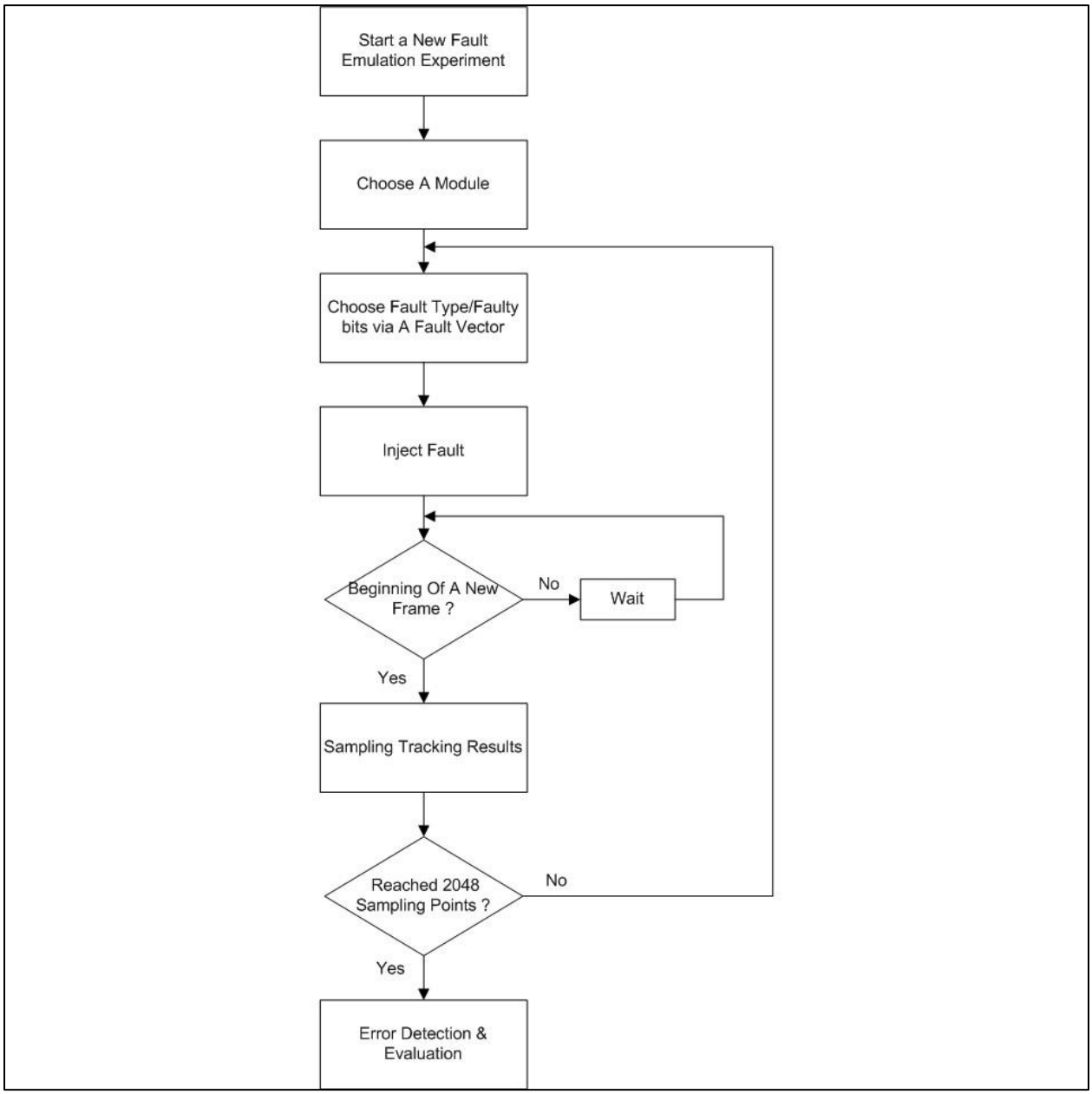

Figure 14: Flow chart of fault emulation experiments

The tracked object moves both in horizontal and vertical directions. To avoid abrupt object movement at the beginning of next video loop, the projectile of the tracked object is set to be an “infinite-looped sinuous wave”. Then the object seemed like to be moving iteratively across the screen. The raw video and two markers are displayed on the monitor via VGA controller, which are exploited to show the tracking results as an illustration of how the faults affect the tracking accuracy.

During each experiment, we inject a specified type of faults into a data bus or registers within a certain functional stage. Because the video pixels are stored as 12-bit RGB values in Mif file, and the width of grayscale intensity value is 10 bits. Then the data widths of most of the fault injection locations were 12 bits or 10 bits. For the Gaussian Filter, the multipliers and dividers introduced a data width of 13 bits due to bandwidth expansion.

The fault models used mainly in the fault emulation experiments are randomly distributed transients (SET), set-high transients(SETH), set-low transients(SETL), bit-flip (SEU) and two-bit-flip (SEMU). SET model intimates a more practical transient faults distribution at the fan-out of the combination logic block. SETH or SETL models assumes the transient faults are all set high or low rather than randomly spread across the bus, the results of these fault models can be a guideline to hardness design. The two-bit-flip model assumes the upsets will happen on adjacent bits because of the spatial proximity of storage cells in common register designs. SEU/SEMU faults were controlled by shift registers. Besides, the fault model that upset faults randomly scattered among output registers (SEMU Random) is also applied as a reference.

For fault vector generation of transient faults, a 11-bit LFSR with a period of 2048 is applied to generate fault vectors of transient faults. The maximal-length polynomial applied here is:

$$Polynomial\ for\ 11\ bit\ LFSR = x^{11} + x^{9} + 1 \qquad (9)$$

The corresponding Galois LFSR to the polynomial in Equation. 9 can be built as the schematic shown in Figure.15, the seed of the LFSR is a binary number “11111111111”.

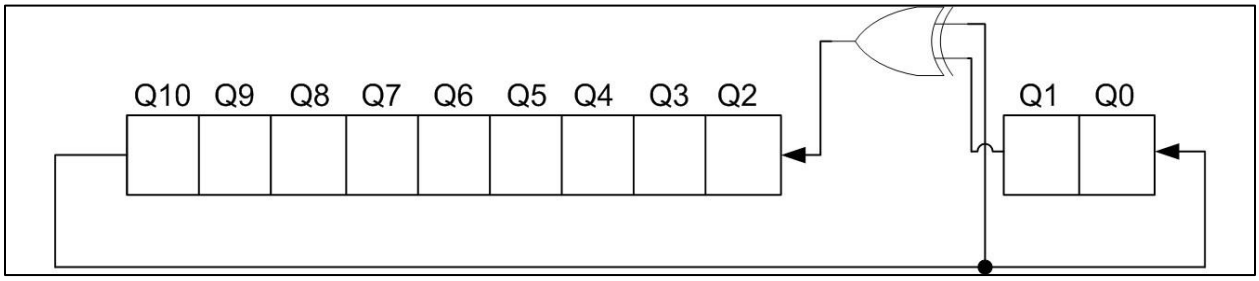

Figure 15: An implementation of 11-bit LFSR based on Equation.9

The higher 10 bits of the LFSR are fed into 10-bit fault injectors, the least significant bit controls whether the transient fault is set-high or set-low. Regarding the 12-bit or 13-bit fault injection buses, several randomly picked bits within the output of LFSR are fed to extra bits then 12 or 13-bit pseudo random sequence can be obtained.

The tracking results of the DUT and the golden copy were sampled by Altera Signal Tap II Logic Analyzer at the positive edge of the 33 Hz frame clock. The number of sampling points is 4,096 in each experiment. The values of monitored pins were logged in the Signal Tap II list files. Because the radius of the object is 5 pixels in the test sequence and the error threshold is set

to be 60% of the radius, i.e., 3 pixels. The error rate and the average error can be calculated via Equation. 1 and Equation. 3, respectively, which is conducted by JavaScript in this thesis. Then, the accuracy of tracking results can be evaluated by comparing the contents of these files. Hence a cross-layered reliability analysis between hardware implementations and algorithms can be carried.

Experimental Results

Based on the fault injection scheme shown in Figure.1, fault emulation was performed on the DUT. The primary metric of cross-layered reliability evaluation is the error rate. The average tracking error can be utilized as a reference. Because the centroid calculation modules are located close to the final outputs, it is predictable that these modules can cause great degradation at the tracking results. Therefore, the preprocessing modules and the spatial filters within core tracking modules are tested here.

A. Resources Consumption

Table II. FPGA Resources (EP2C35F672C6) Consumption Summary

Design Type	Total logic elements	Total Combinational Functions	Dedicated logic registers
Tracking hardware	1,934/33,216 (6%)	1,410/33,216 (4%)	1,172/33,216 (4%)
Tracking hardware with fault injector structures	2,612/33,216 (8%)	2,185/33,216 (7%)	1,212/33,216 (4%)

The object tracking algorithms implemented on FPGA are relatively fundamental and less than 10% logic elements are used, hence the FPGA resources are adequate here. The fault injector and its supporting modules introduced a 33% increase of total resources consumed. As shown in Table II, the fault injection structures did not cause serious overhead problems to FPGA in this design.

B. Numerical Analysis

a. Preprocessing units

The processed tracking error rates of SEU/SEMU fault injection for preprocessing modules are shown in Figure 16,17. For the upset faults based on adjacency assumption employed here, the edge detection-based tracker was more vulnerable to the upsets than the blob-based one, the error rates of edge detection tracking results on both X and Y directions are larger than the blob-based tracker. When SEU faults are injected into the background subtraction module, the error rates of edge tracker and blob-based tracker were 65.67% vs 17.82% in the X direction and 56.44% vs 9.814% in the Y direction. The SEMU faults of Rgb2Gray module did not influence the tracking accuracy greatly and the error rates stayed nearly constant or even lower. Whereas the SEMU faults at background subtraction module had a significant effect on tracking results. The SEMU error rate in the Y direction of the blob-based tracker can be 7x higher than the SEU one. and the SEMU error rate in X direction of edge detection-based tracker was 88.87%. An error rate of 88.87% means the functional failure of the object tracking system.

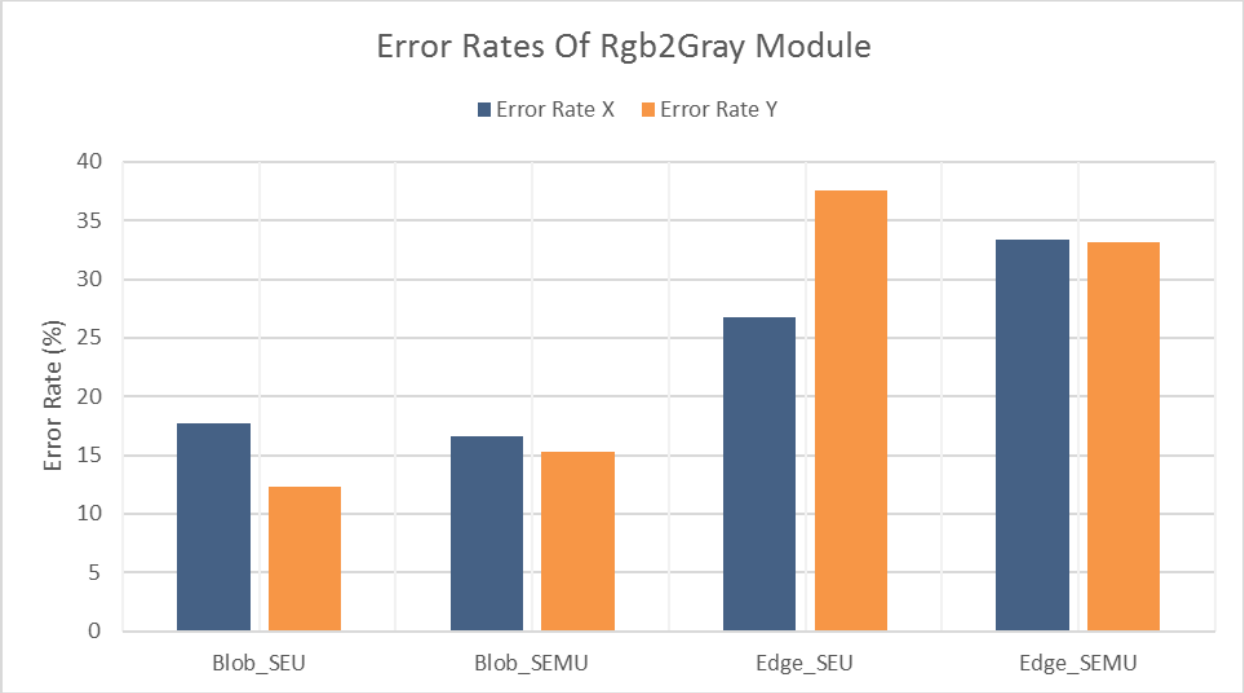

Figure 16: Error rates induced by SEU/SEMU faults in Rgb2Gray module. Edge based tracker was vulnerable to upset faults

