Implementation of SRAM Based Error Correction and Detection in Memory System Using LFSR

M Mounika

MTech Student, Department Of ECE, VLSI System Design, Avanthi Institute of Engg. & Tech., India. Email: mounika.murari@gmail.com

Dr. S. Kishore Reddy

Associate professor, HOD, Department Of ECE, VLSI System Design, Avanthi Institute of Engg.&Tech., India. Email: kishorereddy416@gmail.com

V Nagaraju

Assistant professor, Department Of ECE, VLSI System Design, Avanthi Institute of Engg. & Tech., India. Email: vasanthanagayadav@gmail.com

***Abstract*-Ternary Content Addressable Memories, or TCAMs, are often used by network devices in order to conduct packet categorization. For example, they are used in the construction of software-defined networks, the management of security, and the transmission of packets (SDNs). TCAMs are often used either as standalone devices or as a component embedded into networking application-specific integrated circuits. TCAMs may also be used in either capacity simultaneously. When working with memory, one of the problems that might arise is the possibility of soft errors destroying the bits that have been saved. The memories might be protected by using an error-correcting code or a parity check to locate any errors; however, doing so would require an increase in the number of memory bits used each word. This approach takes into consideration the need of maintaining the integrity of the memory while simulating TCAMs. This technique gives protection against soft faults and the error correcting strategy that provides rapid response time, inexpensive cost, and excellent search performance in order to deliver an error-free SRAM-Based TCAM Design. In addition, this method offers protection against hard faults.**

Keywords- TCAMs, SRAM, Transmission, Parity Check, Soft Faults.

Manuscript received Dec 10, 2022; Revised Dec 25, 2022; Accepted Dec 31, 2022

1. INTRODUCTION

Very-large-scale integration (VLSI) integration refers to the process of creating an integrated circuit by placing a large number of semiconductors on a single chip (IC). In the 1970s, as more complex semiconductor and communication technologies were being created, very large-scale integration (VLSI) was first introduced. The chip in question is a very large-scale integration device. Before the development of VLSI technology, the bulk of ICs were only capable of carrying out a limited number of functions simultaneously. An electrical circuit may include a central processing unit (CPU), ROM, random access memory (RAM), and several other components. VLSI makes it possible to put all of them into a single chip.

The history of the transistor may be traced back to the middle of the 1920s, when many inventors attempted to change the current that was running through solid-state diodes so that they would become triodes. This was the first step in the development of the transistor. Since the end of the Second World War, the production of radar detectors using silicon or germanium crystals has significantly contributed to the advancement of both practical and theoretical knowledge. Researchers working in the field of radar have started producing solid-state radar systems again. The era of vacuum tubes gave way to that of solid-state devices when the transistor was invented at Bell Labs in 1947. This marked the beginning of the modern technological era.

In the 1950s, mechanical engineers came to the realization that the relatively simple transistor might be employed in the construction of more complex circuits. However, complications emerged as the level of circuit intricacy grew. The magnitude of the circuit was one of the issues. The dynamic circuit of speed was similar to a machine. The length of the wires that are connected to the modules will be increased due to the fact that they are through. Before the electronic impulses could be sent via the device, the circuit had to first slow down the device. When Jack Kilby and Robert Noyce built the integrated circuit, they solved this issue by combining all of the components of the semiconductor material into a single block. This allowed the integrated circuit to function (monolith). This is necessary in order to complete the loops and make the manufacturing process more efficient. This gave birth to the concept of medium-scale convergence in the beginning of the 1960s, which is when all of the components are placed on a single silicon wafer. (LSI) and VLSI, which incorporate hundreds of millions (109) of transistors on a single chip, were created by (MSI) in the late 1960s and then again in the 1970s and 1980s. (MSI) was a pioneer in the field of integrated circuitry.

Two transistors were included on the first semiconductor devices. Eventually, additional transistors were added, which resulted in the progressive creation of a variety of different features or devices. These developments occurred as a direct result of the steady addition of more transistors. Because the first integrated circuits only required a small number of parts—perhaps 10 diodes, transistors, resistors, and condensers—it was possible to build one or more logic gates on a single component. This was made possible by the fact that integrated circuits only required a small number of components. The advancement of technology has led to the creation of systems that include hundreds of logic gates. These systems are now referred to as medium-scale integration (MSI), while they were formerly known as small-scale integration (SSI). More advancements were made possible as a result of more sophisticated integration (LSI), which refers to devices that have at least one thousand logic gates. The fact that modern microprocessors have hundreds of unique transistors and many millions of gates is indicative of how far technology has progressed since then.

A. Engineering for SSI

The first integrated circuits had a small number of transistors. The term "small-scale integration" (SSI) refers to the use of logic gates in binary circuits with tens-numbering transistors; early linear integrated circuits, such as the Plessey SL201 or Philips TAA320, featured just two transistors. Rolf Landauer, an IBM researcher, was the first to use the terms SSI, MSI, VLSI, and ULSI when formulating the scientific definition.

B. Engineering for MSI

The next phase in the process of building integrated circuits was the invention of "small integration" (MSI) modules in the late 1960s. These modules each included several hundred transistors and were the next step in the process of building integrated circuits. The fact of the matter is that even if the cost of production was comparable to SSI, it would still be preferable to produce more complex devices with smaller circuit boards, less labor-intensive assembly (due to fewer individual components), and a number of other advantages. This is because of the fact that the reality of the situation is as follows.

C. LOW Scale Inclusion

In the middle of the 1970s, a phenomenon known as "wider convergence," which consisted of tens of thousands of transistors on a single chip, evolved as a response to the same economic forces. In the early 1970s, the production of integrated circuits started in small numbers. At that time, the first microprocessors, computer chips, and 1K-bit RAMs were among the integrated circuits that were created. The first true 10,000-transistor LSI circuits for device huge memory and second-generation microprocessors were invented around 1974.

Using very large scale integration (VLSI), an effort was originally made to calibrate and define various degrees of specific integration. developed concepts like as (ULSI). However, the many doors and transistors that can be seen on modern technology serve as a perfect representation of these wonderful distinctions. Conditions that are more stringent than the VLSI convergence requirements are utilized much less often now. 2008 was the year when billion-transistor processors were available on the commercial market. The manufacture of semiconductors progressed, and new 65 nm technologies were discovered, which led to an increase in its popularity. Recent designs make advantage of robust and independent transistor logic synthesis, which contributes to an increase in the complexity of the logic implementation that is produced. In addition, various hand-crafted high-performance logic blocks, such as the static-random access memory (SRAM) cell, have been designed in order to get the greatest possible output.

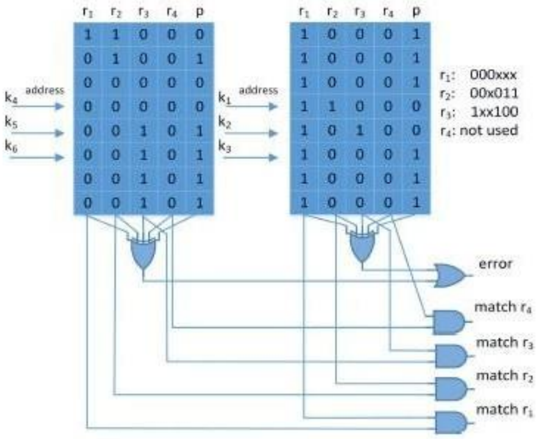
2.FPGA-BASED TCAM IMPLEMENTATIONS

When working with FPGAs, the implementation of TCAMs may primarily be done in one of two ways. The first thing that has to be done is to build the TCAM cells and match lines by making use of the flip-flops and logic facilities provided by the FPGA. The second option is to make advantage of the block memory that is available on the FPGA.

The bits of the rules are first held by flip-flops since this is the most convenient option. As was mentioned before, there are three distinct values that may be assigned to each bit: 0, 1, and x. For instance, a flip-flop can be used to store whether a bit is 0 or 1, and a second flip-flop, which functions as a mask and is set when the bit is do not care, can be used to record the result of the first flip-operation. flop's Both flip-flops can be used together to record the result of an operation. After then, programmable circuitry may be used in order to carry out the comparison with the key. Because of the significant amount of resources that are needed for each rule, this strategy cannot be utilized to construct enormous TCAMs with tens of thousands of rules that are longer than 100 bits and that function quickly.

The second possibility involves making use of the FPGA's built-in memories, which are known as embedded memories. In order to do this, the key is segmented into more manageable chunks of b bits. After then, a rule might be reproduced by making use of a 1-bit memory that has 2b places for each block. When looking for a key, all of the memory is accessible by utilizing the bits that correspond to the key; a match is found if all of the locations read contain a one. In most cases, k rules can be implemented with the use of a memory that has k bi t locations for each block. An illustration is the most effective method for conveying this point. Consider a key that is comprised of two blocks that are each composed of three bits, bringing the total number of bits to six. Then, a TCAM consisting of four rules might be used, as seen in Figure 1. As can be seen, each memory has a width of 4 bits and a total of 23 places, which equals 8 different storage spots. The three bits at the very top of the key may be used to access the memory that is placed the furthest to the left, while the three bits at the very bottom of the key can be used to access the other memory. When reading data from memory, these bits are necessary for determining the address of the memory location being read from. Figure also demonstrates the rules that are kept in each individual bit's storage space.