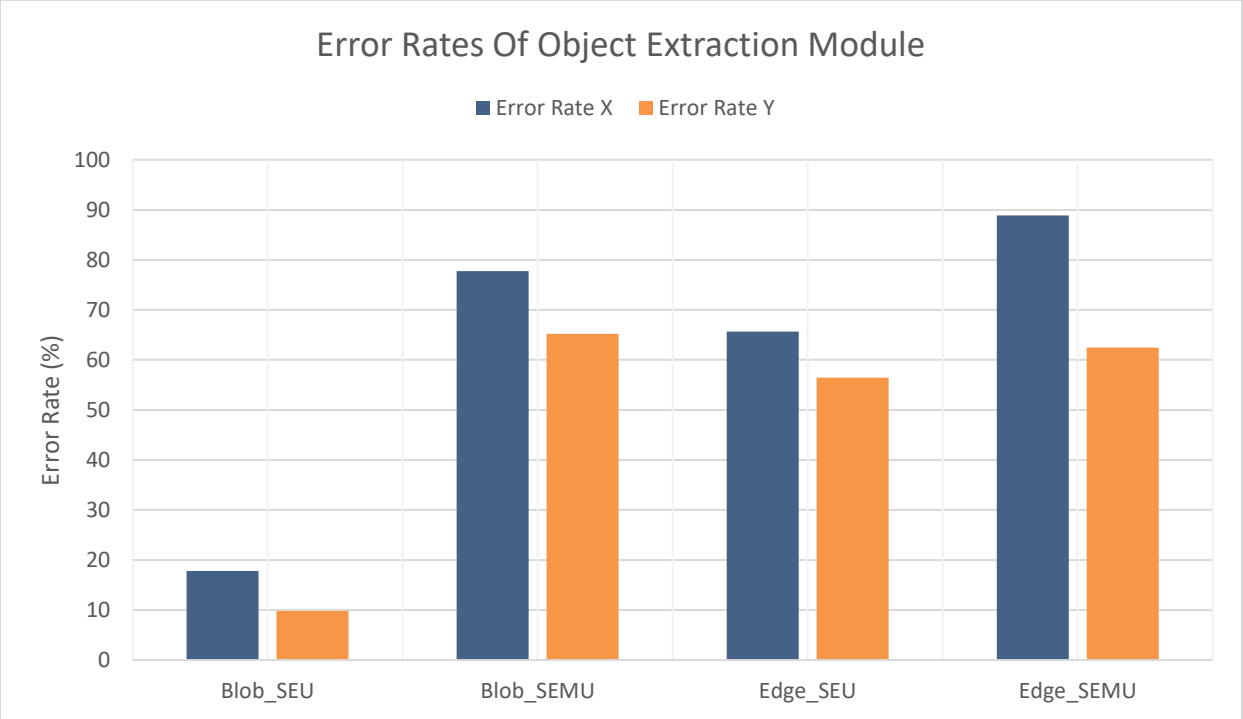

Figure 17: Error rates induced by SEU/SEMU faults in Object Extraction module. Edge based tracker was more vulnerable to upset faults and multiple upsets affected blob tracker significantly

Randomly scattered transient faults were also injected into fan-out of combination logic within the preprocessing units as a comparison. The error rates were compared with randomly distributed upsets rather than the adjacent upsets. As shown in Figure. 18. The transient faults in Object extraction module could not map to random upset faults, which was beyond the expectation. However, the error rates induced by transient faults helped still partially supported the assumption that the higher-level faults can be treated as a collapsed fault list of lower-level faults.

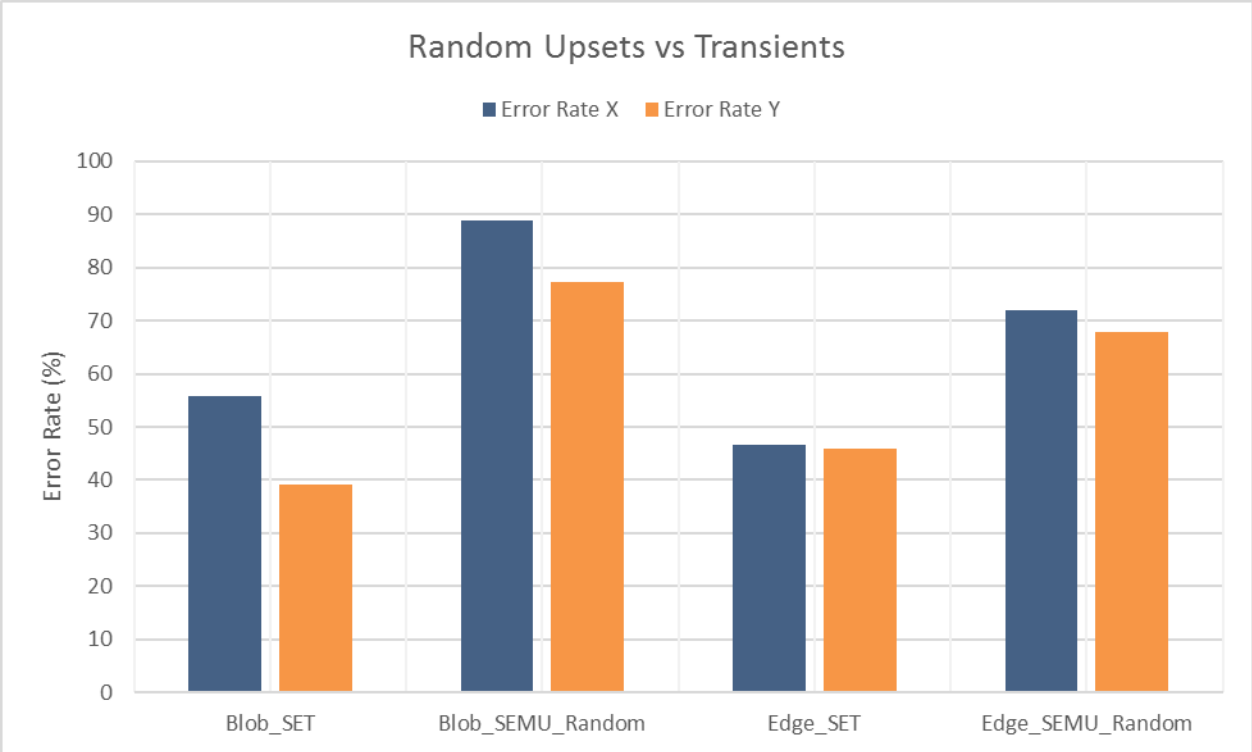

Figure 18: Error rates induced by random SEU or SET faults in Object Extraction module. Error rates of edge-based and blob-based trackers corresponding to SET(stuck-at) and SEMU faults had a similar distribution

The geometric means of errors corresponding to applied fault types were also calculated. Table II list shows that for each fault injection within preprocessing units, the average error values are mostly positive correlated to error rates.

Table III. Geometric mean of output errors and error rates for preprocessing units

Algorithm	FI location	Fault Type	ErrorRateX	ErrorRateX	Error X	Error Y
Blob	Rgb2Gray	SEU	17.73%	12.35%	3.688	1.782
		SEMU	16.65%	15.33%	3.432	1.623
	Object Extraction	SEU	17.82%	9.814%	3.727	2.031
		SEMU	77.78%	65.23%	3.988	2.090
Edge Detection	Rgb2Gray	SEU	26.71%	37.55%	4.445	3.050
		SEMU	33.44%	33.20%	5.999	2.176
	Object Extraction	SEU	65.67%	56.44%	4.695	3.598
		SEMU	88.87%	62.45%	4.923	2.735

b. Data feeder

According to emulation results, most SEUs at the data feeder outputs had a smaller effect on the Gaussian filter of blob-based tracking comparing to edge detection tracking. The distribution of error rates for Gaussian filter roughly matches the approximated Gaussian kernel. Element P22 convolved with the peak caused an error rate of 79% but other elements would lead significantly lower error rates.

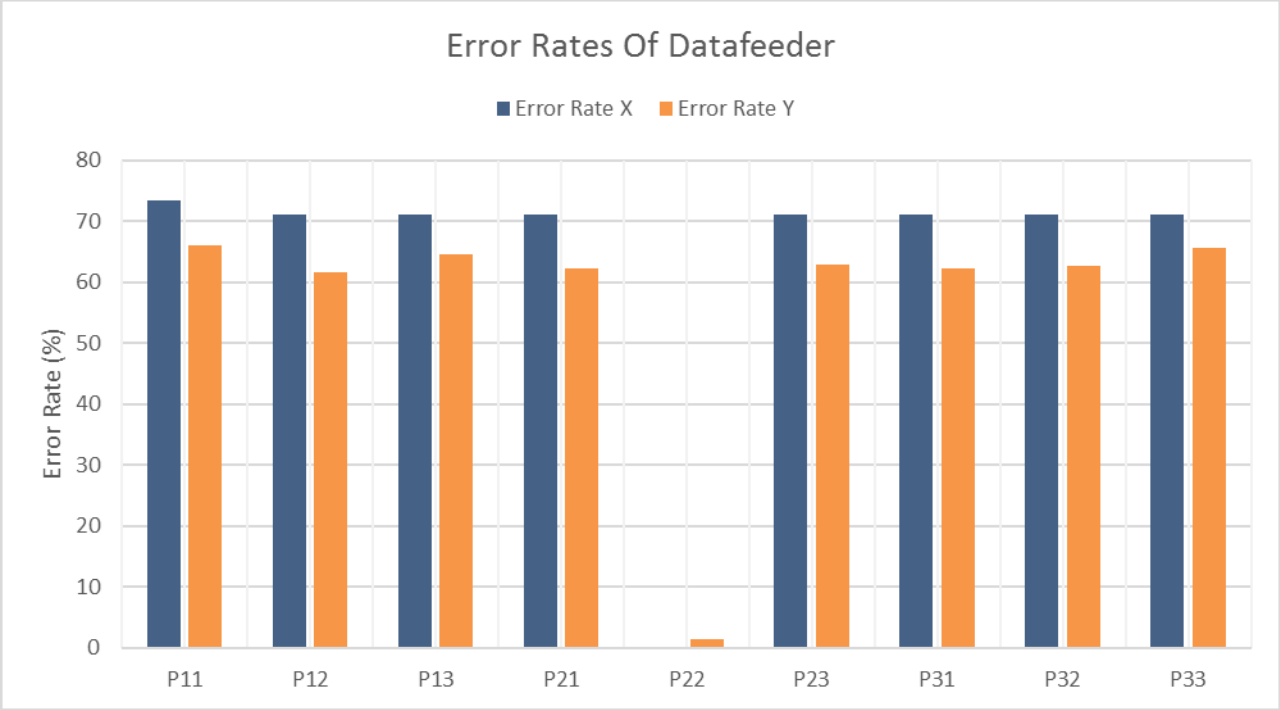

Figure 19: Error rates induced by SEU faults in Data feeder (3-line buffer) module. The pixel (p22) convoluted with "0" had almost no effect on tracking accuracy

However, Figure. 19 shows that these SEU faults would cause great degradation on the performance of edge detection-based tracker. The incorrect tracking results were mainly caused by convolution operations. The outputs of data feeders are virtually the elements within the convoluted image matrices.

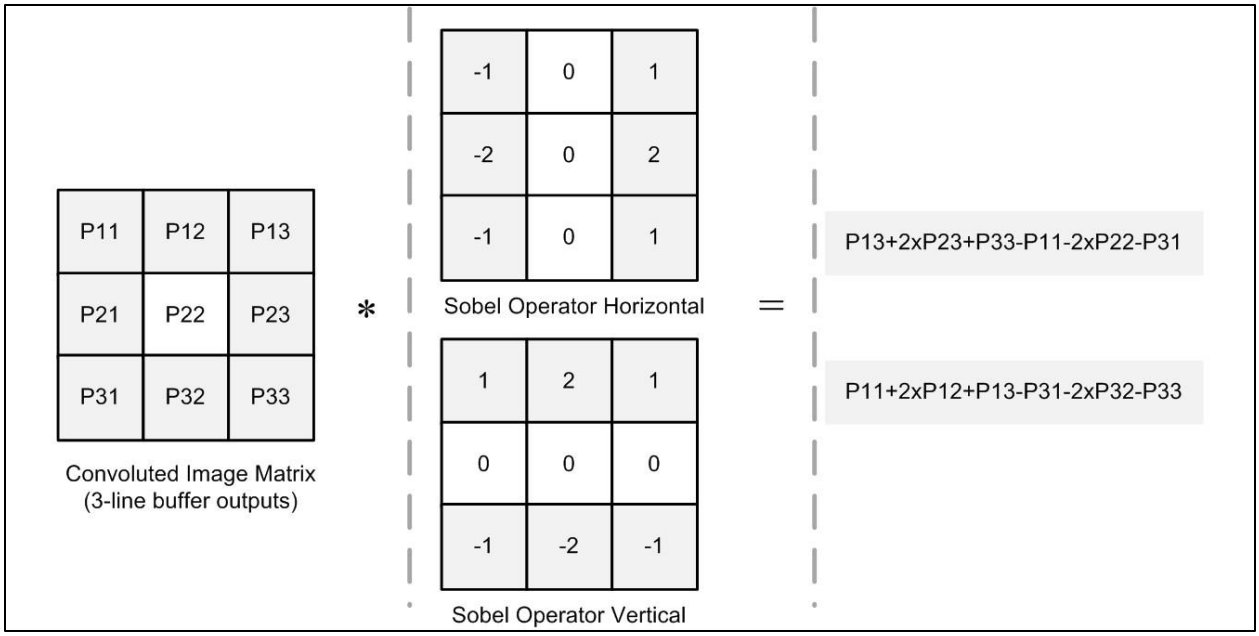

Figure 20: An illustration of fault-vulnerable elements

As shown in Figure.20, if the data feeder outputs elements were multiplied by the non-zero elements in Sobel Operators, the upset faults at these output registers would tend to be much more dangerous. The calculation results of edge would be altered drastically. As shown in Figure.20, the error rates for vulnerable elements ranged from 60% to 80%, whereas P22, the element convoluted with zero was almost not affected by SEU faults. Besides, the results also indicated that both Vertical and Horizontal Sobel Operators are crucial for the edge detection accuracy. If one of Sobel Operators fails, the edge will not be detected precisely, then the corresponding tracking algorithm would also not work properly.

c. Threshold units

Figure. 21 shows that the effects of SEU or SEMU at threshold registers on final outputs were not significant; the error rates of two trackers are similar and mostly ranges from 5% to 10%. The SEMU here could impair the tracking performance but the amount of degradation is

limited. SEMU would cause a maximum error rate of 20.17% and the object tracking system operated moderately. Hence for the object tracking prototype tested here, the corrupted threshold value is not a critical concern.

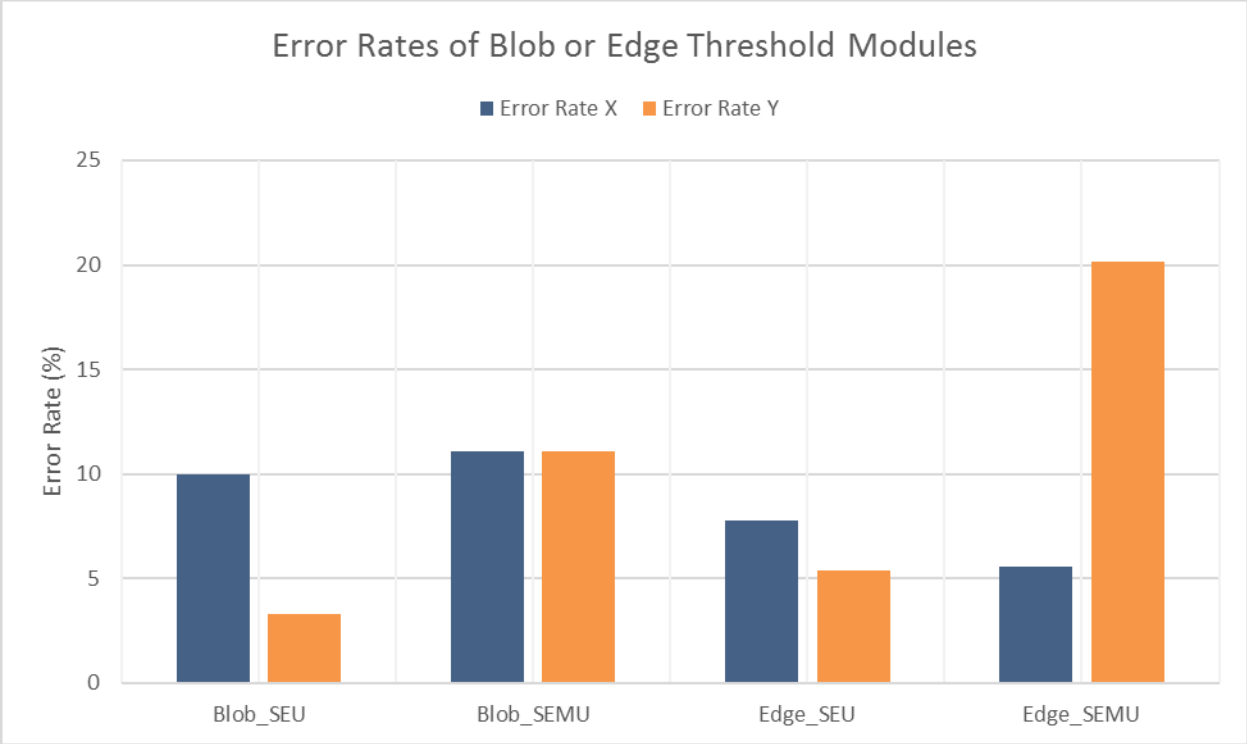

Figure 21: Error rates induced by SEU faults in Threshold units. Corrupted threshold values had very small impact on tracking accuracy

The geometric mean of the error induced by SEU at threshold registers is shown in Table. III, the error values were also positive correlated to error rates within a certain fault injection location. Table III also revealed that though the error rates of faults within edge detection tracker threshold units were relatively low, these faults caused a rather high average error.

Table IV. Geometric mean of output errors and error rates for threshold units

Algorithm	FI location	Fault Type	ErrorRateX	ErrorRateX	Error X	Error Y
Blob	Blob Threshold	SEU	10.01%	3.332%	1.736	1.662
		SEMU	11.08%	11.08%	2.981	1.255
Edge Detection	Edge Threshold	SEU	7.763%	5.371%	6.015	2.494
		SEMU	5.566%	20.17%	5.999	1.689

d. Spatial Filters

The Gaussian filter serves for the blob-based tracking algorithms but Sobel Operators help edge detection tracker. Then the comparison of resilience to SET faults between these two algorithms was performed. As shown in Figure 22-24, When set-high or set-low transient faults are randomly injected among the bits at the fan-out, the vulnerability of two trackers tends to be similar. Horizontal and vertical Sobel operators had nearly the same response to SET faults, and even if one of the Sobel Operator suffered from faults, the final tracking accuracy would be diminished. We can also find that the faults at multipliers in the Sobel operators and adders in Gaussian filter affected tracking accuracy more significantly than another combinational block within the filters. The faults at multipliers in Sobel operators can cause more than 2x error rates comparing with the adders.

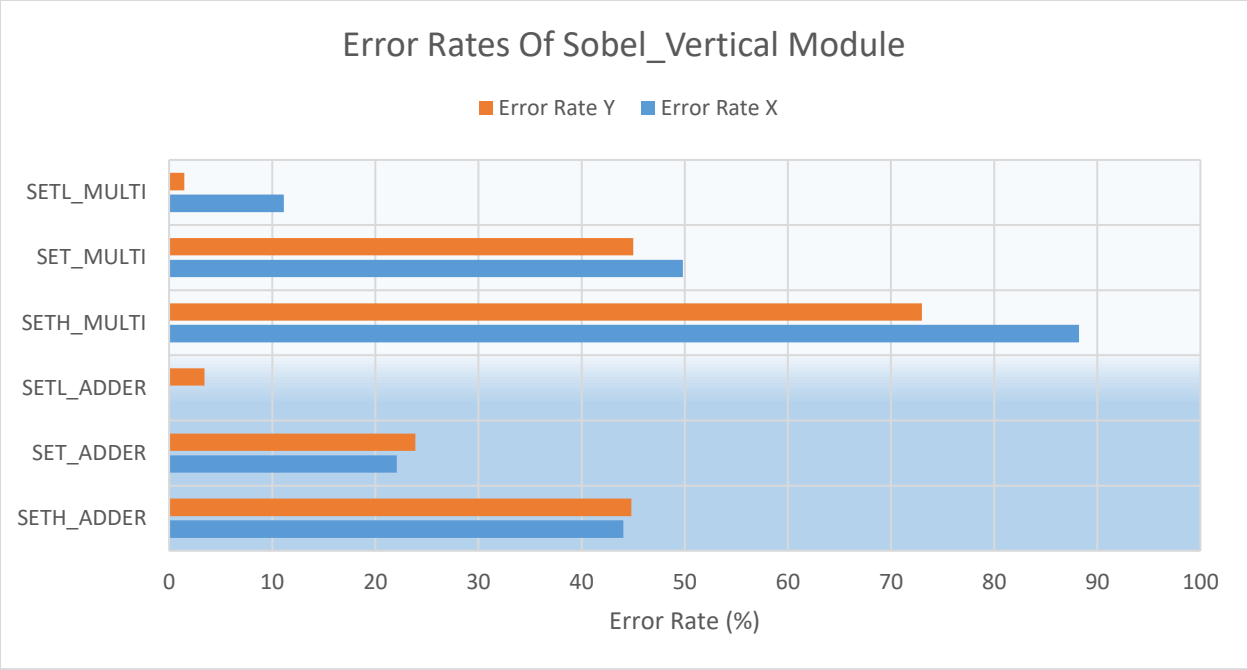

Figure 22： Error rates induced by transient faults in Vertical Sobel Operator module. Transient (stuck-at) faults in multipliers within Sobel Operator could cause more degradation on edge detection-based tracker tracking