1. Let's have a look at the results of a key search for the number 000011. We would go closer and closer to the commencement of the story. The address on the memory on the far left reads 1100, and the address on the memory on the far right reads 011, with the four position reading 1100. Following the AND operation, the only rules that would be satisfied are r1 and r2. When we examine the rules in more detail, we see that the rules (r4) that are not being utilized have zeros in every memory location and location. This is visible when we examine the rules more carefully. The number of ones that are available in a specific memory for the remaining rules is determined by the amount of x bits a rule has on the address bits that are used as the memory's key bits. A one is stored in the one position if there are no x bits; whenever there are one or two x bits, the two locations store a one; whenever there are three or more x bits, the four places store a one; and so on. There will typically be a total of 2n x ones on the memory if there are n x bits that have the value x.



r_1	r_2	r_3	r_4	p
1	1	0	0	0
0	1	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	1	0	1
0	0	1	0	1
0	0	1	0	1
0	0	1	0	1

r_1	r_2	r_3	r_4	p
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	0	0	0
1	0	1	0	0
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1

r_1: 000xxx
r_2: 00x011
r_3: 1xx100
r_4: not used

Fig.1 Parity protected TCAM with 6-bit keys and four rules emulated usingtwo SRAMs.

Now take into consideration the expenses associated with the implementation, keeping in mind that each block must have b bits of SRAM memory and includes b bits of a rule. Using this strategy will result in a cost of 2b/b for each SRAM bit required for the TCAM bit [13]. Therefore, it would seem that lower values of b are more effective. However, this is not quite accurate since the amount of mental effort required to piece together the structure grows in proportion to the number of bricks used. Be aware that a significant amount of data stored in a physical memory may be partitioned into a number of blocks, each of which may execute via different components of a rule. At that point, more memory accesses are required to complete a search operation; however, it is possible to prevent this by using multiport memories or running the memory at a quicker speed [16].

Memory resources for Xilinx FPGAs come in the form of either lookup table random access memories (LUTRAMs) or basic random access memories (BRAMs). The early ones are typically compact with 32 or 64 places and are constructed using the same lookup tables (LUTs) that are used to implement the logic. BRAMs, on the other hand, are bigger and can store up to 36 k bits. They are capable of being set up with a variety of word sizes, the greatest of which is 72 bits, which is equivalent to 512 locations. As a direct consequence of this, LUTRAMs have a cost per bit that is much lower (25/5) than BRAMs do (29/9). On the other hand, compared to LUTRAMs, BRAMs have access to a greater number of total memory bits.

An essential finding for the safety of SRAM-based TCAM implementations is that just a few permutations of all of the potential values are utilized, and the contents of the SRAMs are decided by the rules that have been stated. This would seem to imply that the contents of the SRAM had some kind of built-in redundancy that may be used to safeguard the memories. This concept will be further explored in the next paragraphs of this short text.

An Energy-Efficient SRAM-Based TCAM on FPGA: Ternary content-addressable memory (TCAM) chooses a word from the ternary data it has saved depending on the information contained inside the word. Following completion of one cycle, the address of the matched word is obtained by carrying out a parallel comparison of the search key and each of the TCAM words that have been stored. The circuitry of a TCAM cell is responsible for storing, as well as comparing, three different states. These states are 0, 1, and the don't care state x. A priority encoder and a collection of TCAM cells make up the TCAM architecture (PE). Each TCAM cell has both a comparison circuitry and two SRAM cells, each of which may store a ternary bit. Both of these components are responsible for storing the bit. In the search technique, both search lines (also known as SLs) and match lines are used (MLs).

The SLs give the TCAM words' matching cells with search key bits in order to match them. The MLs are used to illustrate the comparison findings for each each TCAM word. If there is more than one TCAM word that successfully matches the search key, the PE will choose the address that has the greatest priority among the matching addresses. Figure 1 may depict an example 4 3 TCAM design. Native TCAM is a kind of TCAM that was built specifically for an application as an integrated circuit system (ASIC).

TCAM is used in a variety of different systems, including look-up tables in networking routers [1, 2], translations-look-aside buffers (TLB) caches in microprocessors [3, 4], database accelerators in big-data analytics [4,5], filters for storing signature patterns in the Internet of Things [6, 7], and local binary patterns recognition systems in image processing and DNA sequence matching [8, 9].

However, because of the specialized bit comparison circuitry, the memory density of the native TCAM cells is decreased. Additionally, because of the high degree of parallelism in the system, native TCAM is both costly and energy-intensive. In addition, the native TCAMs that are built into ASICs have a restricted number of configurations, which hampers their capacity to adapt to evolving market demands and prospective TCAM application trends that may emerge in the future.