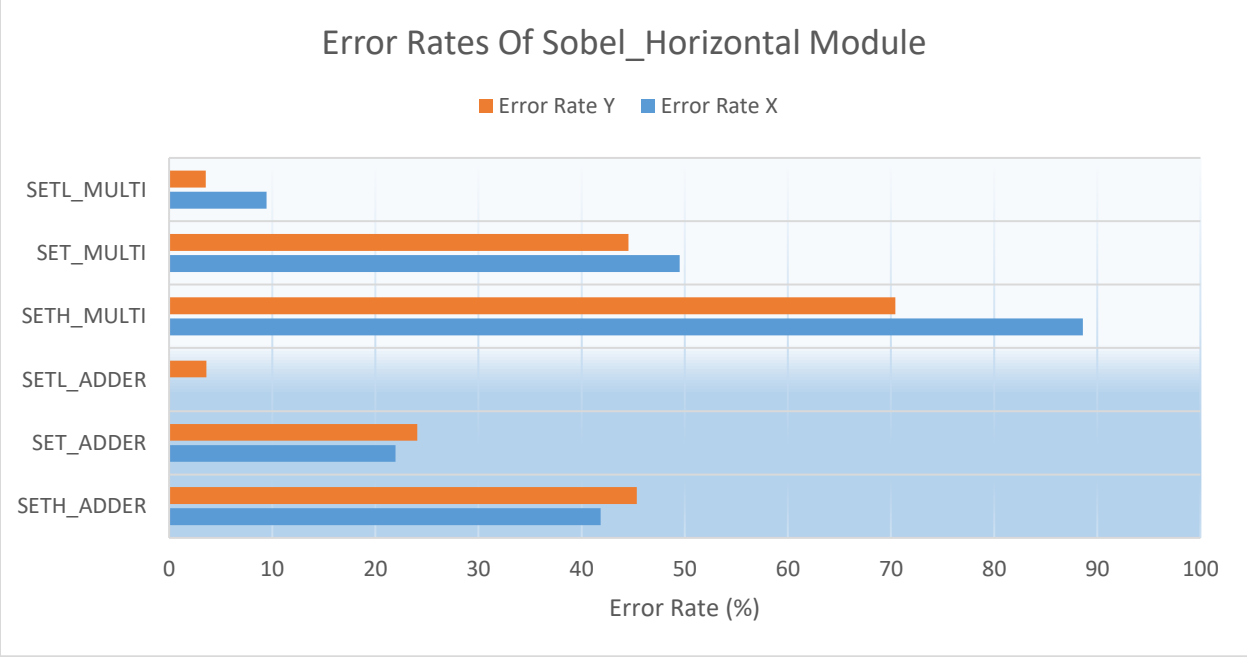

Figure 23： Error rates induced by transient faults in Horizontal Sobel Operator module. Horizontal or Vertical Sobel Operators show similar response to injected faults, and randomly scattered faults induced error rate was the average of all-stuck-H or all-stuck-low

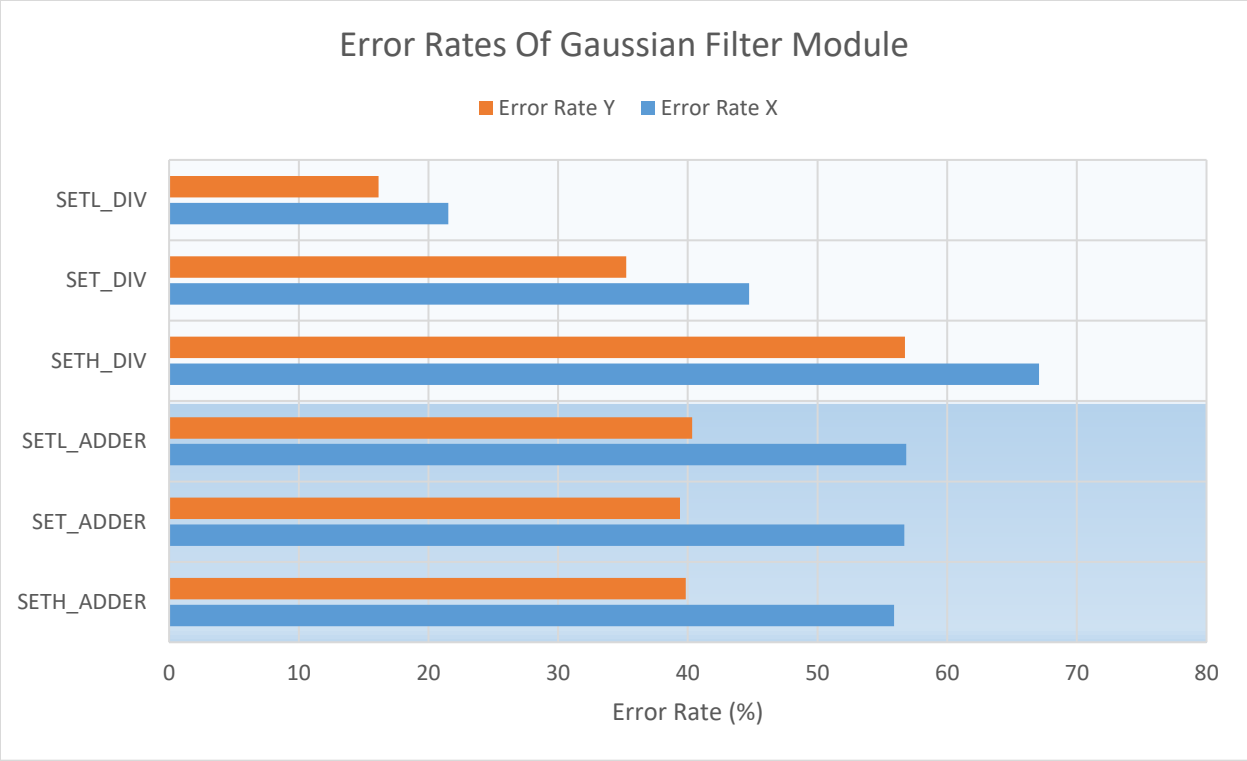

Figure 24. Error rates induced by transient faults in Gaussian Filter module. Gaussian filter tended to be slightly less resilient to transient (stuck-at) faults. SETH (stuck-at-H), SETL (stuck-at-L) or SET(stuck-at) faults in the adders within Gaussian Filter shown similar response

Set-high or set-low faults were also injected into the spatial filters separately, i.e. the bits within the fan-out of combination logic were only set to high or only set to low depending on the fault model selected. Though assuming the transient faults at the fan-out bits to be all-high or all-low does not suit the physics mechanisms and not practical. The processed object tracking results are still worthwhile. One can learn whether the set-high or set-low will cause more degradation on tracking performance and may give a reasonable guideline of hardness. The results show that the output response to set-high faults is far more drastic than set-low faults. Set-low faults had little influence on tracking results, whereas set-high faults can lead to failure of the tracker. For example, when set-high or set-low faults are injected to the multipliers within Horizontal Sobel Operator, the tracking error rates in X direction were 88.23% (set-high) vs.9.443% (set-low).

Another finding from Figure 22-24 is the error rates of randomly injected transient faults nearly equals to the arithmetic average of the error rates induced by solely set high and set low transient faults.

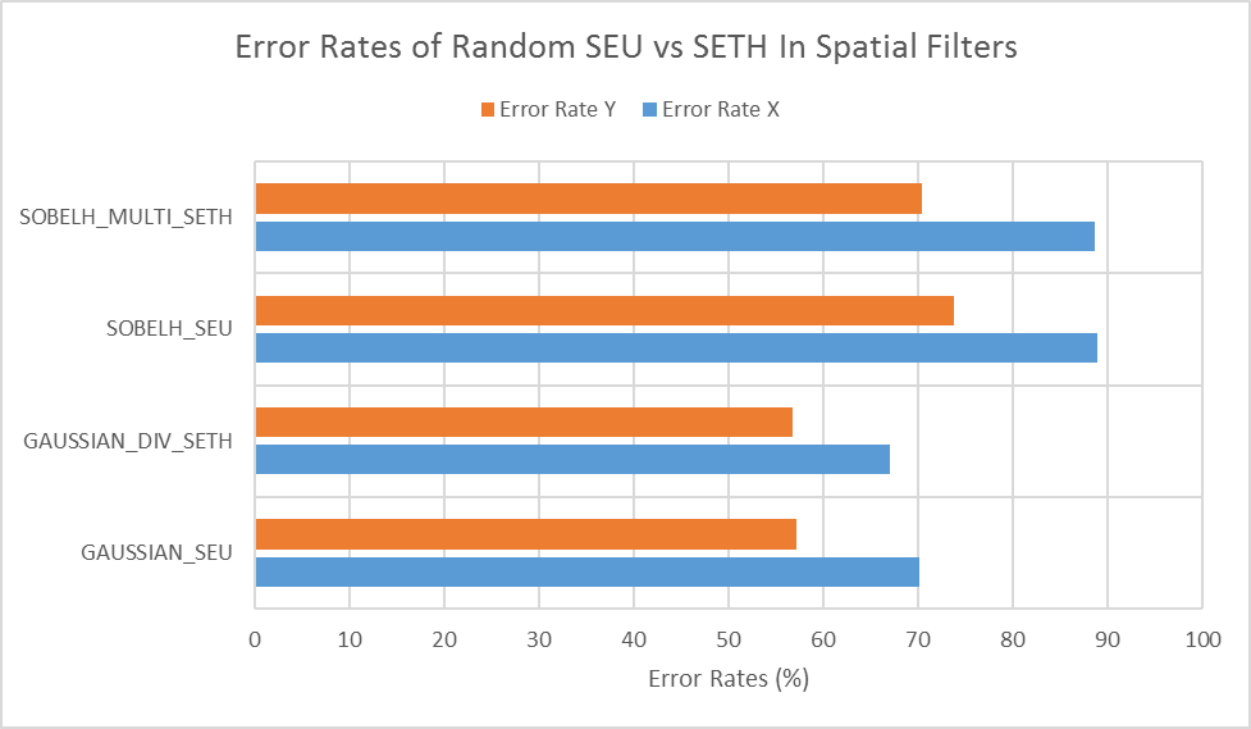

Figure 25： Error rates induced by random SEU and SETH faults in Horizontal Sobel Operator or Gaussian Filter modules. Error rates induced by randomly scattered upset faults correspond well to the worst-case of transient (stuck-at) faults

Random upset faults were injected into the circuit and compared with the error rates induced by SETs. The results indicate the error rates of random upset faults at the output registers of spatial filters matched well with the worst cases (i.e. Set-high faults) of transient faults. As shown in Figure.25, the error rates of SETH or random SEU faults in the Sobel Operators or Gaussian filters are nearly identical. This finding may help to build a relevance between RTL and higher abstraction levels such as larger module composed of a series of RTL modules.

Table V. Geometric mean of output errors and error rates for spatial filter modules

Algorithm	FI location	Fault Type	ErrorRateX	ErrorRateX	Error X	Error Y
Edge detection	Sobel_V ADDERS	SETH	44.04%	44.82%	4.249	2.232
		SET	22.07%	23.88%	4.259	2.051
		SETL	0	3.418%	0	2.894
	Sobel_V MULTI	SETH	88.23%	72.99%	4.197	4.285
		SET	49.80%	45.02%	4.661	2.475
		SETL	11.13%	1.465%	11.999	3.726
	Sobel_H ADDERS	SETH	41.83%	45.36%	4.183	2.394
		SET	21.97%	24.07%	4.197	1.889
		SETL	0	3.613%	0	2.387
	Sobel_V MULTI	SETH	88.61%	70.43%	4.221	4.117
		SET	49.51%	44.53%	4.673	2.508
		SETL	9.443%	3.573%	9.254	2.627
Blob	Gaussian ADDERS	SETH	55.91%	39.84%	3.541	2.005
		SET	56.69%	39.40%	3.719	1.923
		SETL	56.84%	40.33%	3.723	1.912
	Gaussian DIVIDERS	SETH	67.09%	56.74%	3.688	2.575
		SET	44.73%	35.25%	3.816	2.278
		SETL	21.53%	16.16%	3.565	1.871

The average error values of SET faults at spatial filters are shown in Table IV. In most cases, for the faults at a certain fault injection location, the geometric mean of errors also shown a positive correlation with the error rates. It is also noticeable that although the Set-low faults at the multipliers within Sobel Operators had little effects on the tracking accuracy judging by error rates, the average error on X direction could reach 11.999 pixels, which was far too out of bound. It will be a critical concern for tracking systems. For example, if the edge detector tracking results were fed into a filtering-based tracker, such abrupt peaks would be highly likely to cause disturbances on filters. Besides, if one uses average error to evaluate the reliability of the object tracking system, the Horizontal and Vertical Sobel Operators still behaved similarly. Then it is possible to analyze the reliability of the whole Sobel Operators by analyzing one of the operators.

C. Demo Illustration

Two demonstrations were provided, one shows the observable tracking error, another demonstrates the faults can kill the function of spatial filter such as Sobel Operators.

a. Tracking results degradation: fault free vs SET faults in Sobel Operators

As shown in Figure.26. When the circuit was fault-free, the trackers could indicate the approximate position of the object accurately. When randomly scattered SET faults were injected into Horizontal Sobel Operators, the tracker of edge detection-based tracking algorithm (aero blue colored cursor) might stay at the middle or random position on screen rather than following the trajectory of the object. Whereas in some of the frames, both trackers worked fine. The reason could be the Set-low faults were injected at that time. This demo justified the performed numerical analysis.

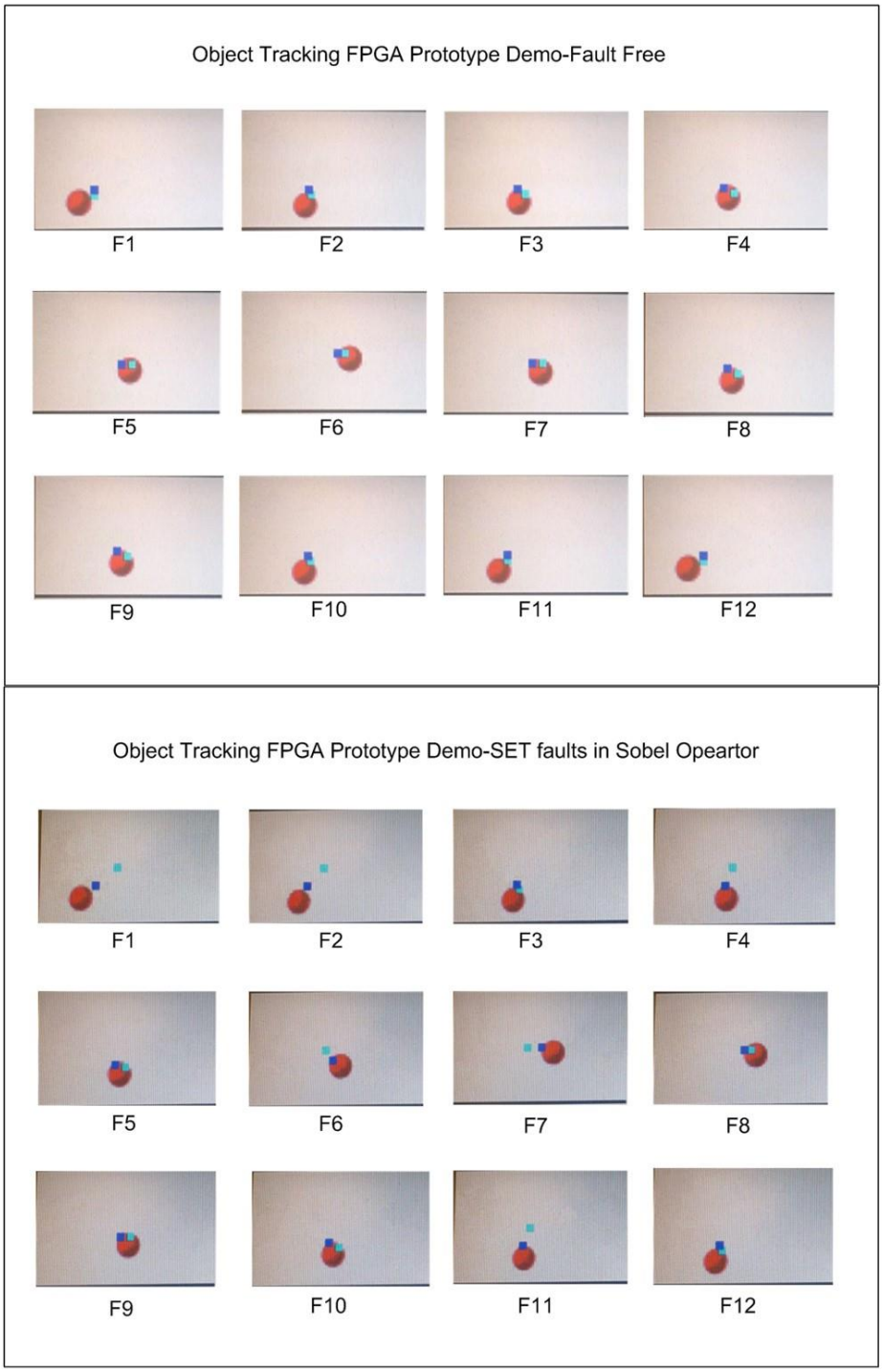

Figure 26: Demo of the impact of radiation-induced faults on object tracking accuracy. When randomly scattered SET faults were injected into Horizontal Sobel Operators, the tracker of edge detection-based tracking algorithm (aero blue colored cursor) might stay at the middle or random position on screen rather than following the trajectory of the object

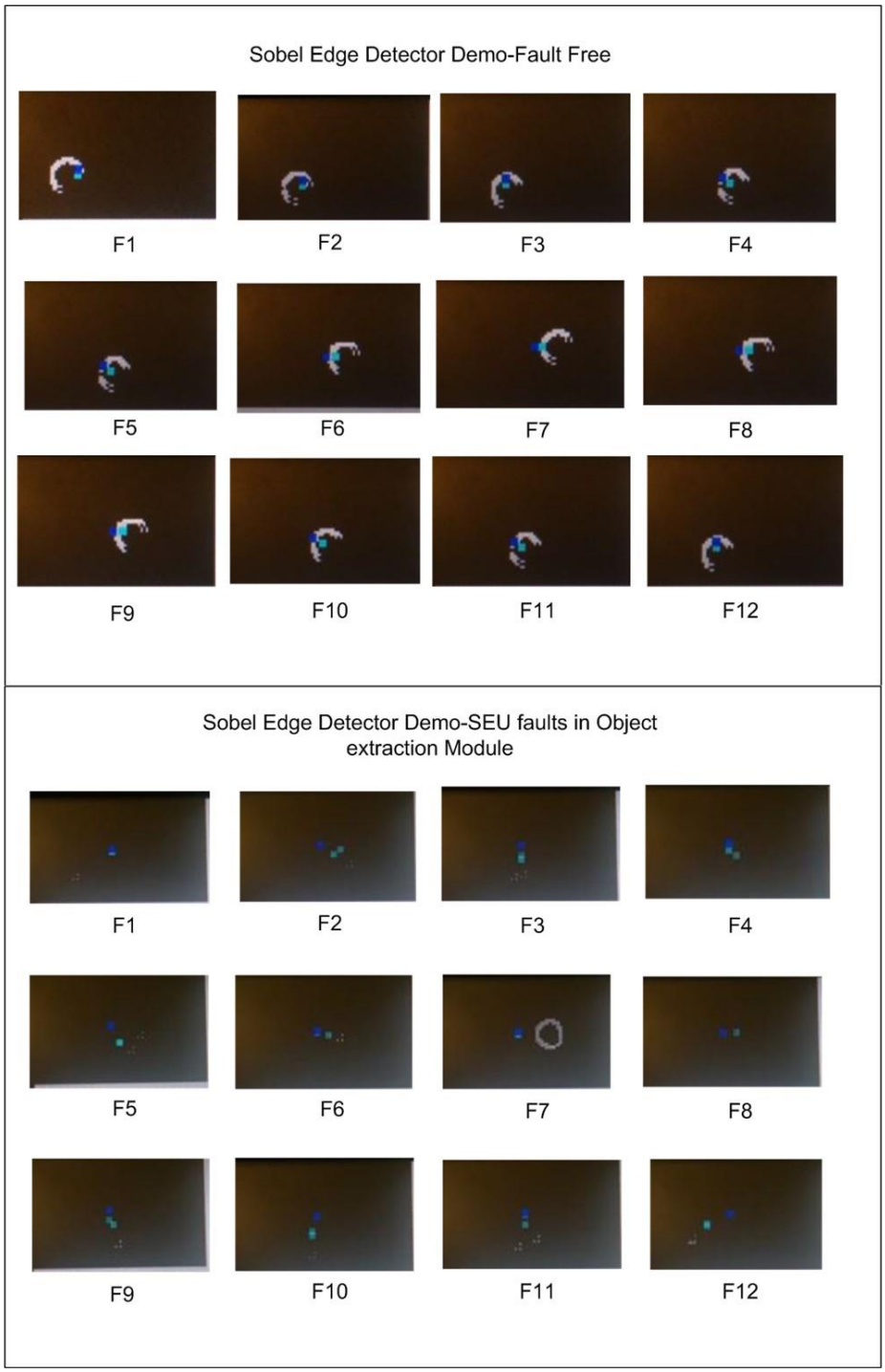

Figure 27: Demo of the impact of radiation-induced faults on edge detector. When SEU faults were injected to object extraction module, the edge of tracked object could not be extracted correctly

b. Edge detection: fault free vs SEMU in object extraction modules

As shown in Figure.17. Both blob-based tracking and edge detection-based tracking were almost failed when SEMU faults were injected into object extraction modules. The demo in Figure.26 shown how SEMU faults killed the Sobel edge detector. When the circuit is fault free, the edge detector can obtain the edge of the object correctly. However, if the SEMU faults are injected, the edge detector was not able to work properly for most of the frames. This demo helped to explain why the tracker would fail when certain types of faults were injected to critical modules.

Comprehensive Analysis

A. General evaluation of algorithms

Apart from analyzing the reliability of individual modules, an aggregated analysis is carried to determine which tracking algorithm is more resilient to radiation-induced faults. By comparing the random scattered transient faults (SET) error rates and average errors in spatial filters, which are shown in Figure.28 and Figure.29. The error rates of SET faults at edge detection ranges from 20% to 50%, whereas SET faults lead to 40% to 60% error rates at the blob-based tracker. Though the average errors for edge tracker are higher, the primary metric is the error rate. Therefore, the blob-based tracker tends to be more sensitive to SET faults.

Regarding the SEU/SEMU faults in preprocessing units, the blob-based tracker had both lower error rates and average error. Besides, the SEU faults in data feeder did not cause any

erroneous outputs on the blob-based tracker (i.e. error rates are zeros). Hence for the DUT in this thesis, the blob-based tracker is more resilient to upset faults than edge detection tracker.

The analysis of the data sets of error rates or average error is important to harden the object tracking hardware for radiation-induced faults. The users of tracking hardware applications can choose a more resilient algorithm to resist certain types of faults. Circuit designers may locate the radiation-sensitive RTL stages within higher level modules and selectively harden these logic blocks. It is also worthwhile to focus on the SEU-resilience of the data path between each large module. Hence the cross-layered reliability analysis between various abstraction levels is worthwhile for the study of IC reliability.