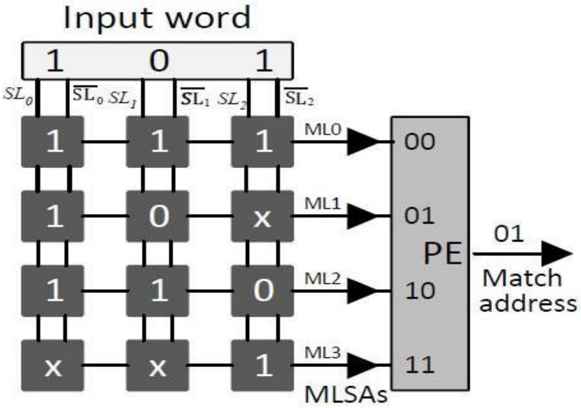


Figure 2 A 4 _ 3 TCAM: (MLSAs: Match line sense amplifiers)

The ability of modern field-programmable gate arrays, often known as FPGAs, to enable massive parallelism in addition to flexibility via on-the-fly reconfiguration makes them an appealing choice for the development of new systems. This is because tremendous progress has been made in CMOS technology, which has led to this outcome. Block RAMs, often known as BRAMs, are a prevalent kind of embedded memory that is used in contemporary SRAM-based FPGA devices such as the 16-nm Xilinx Virtex Ultra SCALE FPGA. These BRAMs can store large amounts of data. BRAMs are built utilizing silicon substrates, and they are capable of high speeds while using just a little amount of power.

Integrated memory BRAMs on current SRAM-based FPGAs are preferred due to the need for rapid, adaptable (reconfigurable), and adaptive (easy for integration) TCAM design. This is because integrated memory BRAMs are easier to integrate. SRAM is used to implement TCAM in FPGAs. This is done by addressing SRAM with the contents of TCAM and storing data for the whole of the TCAM table in SRAM. The existence as well as the address of a single TCAM pattern is stored in each word of SRAM. currently available SRAM-based

TCAMs on FPGAs have a greater energy use than other types of memories because it takes an excessive amount of power to activate all of the SRAM memory that is required for a lookup. For example, the BRAMs on the FPGA were utilized to construct 89 kb and 150 kb TCAM tables via the SRAM-based TCAM design approaches, which required a total of 2.5 Wand and 3.2 Wand, respectively. These table sizes were accomplished by employing the SRAMs. When capacity rises, the already high power consumption of SRAM-based TCAM devices becomes even more problematic.

3. SOFT ERRORS IN SRAM

The sensitivity of semiconductor devices to radiation has significantly grown as a result of the evolution of technology. SRAM arrays are often the most densely packed circuitry on a chip because they make full use of very small cells. Due to the high bit count, there is a greater chance that an ionizing particle would collide with a sensitive node in the array, therefore erasing the data that was previously stored. The lowest layout dimensions reduce the capacitance of the storage nodes and, as a result, the critical charge Qcrit that might be introduced by radiation and cause a disturbance in the SRAM cell.

The Qcrit is reduced even more as a result of the falling supply voltages. Radiation may cause data mistakes, which makes it difficult to construct dependable SRAM arrays using nano scaled technologies. These variables contribute to radiation-induced data errors. Radiation may be the major cause of localized ionization events in semiconductor devices or it may be the outcome of a secondary process. A small number of these radiation-induced events create a enough number of electron-hole pairs to cause damage to the storage nodes of SRAM cells. A "soft" mistake is what we mean when we say anything like this while we're angry. A disturbance of this magnitude might cause a data mistake, although the device structures themselves are not irreparably harmed. Though the voltage disturbance on one of an SRAM cell's storage nodes is lower than the node's noise margin, the SRAM cell will continue to function correctly and will maintain the integrity of the data it stores even if the disturbance occurred.

A "soft" mistake will occur, however, if the noise buffer of a cell is insufficient to survive the disruption induced by ionizing radiation. This might lead to a cell malfunctioning. In the late 1970s, soft errors were identified as a potential issue for dynamic random-access memories (DRAMs) that used planar storage capacitors. The charge that was stored in these capacitors was kept in two-dimensional p-n junctions that covered a huge region. Early DRAM cells were very prone to soft mistakes because of the high radiation-induced charge collection efficiency of the large planar reverse-biased connections. Technology advances and attempts to lower the high soft error rates and poor pause/refresh time ratios of DRAMs (SER) (SER)

As a result, there was an increased need for portable three-dimensional storage capacitors. Because of the decreased volume of the p/n junction, the newly developed 3D capacitors demonstrated a junction collection efficiency that was much lower than that of the older 2D planar capacitors. Despite the fact that a DRAM cell's Qcrit will decrease as a result of VDD scaling, the storage capacitor's aggressive junction volume growth will more than compensate for this loss. Because of this, the SER of a DRAM bit cell drops by about 4 times with each new generation of technical advancement.