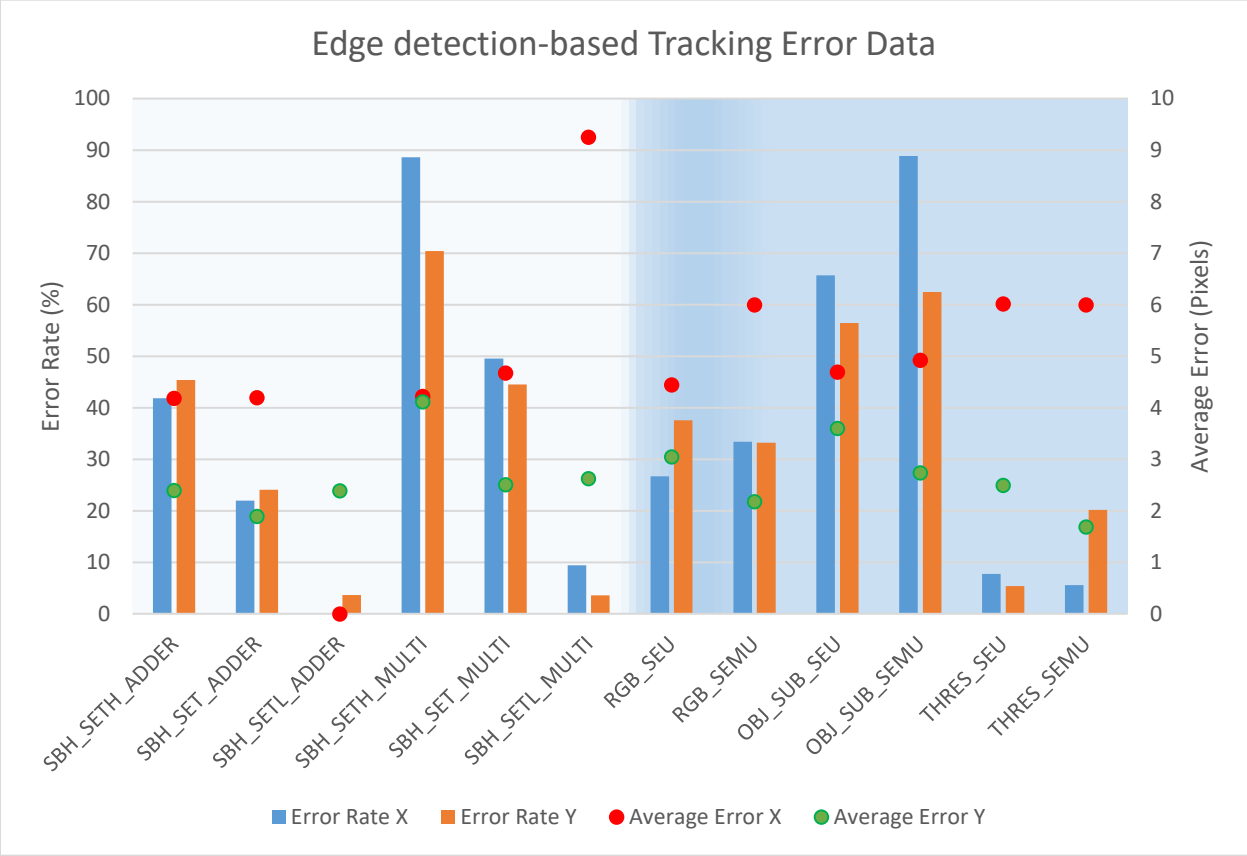

Figure 28: Aggregated tracking error data of edge detection-based tracker

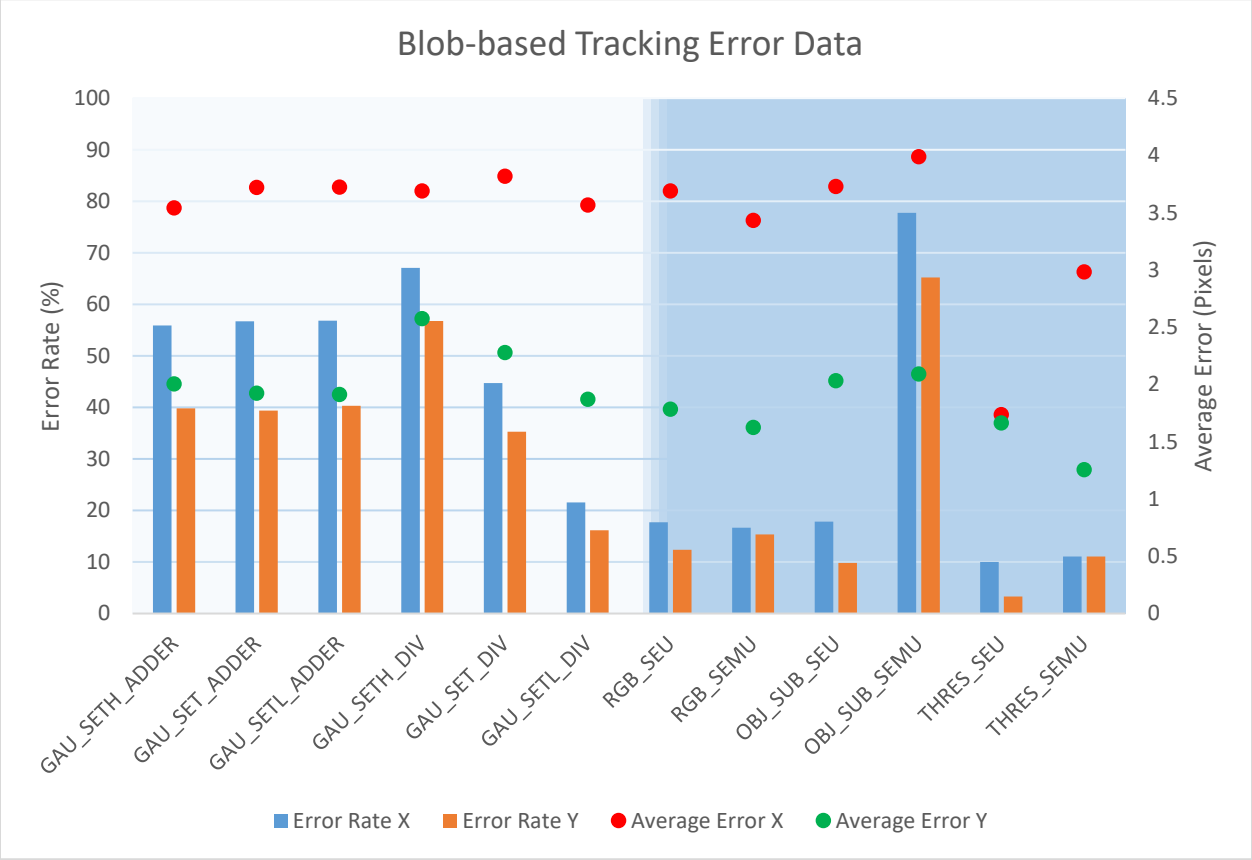

Figure 29: Aggregated tracking error data of blob-based tracker. Comparing with Fig. 28, blob-based tracker is more reliable to upset faults in preprocessing units but slightly more susceptible to stuck-at faults in spatial filters

B. Normalized error rates

a. Normalization based on feature size

Because of the discrepancy between the feature size of edge or blob-based, each error rate corresponds to a certain fault type was normalized by the feature size of the tracked object for both blob tracker and edge tracker. Feature size could be obtained by counting the pixels which represent the object in each frame of filtered image processed by threshold unit modules.

For the fault-free object tracking system in this work, the feature size of blob tracking algorithm is approximately 20 pixels and the feature size of edge-detection tracking is 30 pixels.

As shown in Figure.30, error rates normalized by feature size distributed similarly to aggregated error rates. Blob tracking was more vulnerable to transient (Stuck-at) faults but more resilient to upset faults, which justified the conclusion drawn from aggregated analysis.

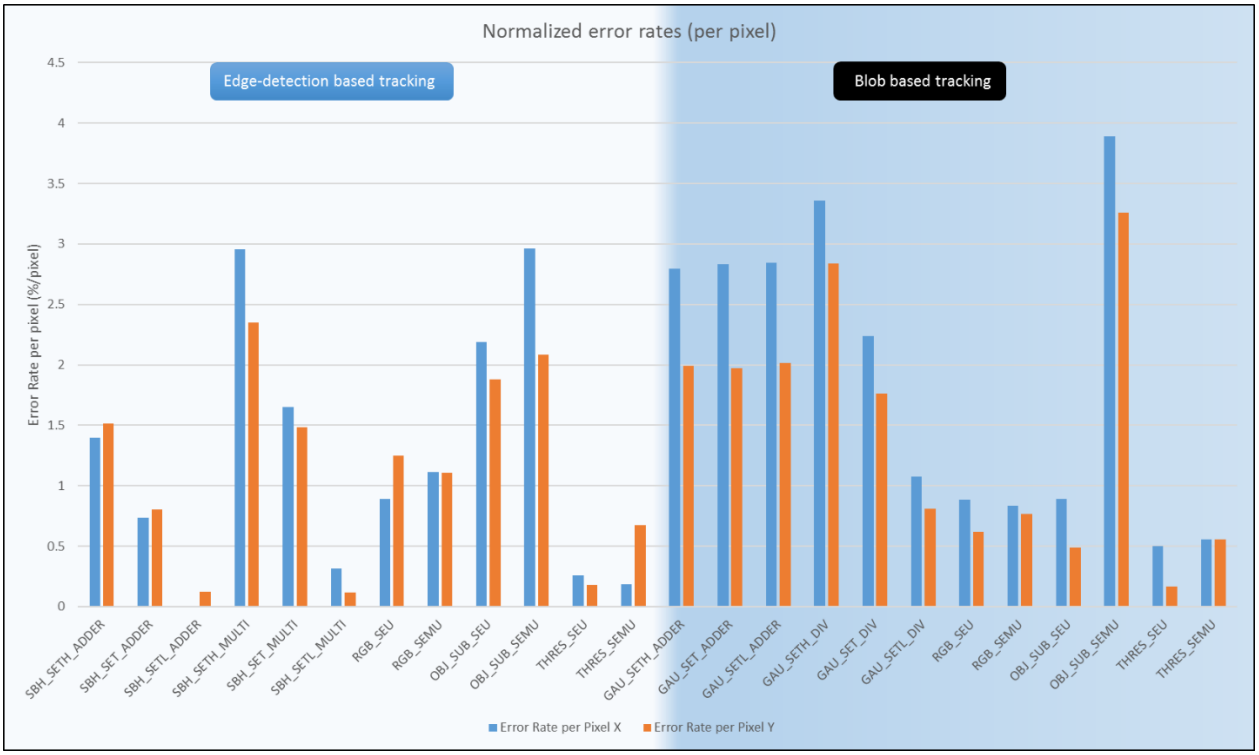

Figure 30: Normalized Error Rates (normalized to error rate per feature). The distribution of error rates matches the results in Figure.28 and Figure.29 well

b. Normalization using coefficient of variation

To evaluate the dispersion of the tracking accuracy of each tracking algorithm, the error rates and average errors of blob tracker and edge-detection tracker were also normalized using the coefficient of variation, which is defined by Equation. 10.

$$Coefficient\ of\ variation_{tracker} = \frac{Standard\ Deviation_{tracker}}{Mean_{tracker}} \quad (10)$$

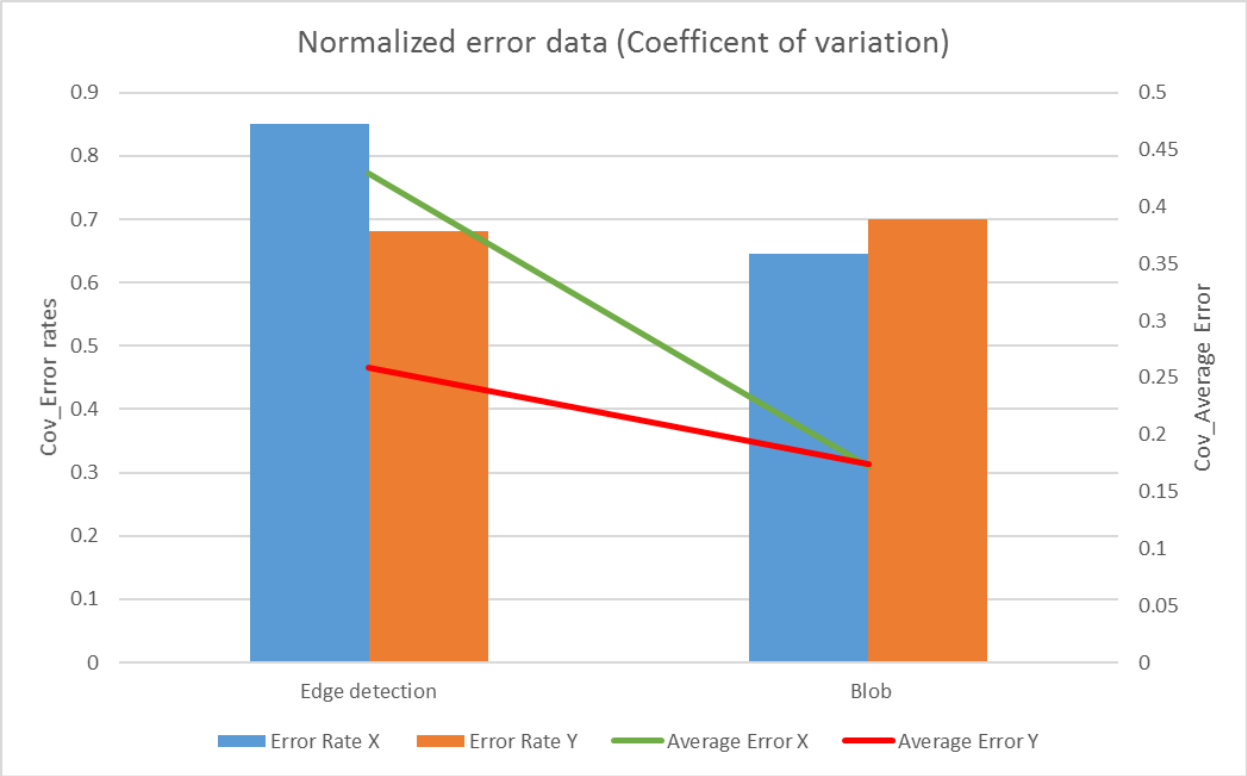

Figure 31: Normalized Error Rates and Average Error (Normalized using the coefficient of variation). Blob-based tracking had slightly lower coefficients than edge detection-based tracking

As shown in Figure.31, Both of error rates and average errors coefficient of variance of the blob tracker tended to be lower the than edge detection-based tracker. Therefore, blob tracker in this design would be preferable if the application required a low variance.