This decrease in DRAM SER, however, is being countered by the rapidity with which the system-level DRAM bit count is growing. Because bigger DRAM arrays are statistically more prone to generate soft mistakes, the resultant DRAM SER at the system level has remained impressively consistent over numerous recent technological generations. This is despite the fact that larger DRAM arrays are statistically more likely to create soft errors. The feedback mechanism that safeguards the state of an SRAM cell allowed early SRAMs to be more resistant to the effects of soft faults than their DRAM counterparts. The critical charge of an SRAM cell is impacted in two different ways: first, by the restoring current of the pull-up or driver transistors, and second, by the capacitance of the storage node.

As a result of scaling in technology, the size of the SRAM cell and, therefore, the junction area of the storage nodes are both decreasing (Fig S1). Additionally, the capacitance of the storage node was decreased, and there is a possibility that the leakage at the cell junction was lowered. A more aggressive scaling of the VDD was the outcome of switching from constant voltage scaling to constant electric field scaling. The combined effect of these two elements is to lower the Qcrit, which in turn leads to an increase in the likelihood of soft mistakes, which in turn leads to a rise in SER levels. With each new generation of technology, decreases in cell collecting efficiency brought on by decreases in cell depletion volume were compensated for by improvements in storage node and VDD capacitance. These changes occurred with each new generation of technology.

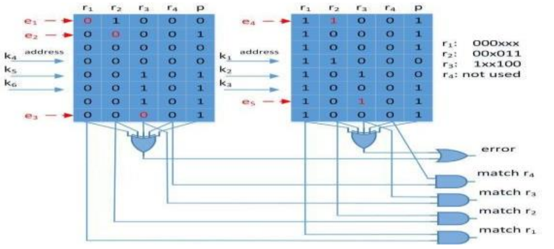
4. ERROR DETECTION AND CORRECTION INSRAM – BASED TCAMS

The emulated TCAM memory is equipped with a protection system that employs a per-word parity bit in order to identify instances of single-bit mistakes. When an error is found, an attempt is made to fix it by using the built-in redundancy of the memory contents. This is done in the event that the fault cannot be corrected. Figure 2 depicts the implementation of the parity protection, where the letter p stands for the parity bit. It is abundantly clear that in addition to the match signal, an error signal is also produced if there is a discrepancy between the parity that has been saved and the one that has been recomputed. This parity check is the industry standard, and it can detect any and all single-bit defects [5]. Error detection on each and every access is very necessary in order to avoid producing inaccurate search results.

Let's suppose for the time being that a specific word had a single-bit mistake and that the parity check was able to find it. In the event that a mistake is discovered, we can investigate the matter by looking at the information stored in the memory. A good place to start would be to read every word in the memory and make a note of the number of times each rule occurs in a single area. This would be a nice place to begin. Make use of that number to indicate the significance of the rule associated with that memory. Taking Fig. 2's leftmost memory as an example, r1 would have a weight of 1, r2 would have a weight of 2, and r3 would have a weight of 4. It's possible that this will assist us in locating the incorrect bit, given that the weight of an error-free rule for an 8-position memory can only be 0, 1, 2, 4, or 8. Let's concentrate on the cases of single-bit mistake shown in Fig. 3 so that we can have a more in-depth conversation about the process of error repair. For example, e3 reduces the weight of r3 in the leftmost memory from 4 to 3, making no other modifications. After finding the parity mistake, we would determine that the bit in register 3 (r3) is the one that is incorrect, and then we would correct it since 3 is not a legal value. It is possible that using this method will be beneficial for rules which either have weights greater than two or have two or more "x" bits on the key bits relating to the memory in question. On the other hand, when it comes to regulations that have a smaller weight, it's possible that examining the weight alone won't be sufficient. Now, let's think about a rule that has a weight of two. After then, there will be a blunder that causes a 0 to be changed into a 1.

In order to rectify the situation, the value of the weight is going to be raised to three. It is impossible to repair the mistake when a one is converted to a zero (as in e2) since the new weight would then be a legal integer. This occurs when the error occurs in the calculation for e2. However, the fact that there are only two spots with a one makes the occurrence of this event less probable. If we are going to take the guideline for weight one as our standard, then an error that changes another bit to one will result in a weight of two, which is likewise acceptable. However, not every possible combination of weights two can be implemented. When looking at e4, this is really obvious to see. In this scenario, the key values 000 and 011 would be similar to r2 values of one, which are not equivalent to a legitimate rule. r2 values of one are not equivalent to a valid rule. Most of the time, the search will not be successful until the place in question matches a key value that is one step removed from the initial value. On the other hand, a mistake that resets a weight one rule's position from one to zero may be remedied by confirming that the rule has zero weight on the other memory. This corrects the error that caused the position to be reset.

In this scenario, the rule would be deactivated, and there would be no indication that the bit was incorrect. If this was not the case, the mistake was fixed, and the weight of the rule was increased to 1. Last but not least, an inaccuracy in a rule with a weight of 0 may also be rectified by evaluating the impact of the rule on the other memories in order to determine its relative importance.



r1	r2	r3	r4	p
0	1	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	1	0	1
0	0	1	0	1
0	0	1	0	1
0	0	0	0	1

r1	r2	r3	r4	p
1	1	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	0	0	0
1	0	1	0	0
1	0	0	0	1
1	0	1	0	1
1	0	0	0	1

Fig.3. Examples of single-bit errors on a parity protected TCAM with 6-bit keys and four rules emulated using two SRAMs.

The section that came before it demonstrated how several single-bit error patterns may be resolved by making advantage of the redundancy that was already built into the memory contents. Now that we have it figured out, let's calculate out how many single-bit error patterns there are in a memory that has 2b places that can be repaired for each weight.

1. There is no weight; errors in patterns may be corrected.

2. Weight one: Everyone, with the exception of those who set a bit to one for a place that has an address at a distance of one; the sum of these two values is equal to 1 b/2b.

3. The second weight entails that all patterns, with the exception of the two that set a position with a one to a zero, are able to have their values altered. This is the same as 12 divided by 2b.

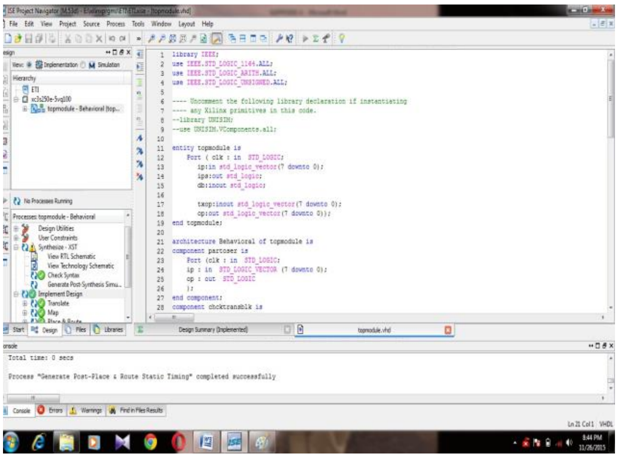
All patterns may be corrected up to and including level four when using weights four and above.

The vast majority of the erroneous patterns have been, without a doubt, fixed. This idea is made more understandable by referring to Table I, which provides an overview of the proportion of designs that may be modified to accommodate columns with varying weights. The only circumstances in which all mistakes cannot be rectified are those involving weights one and two; in such circumstances, the percentage will be extremely close to 100% when b is big. The only other circumstance is when all faults occur in weights three and four. Table II presents, for each of many possible values of b, the proportion of mistakes that are amenable to correction. Even with a relatively little amount of memory (b = 5, which equates to 32 locations), the error coverage is rather near to 90%, even in the worst case scenario, as can be shown. The coverage for bigger memories is more than 95% and comes very close to reaching 100%. As an example, the coverage for b = 9 is more than 98% even in the most dire of circumstances. This demonstrates how successful the suggested method is at repairing single-bit mistakes in situations where the parity bit is used to ensure security for the memory.

5. SYNTHESIS AND IMPLEMENTATION OF THE DESIGN

Before the design can be implemented on the prototype board or checked for accuracy using functional simulation, it must first be synthesized and implemented. This may be done at any time throughout the design process. While the top-level VHDL file is accessed (by double-clicking that file in the HDL editor window on the right section of the Project Navigator), the implement design option may become accessible in the process view of the project. This occurs when the project view is in the Module view. In addition, the process view displays the available choices for the Generate Programming File and Design Entry utilities. In the event that there are any user limitations, they might be dealt under the former, while the latter will be considered in a later section.

Double-clicking the option in the Processes box that is titled "Synthesize Design" will cause you to synthesize the design. To get started, double-click the "Implement design" option that is located in the Processes box. There are several steps involved, including Translate, Map, Place & Route, and others. Any of these stages that were either impossible to finish or were done in an unacceptable manner will have a cross superimposed over them. In such case, a check mark will be added after each activity to show that it was completed without any problems. If everything goes according to plan, there will be a check mark shown next to the choice to Implement Design. In the event that there are any cautions, a checkmark will appear in front of the option to indicate whether or not there are any warnings. The Console window, which may be seen at the bottom of the Navigator window, displays any warnings or problems that may have occurred. Because each of these markers is deleted every time the design file is saved, you will need to start the compilation process from scratch.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity topmodule is
    Port ( clk : in  STD_LOGIC;
        ip:in std_logic_vector(7 downto 0);
        ips:out std_logic;
        db:inout std_logic;

        txop:inout std_logic_vector(7 downto 0);
        op:out std_logic_vector(7 downto 0));
end topmodule;

architecture Behavioral of topmodule is
component partoser is
    Port (clk : in  STD_LOGIC;
    ip : in  STD_LOGIC_VECTOR (7 downto 0);
    op : out  STD_LOGIC
    );
end component;
component chcktransblk is
```