CHAPTER 5

Conclusion and Future Works

A cross-layered reliability analysis between RTL and application of object tracking level was performed on two basic tracking algorithms implemented on the FPGA prototype. The reliability of object-tracking algorithm is evaluated from the perspective of how the circuit structures would response to radiation-induced faults. A fault emulation experiment was carried and common radiation-induced faults including upsets and transients were injected into the data buses or registers within core modules of the object-tracking FPGA prototype. Metrics including error rate and average error value were provided for analyses. Demos were also presented here to give an illustration on how the faults affected the monitor-side of the object-tracking application.

The emulation results indicated that radiation-induced faults in hardware can cause erroneous tracking results that can be observed on the monitor. When the faults strike the critical modules, the tracker even fails. Typically, the average observable error values range from 1.8 to 5 pixels. The error value can be as high as 11.999 pixels under the worst cases. A more important evaluation metric is the error rate, which helps evaluate the sensitivity of individual modules to various types of faults. The level of tracking accuracy degradation of different modules can be quantified via error rates, then it is possible to selectively harden the modules within the tracking algorithm hardware based on the error rates via trending circuit hardening techniques. From the perspective of an algorithm designer, another approach is to pick a proper tracking algorithm based on available hardware and the radiation environment. For example, in the FPGA prototype

tested here, generally the blob-based tracker is more resilient to SEU/SEMU faults at preprocessing units but more vulnerable to SET faults at spatial filters than the edge detection-based tracker, then designer could optimize the hardware to the given types of faults.

To explore the impact of radiation-induced faults on state-of-the-art object-tracking algorithms, more advanced object-tracking algorithm will be implemented in hardware and utilized as DUT for fault emulation. It will also be worthwhile to perform a gate-level fault emulation; the results of gate-level emulation can be used to confirm the credibility of higher level fault injection techniques. A more precise evaluation could be presented when the tracking algorithms are implemented with a standard cell library for an ASIC, whose results not only reflect the reliability of the design of digital systems but can be used to explore the connection between higher level faults and lower level faults. Additionally, the gathered data of fault emulation or simulation experiments could be processed in a more elegant way. To give an impartial evaluation on the reliability of different object tracking algorithms hardware implementations, more reasonable statistics or data processing methods should be applied because of possible discrepancies in feature size of tracked objects or other factors in various tracking algorithms.

Another possible direction of this work is estimating the reliability of object tracking hardware using the flipping state bits in cycle-accurate simulation method. The vulnerability factor of a tracking could be calculated if the analysis such as architecturally correct execution analysis is carried (ACE Analysis). ACE analysis provides a conservative reliability estimation, which could be compared with the estimation obtained by performing RTL fault injection, gate-level fault simulation with a standard cell library, and experimental fault injection using laser and an ASIC chip of the design. The reliability evaluation carried on various abstraction layers will

be used to validate each other. This comparison could help us to find an optimized reliability analysis method which balanced the accuracy, the time consumption, and the computational complexity.

Appendix

Altera DE2 on board SRAM initialization

In this work, the test video was stored in on board SRAM rather than the commonly used on-chip memory (M4K blocks). The syntax of SRAM initialization file is also slightly different from the .mif file or .hex file we use to initialize the memory generate by Altera Megafunctions. Though SRAM initialization file is also an ASCII text file that specifies the content of a memory block, which assigns an initial value for each address. We only need the data values corresponding to their addresses. and the data values should be specified as 16-bit hexadecimal numbers. An example of initialization file is shown below, which initialized address 16'b0 to 16'b4. As shown in the example, only includes the values used to initialize SRAM:

```
0100
0111
0454
0887
0232
```

The initialization file is generated using Matlab here, an example of SRAM initialization generation Matlab code is provided below, which is configured to resize any input images to 80×45 resolution, 12-bit RGB initialization file. The function takes a series of video frames as input, the user should specify the output resolution (number of columns or rows) and the number of frames. The outputs are 16-bit hexdecimal numbers but only lower 12 bits were used to store RGB pixel values (4-bit for each channel).

```
function [outfname, rows, cols] = sram_gen(frames , numrows, numcols)
outfname=strcat('frame',int2str(frames),'.hex');
fid = fopen(outfname,'w');
```

```
n=frames;
for i=0:n
   infile=strcat('f',int2str(i),'.jpg');
   img = imread(infile);
   imgresized = imresize(img, [numrows numcols]);
   [rows, cols, rgb] = size(imgresized);
   imgscaled = imgresized/16 - 1;
   imshow(imgscaled*16);
   index=4096*(n+1);
      for r = 1:rows
         for c = 1:cols
            red = uint16(imgscaled(r,c,1));
            green = uint16(imgscaled(r,c,2));
            blue = uint16(imgscaled(r,c,3));
            color = red*(256) + green*16 + blue;
            fprintf(fid,'%04x \n',color);
         end
      end
      for m=1:496
         fprintf(fid,'%04x \n',0);
      end
end
% fprintf(fid,'END;');
fclose(fid);
```

Additional Study of Blob tracker using Median filter

After the thesis presentation, I designed and performed similar soft error fault emulation experiments to a third tracker: blob tracker using the median filter, as a supplemental material for this thesis. The new tracker shared the same preprocessing units and line-buffers with the rest trackers. The median filter was implemented with a series of comparators, which was also a full-hardware implementation described in Verilog. Fault models are better defined to be:

- Upset faults induced by direct particle strike in memory cells, these faults should be modeled as single (SEU) or multiple bit-flip faults (SEMU) happen on one bit or adjacent bits.
- Upset faults induced by registered transient faults, which are modeled as set-high (SETH) or set low (SETL) faults. The distribution of SETH or SETL faults is approximated to random scattering across the corrupted data.

Fault emulation was performed using the same experiment configurations. However, the threshold of observable errors is redefined. A notation of overlap ratio [37] is used to measure the tracking accuracy of DUT. The overlap ratio is defined by Equation (1). If the overlap ratio is 0, then the object is not tracked correctly at all and the tracker fails, whereas when overlap ratio is Equation. 11, the tracking results of DUT match gold reference perfectly.

$$Overlap\ Ratio = \frac{Area\ of\ overlapped\ bounding\ box_{DUT\ vs\ Gold\ ref}}{Area\ of\ tracked\ object\ bounding\ box} \quad (11)$$

The error rates with respect to the overlap ratio between the DUT and ground truth for fault injection at preprocessing sub-modules are shown in Figure. 32. Blob tracking using a median filter was highly resilient to SEU/SEMU faults and the effect of SEMUs is negligible.

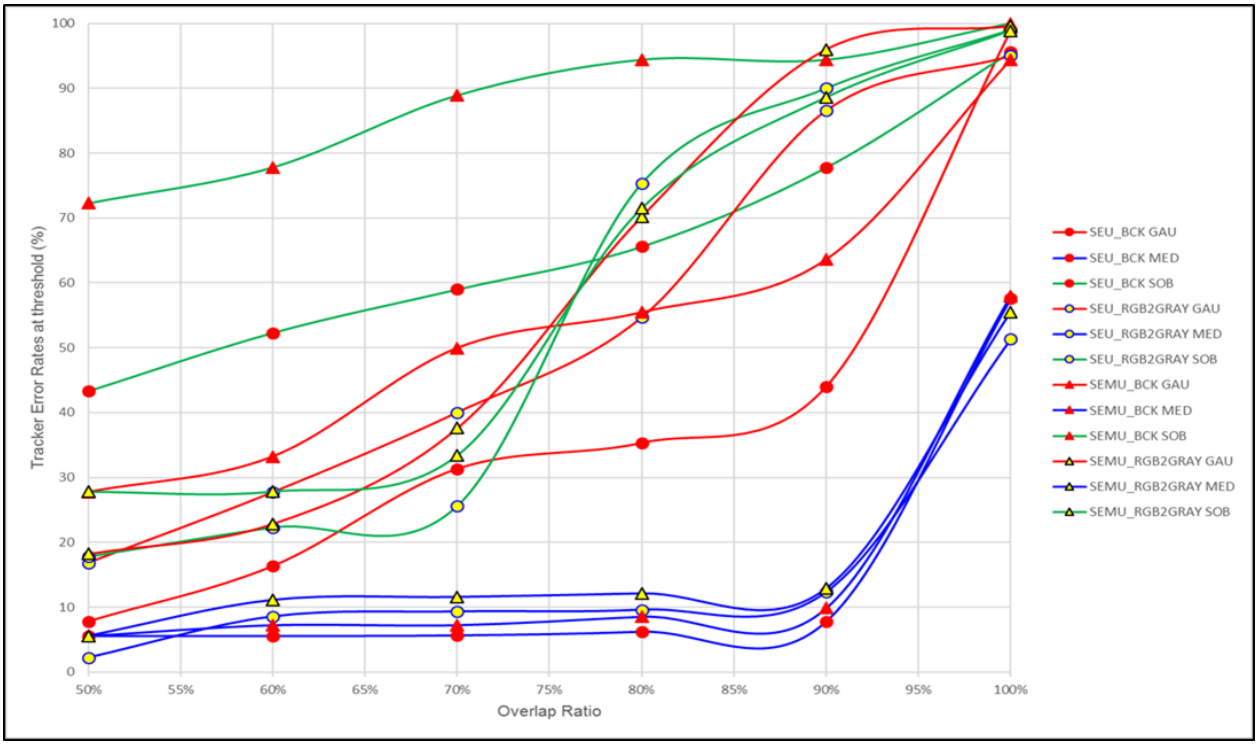

Figure 32. Observable error rates vs. overlap ratio for fault injection at preprocessing sub-modules, median filter is resilient to SEU/SEMU faults, Blob tracker using median filter is highly resilient to upset faults