```
Total time: 0 secs

Process "Generate Post-Place & Route Static Timing" completed successfully
```

FIG.4 Implementing the Design (snapshot from Xilinx ISE software)

The schematic representation of the synthesized VHDL code may be examined in the Process Window by selecting Synthesize-XST from the menu and then selecting View RTL Schematic by clicking it twice. This is a useful way for troubleshooting the code and would be beneficial in the event that the output on the prototype board did not meet our needs.

When you double click, the top-level module will open, and all it will show you are the module's inputs and outputs, as seen below.

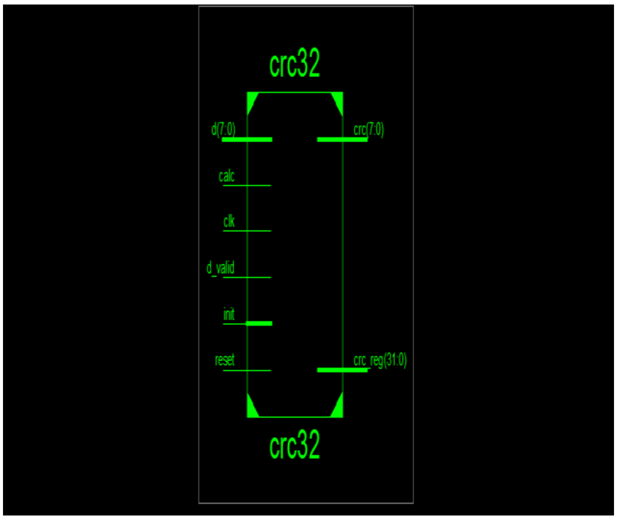


Fig 5 Top Level Hierarchy of the design

When you double click on the rectangle, the underlying logic that has been established is revealed, as demonstrated in the following example. XilinxISE's implementation of the logic included in the VHDL code The ETI simulation has been finished successfully with the help of the model Xilinx simulator.

A. Simulation and synthesis has been carried out with XILINXISE:

Before the design can be implemented on the prototype board or checked for accuracy using functional simulation, it must first be synthesized and implemented. This may be done at any time throughout the design process. The implement design option becomes accessible in the process view when the top-level VHDL file is opened (by double-clicking that file in the HDLeditor window located on the right side of the Project Navigator) and the project is shown in the Module view.

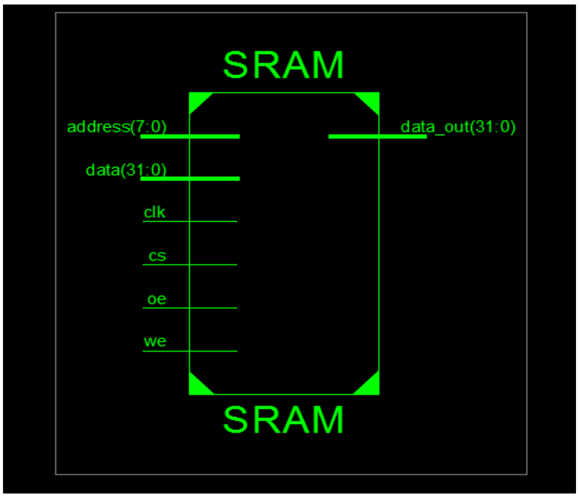
In addition, the process view displays the available choices for the Generate Programming File and Design Entry utilities. In the event that there are any user limitations, they might be dealt under the former, while the latter will be considered in a later section. Double-clicking the option in the Processes box that is titled "Synthesize Design" will cause you to synthesize the design. In order to put the plan into action,

To implement the design, double-click the corresponding option in the Processes pane. There are several steps involved, including Translate, Map, Place & Route, and others. Any of these stages that were either impossible to finish or were done in an unacceptable manner will have a cross superimposed over them. In such case, a check mark will be added after each activity to show that it was completed without any problems. If everything goes according to plan, there will be a check mark shown next to the choice to Implement Design.

One may be able to see the warnings, if there are any! Put a checkmark in front of the choice to let consumers know that there are certain precautions they should take. The Console window, which may be seen at the bottom of the Navigator window, displays any warnings or problems that may have occurred. When the file containing the design is saved, each of these markers is removed, which means that a new compilation will need to be done.

If you double click the View RTL Schematic option in the Synthesize-XST menu in the Process Window, you will be able to view the schematic diagram that corresponds to the VHDL code that was synthesized. If the output on the prototype board did not match our standards, then this would be a helpful approach for debugging the code.

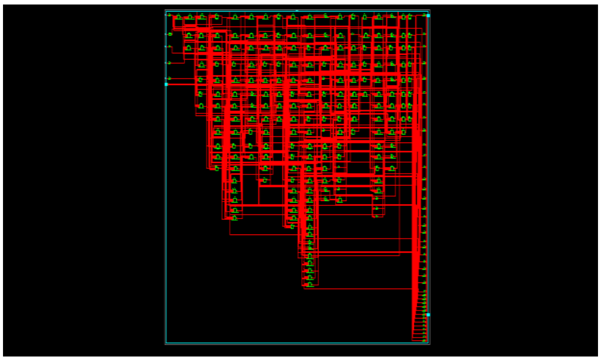