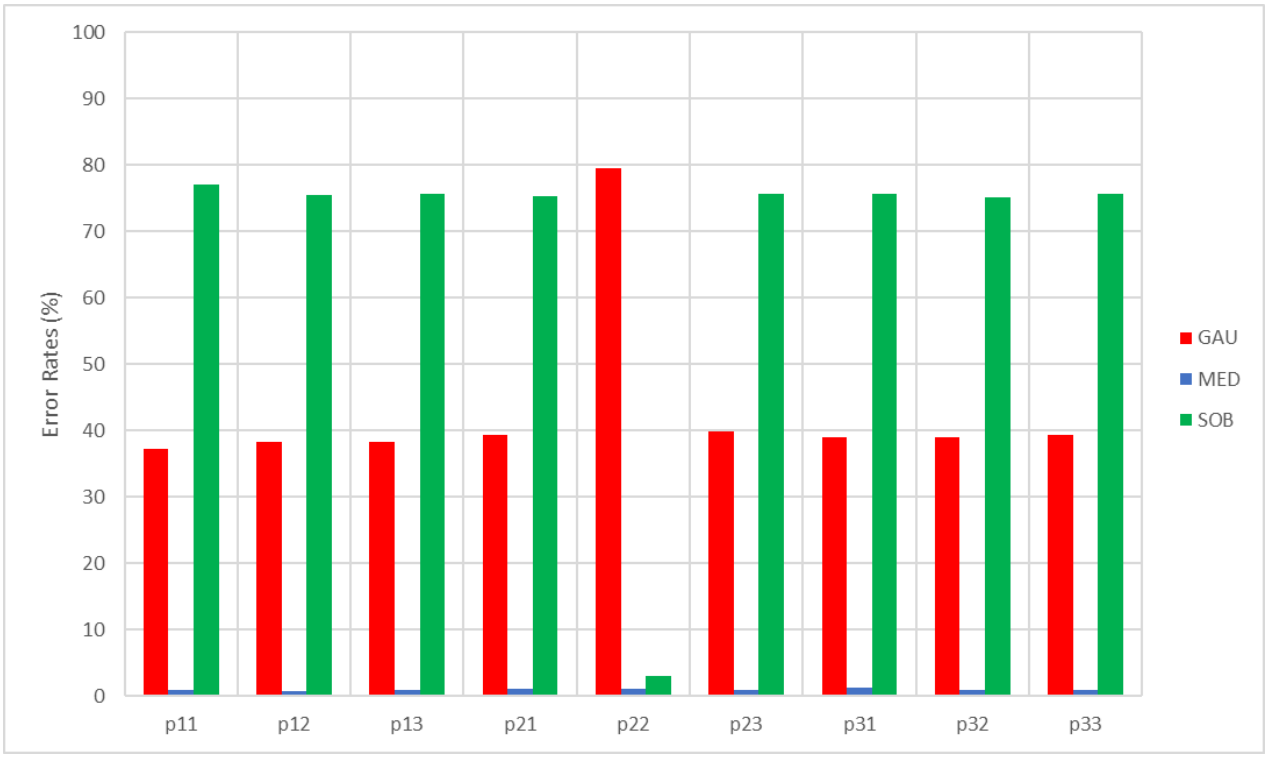

Figure 33. SEU induced observable error rates for output elements of Line buffer, SEU had negligible effect on median filter

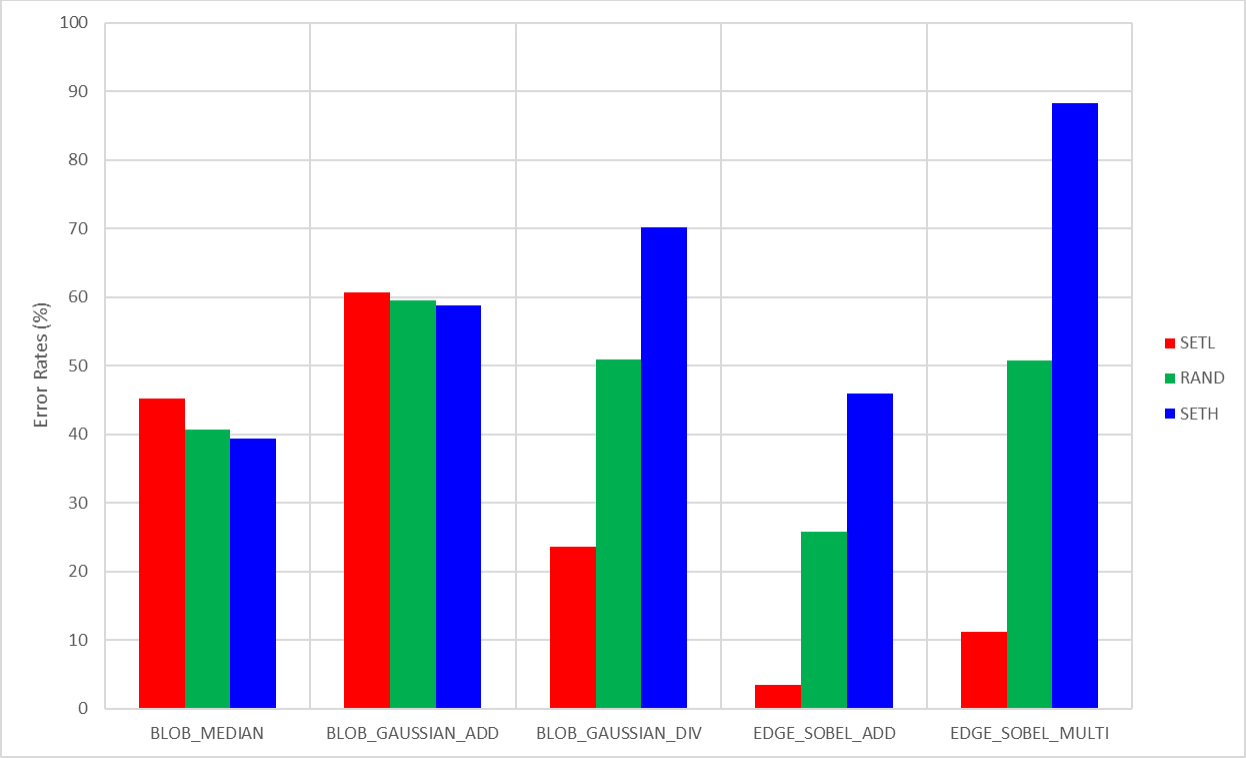

Figure 34. Transient induced observable error rates for registers in digital Filters

In this work, the threshold is set to be an overlap ratio of 0.8. As shown in Figure. 33, SEUs at the line buffer outputs had a negligible impact on the median filter because any single faulty pixel is highly likely to be masked by comparator trees. Figure. 34 shows the error rates induced by latched transient faults. The error rates induced by latched transient faults at Median filters 39% to 45%. The experimental results indicated that blob tracker using the median filter is significantly more resilient to upset faults induced by direct particle strike than the blob tracker using the Gaussian filter or edge detection tracker with Sobel Operator. However, the median filter is still vulnerable to latched transient induced upset faults. If we need to harden the object-tracking hardware to upset faults induced by particle strike, a blob tracker using the median filter is the best option among the three trackers.

REFERENCES

[1] Yilmaz, A., Javed, O., and Shah, M. “Object tracking: A survey”. *ACM Comput. Surv.* 38, 4, Article 13, Dec 2006.

[2] D. B. K. Trieu and T. Maruyama, "An Implementation of the Mean Shift Filter on FPGA," *2011 21st International Conference on Field Programmable Logic and Applications*, Chania, 2011, pp. 219-224.

[3] Bonato, Vanderlei, Eduardo Marques, and George A. Constantinides. "A floating-point extended kalman filter implementation for autonomous mobile robots." *Journal of Signal Processing Systems* 56.1 2009, pp. 41-50.

[4] S. Liu, A. Papakonstantinou, H. Wang and D. Chen, "Real-Time Object Tracking System on FPGAs," *2011 Symposium on Application Accelerators in High-Performance Computing*, Knoxville, TN, 2011, pp. 1-7.

[5] P. E. Dodd and L.W.Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics, ” *IEEE Transactions on Nuclear Science*, vol. 50, pp. 583–602, 2003.

[6] N. Miskov-Zivanov and D. Marculescu, "Multiple Transient Faults in Combinational and Sequential Circuits: A Systematic Approach, " *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1614-1627, Oct. 2010.

[7] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, "Single event transients in digital CMOS - A Review," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 1767-1790, 2013.

[8] Han, Bing, et al. "Robust feature-based object tracking." *Proc. of SPIE Vol*. Vol. 6568. 2007.

[9] A. DeHon, H. M. Quinn, and N. P. Carter, "Vision for cross-layer optimization to address the dual challenges of energy and reliability," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010, pp. 1017-1022.

[10] J. L. Barth, C. S. Dyer and E. G. Stassinopoulos, "Space, atmospheric, and terrestrial radiation environments," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 466-482, June 2003.

[11] Yang, Hanxuan, et al. "Recent advances and trends in visual tracking: A review." *Neurocomputing* 74.18 (2011): 3823-3831.

[12] Choi, David, and Benjamin Van Roy. "A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning*." Discrete Event Dynamic Systems* 16.2 2006, pp. 207-239

[13] C. R. Lee and Z. Salcic, "A fully-hardware-type maximum-parallel architecture for Kalman tracking filter in FPGAs," *Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications*, 1997, pp. 1243-1247 vol.2.

[14] R. Serrano-Gotarredona *et al*., "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory–Processing– Learning–Actuating System for High-Speed Visual Object Recognition and Tracking," *IEEE Transactions on Neural Networks*, vol. 20, no. 9, pp. 1417-1438, Sept. 2009.

[15] C. R. Yount and D. P. Siewiorek, "A methodology for the rapid injection of transient hardware errors," *IEEE Transactions on Computers*, vol. 45, pp. 881-891, 1996.

[16] R. D. Schrimpf *et al*., "Multi-Scale Simulation of Radiation Effects in Electronic Devices," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 1891-1902, Aug. 2008.

[17] R. A. Reed, et al, "Anthology of the development of radiation transport tools as applied to single event effects," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 1876-1911, 2013.

[18] ITC'02 website, available at http://itc02socbenchm.pratt.duke.edu/

[19] Kiddie, Bradley T., William H. Robinson, and Daniel B. Limbrick. "Single-event multiple-transient characterization and mitigation via alternative standard cell placement methods." *ACM Transactions on Design Automation of Electronic Systems* (TODAES) 20.4 (2015): 60.

[20] P. A. Thaker, V. D. Agrawal and M. E. Zaghloul, "A test evaluation technique for VLSI circuits using register-transfer level fault modeling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1104-1113, 2003.

[21] Frank Vahid (2010). Digital Design with RTL Design, Verilog and VHDL (2nd ed.). John Wiley and Sons. p. 247. ISBN 978-0-470-53108-2.

[22] Kudva, PaKJW, et al. "Fault injection verification of IBM POWER6 soft error resilience." *Architectural Support for Gigascale Integration (ASGI) Workshop*. 2007.

[23] H. M. Quinn, D. A. Black, W. H. Robinson and S. P. Buchner, "Fault Simulation and Emulation Tools to Augment Radiation-Hardness Assurance Testing," in *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2119-2142, June 2013.

[24] H. Quinn and M. Wirthlin, "Validation Techniques for Fault Emulation of SRAM-based FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 62, no. 4, pp. 1487-1500, Aug. 2015.

[25] Altera Fault Injection IP Core User Guide, available at https://www.altera.com/en_US/pdfs/literature/ug/ug_fault_injection.pdf

[26] M. A. Aguirre *et al*., "Selective Protection Analysis Using a SEU Emulator: Testing Protocol and Case Study Over the Leon2 Processor," in *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 951-956, Aug. 2007.

[27] F. Ghaffari, F. Sahraoui, M. El Amine Benkhelifa, B. Granado, M. A. Kacou and O. Romain, "Fast SRAM-FPGA fault injection platform based on dynamic partial reconfiguration," *2014 26th International Conference on Microelectronics (ICM)*, Doha, 2014, pp. 144-147.

[28] M. Shokrolah-Shirazi and S. G. Miremadi, "FPGA-Based Fault Injection into Synthesizable Verilog HDL Models," *2008 Second International Conference on Secure System Integration and Reliability Improvement*, Yokohama, 2008, pp. 143-149.

[29] A. Vallero *et al*., "Cross-layer system reliability assessment framework for hardware faults," *2016 IEEE International Test Conference (ITC)*, Fort Worth, TX, 2016, pp. 1-10.

[30] Altera ALTPLL (Phase-Locked Loop) IP Core User Guide, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_altpll.pdf

[31] DE2 User Manual, ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

[32] Altera Embedded Memory IP Cores User Guide, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_ram_rom.pdf

[33] Cornell ECE 5670 Course Page, http://people.ece.cornell.edu/land/courses/ece5760/index.html

[34] Altera RAM-Based Shift Register (ALTSHIFT_TAPS) IP Core User Guide, https://www.altera.com/en_US/pdfs/literature/ug/ug_shift_register_ram_based.pdf

[35] C. V. Krishna, A. Jas and N. A. Touba, "Test vector encoding using partial LFSR reseeding," *Proceedings International Test Conference 2001 (Cat. No.01CH37260)*, Baltimore, MD, 2001, pp. 885-893.

[36] Altera SignalTap II with Verilog Designs, available at: ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Tutorials/Verilog/SignalTap.pdf

[37] Y. Wu, J. Lim and M. H. Yang, "Object Tracking Benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834-1848, Sept. 1 2015.