Simply clicking the plus symbol that is located next to the Modelsim simulator Tab in the Processes window will allow you to make it larger (while making sure the test bench file in the Sources window is chosen). You need to click the Simulate Behavioral Model button a total of two times. There is a good chance that you may run across a compiler error. When asked whether you want to quit the simulation, choose "NO" from the drop-down menu. There is nothing to be afraid of in this situation. As a direct result of this, ModelSim ought to begin running. Continue to watch it until it stops moving.

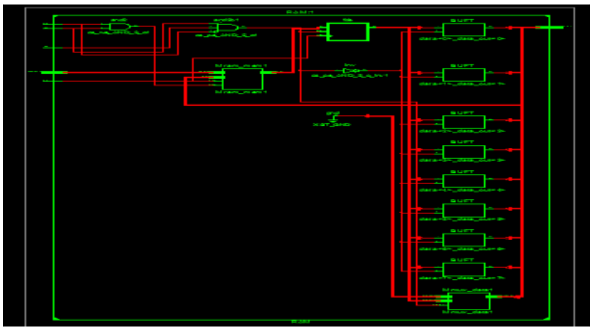


B. Technology schematic of crc Block replaced by XOR gate in ETI architecture

6. EXPERIMENTAL RESULTS

RTL





CONCLUSION

In this paper, a method is proposed for securing the SRAMs that are present on FPGAs and serve the purpose of simulating TCAMs. The technique is founded on the discovery that while not all values may be obtained in those SRAMs, there are certain degrees of duplication in the memory contents. This observation is the foundation for the method. When the memory is secured with a parity bit to detect faults, this redundancy is employed to rectify the majority of single-bit error patterns so that the memory can function properly. Because the suggested method reduces the amount of resources required to safeguard memory by a large margin, it is an attractive choice for implementation in systems in which dependability is essential but resources are in short supply.

This condensed description of the underlying idea may be used for a variety of memory architectures. For example, by routinely evaluating the accuracy of an unprotected memory, it might be used to locate problems in the memory and determine how to fix them. It is possible to utilize it to rectify numerous bit mistakes in the memory when it is protected by a more robust code that is able to notice a string of faults in succession. For instance, double-bit error patterns for a memory that is guarded by an SEC code may be identified and corrected by the use of the built-in redundancy of the memory's contents.

REFERENCES

1. N. Kanekawa, E. H. Ibe, T. Suga, and Y. Uematsu, Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances. New York, NY, USA: Springer-Verlag, 2010.

2. J. L. Autran et al., “Soft-errors induced by terrestrial neutrons and natural alpha-particle emitters in advanced memory circuits at ground level,” Microelectron. Rel., vol. 50, no. 9, pp. 1822–1831, Sep. 2010.

3. L. Silburt, A. Evans, I. Perryman, S. J. Wen, and D. Alexandrescu, “Design for soft error resiliency in Internet core routers,” IEEE Trans. Nucl. Sci., vol. 56, no. 6, pp. 3551–3555, Dec. 2009.

4. Evans, S.-J. Wen, and M. Nicolaidis, “Case study of SEU effects in a network processor,” in Proc. IEEE Workshop Silicon Errors Logic-Syst. Effects (SELSE), Mar. 2012, pp. 1–7.

5. L. Chen and M. Y. Hsiao, “Error-correcting codes for semiconductor memory applications: A state-of-the-art review,” IBM J. Res. Develop., vol. 28, no. 2, pp. 124–134, Mar. 1984.

6. K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: A tutorial and survey,” IEEE J. Solid-State Circuits, vol. 41, no. 3, pp. 712–727, Mar. 2006.

7. F. Yu, R. H. Katz, and T. V. Lakshman, “Efficient multimatch packet classification and lookup withTCAM,” IEEE Micro, vol. 25, no. 1, pp. 50–59, Jan./Feb. 2005.

8. P. Bosshart et al., “Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN,” in Proc. ACM SIGCOMM, 2013, pp. 99–110.

9. Syafalni, T. Sasao, and X. Wen, “A method to detect bit flips in a soft-error resilient TCAM,” IEEETrans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 6, pp. 1185–1196, Jun. 2018.

10. S. Pontarelli, M. Ottavi, A. Evans, and S. Wen, “Error detection in ternary CAMs using Bloom filters,”in Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE), Mar. 2013, pp. 1474–1479.