

ER-TCAM: A Soft Error Resilient SRAM-based Ternary Content Addressable Memory for FPGAs

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Electronics and Communication Engineering

by

S. LOKESH (37130353)

S. SADIQ BASHA (37130373)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING
SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

APRIL - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI - 600 119

www.sathyabama.ac.in



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of S. LOKESH (Reg.No: 37130353) and S. SADIQ BASHA (Reg.No: 37130353), who carried out the project entitled “**ER-TCAM: A Soft Error Resilient SRAM-based Ternary Content Addressable Memory for FPGAs**” under my supervision from September 2020 to April 2021.

Internal Guide

Mr. K.T ILAYARAJAA, M.E., (Ph.D)

Head of the Department

Dr.T.RAVI, M.E., Ph.D

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, S.LOKESH (Reg.No: 37130353) and S.SADIQ BASHA (Reg.No: 37130373) hereby declare that the Project Report entitled **ER-TCAM: A Soft Error Resilient SRAM-based Ternary Content Addressable Memory for FPGAs** is done by us under the guidance of Mr. K.T.ILAYARAJAA, M.E.,(Ph.D) and at SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

DATE: 1.

PLACE: 2.

SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr.N.M.NANDHITHA, M.E., Ph.D. Dean, School of ELECTRICAL AND ELECTRONICS** and **Dr.T.RAVI, M.E., Ph.D. Head of the Department, Dept. of ELECTRONICS AND COMMUNICATION ENGINEERING** for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide of **Mr. K.T.ILAYARAJAA, M.E., (Ph.D)** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the Department of **ELECTRONICS AND COMMUNICATION ENGINEERING** who were helpful in many ways for the completion of the project.

ABSTRACT

Static random-access memory (SRAM)-based ternary content-addressable memory (TCAM) on field-programmable gate arrays (FPGAs) is used for packet classification in software-defined networking (SDN) and Open Flow applications. SRAMs implementing TCAM contents constitute a major part of a TCAM design on FPGAs, which are vulnerable to soft errors. The protection of SRAM-based TCAMs against soft errors is challenging without compromising critical path delay and maintaining a high search performance. This brief presents a low cost and low-response-time technique for the protection of SRAM-based TCAMs. This technique uses simple, single-bit parity for fault detection which has a minimal critical path overhead. This technique exploits the binary-encoded TCAM table maintained in SRAM-based TCAMs for update purposes to implement a low-response-time error-correction mechanism at low cost. The error-correction process is carried out in the background, allowing lookup operations to be performed simultaneously, thus maintaining a high search performance. The proposed technique provides protection against soft errors with a response time of 293 ns, whereas maintaining a search rate of 222 million searches per second on a 1024×40 size TCAM on Artix-7 FPGA.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	v
	LIST OF ABBREVIATIONS	viii
	LIST OF FIGURES	ix
	LIST OF TABLES	x
1	INTRODUCTION	1
	1.1 WHAT IS VLSI	1
	1.1.1 DEALING WITH VLSI CIRCUITS	1
	1.1.2 INTRODUCTION TO VHDL	2
	1.1.2.1 LIMITATIONS OF TRADITIONAL PROGRAMMING LANGUAGE	2
	1.1.3 ELECTRONIC DESIGN AUTOMATION	4
	1.1.4 FPGA	5
2	LITERATURE SURVEY	8
	2. 1 LITERATURE SURVEY	8
	2. 2 LITERATURE SUMMARY	11
	2.3 PROBLEM DEFINITION	11
	2.4 OBJECTIVES	12
	2. 5 EXISTING SYSTEM	12
	2.6 PROPOSED SYSTEM	12
3	AIM AND SCOPE OF PRESENT INVESTIGATION	13
	3.1 AIM	13
	3.2 SCOPE	13
4	METHODS AND ALGORITHM USED	14
	4.1 INTRODUCTION	14
	4.2 PROPOSED SYSTEM	17

	4.3 SOFTWARE REQUIREMENTS	21
	4.3.1 MODELSIM-ALTRA	21
	4.3.1.1 ASSUMPTIONS	21
	4.3.1.2 MODEL SIM INTRODUCTION	21
	4.3.1.3 BASIC SIMULATION FLOW	21
	4.3.1.4 BASIC SIMULATION FLOW OVERVIEW	22
	LAB	
	4.3.2 PROJECT FLOW	22
	4.3.3 MULTIPLE LIBRARY FLOW	23
	4.3.4 DEBUGGING TOOLS	24
	4.3.5 BASIC SIMULATION	25
	4.3.5.1 INTRODUCTION	25
	4.3.5.2 DESIGN FILES FOR THE LESSON	25
	4.3.6 XILINX ISE	25
	4.4 PREREQUISITE	26
	4.4.1 SIMULATION LIBRARIES	26
	4.4.2 SETTING UP ENVIRONMENT	26
	4.4.3 SIMULATION LIBRARY COMPILATION	27
	WIZARD	
5	RESULTS AND DISCUSSION	29
	5.1 RESULT	29
	5.1.1 TCAM 64*40	31
	5.1.2 512*28	33
	5.1.3 512*40	35
	5.1.4 1024*40	37
	5.1.5 2048*40	39
	5.2 DISCUSSION	39
6	SUMMARY AND CONCLUSION	40
	REFERENCES	41

LIST OF ABBREVIATIONS

VLSI	-	VERY LARGE-SCALE INTEGRATION CIRCUITS
HDL	-	HARDWARE DESCRIPTION LANGUAGE
VHDL	-	VHSIC HARDWARE DESCRIPTION LANGUAGE
VHSIC	-	VERY HIGH-SPEED INTEGRATED CIRCUITS
IEEE	-	INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS
VITAL	-	VHDL INITIATIVE TOWARDS ASK LIBRARIES
RTL	-	REGISTER TRANSFER LANGUAGE
WAVES	-	WAVEFORM AND VECTOR EXCHANGE TO SUPPORT DESIGN AND TEST VERIFICATION
PLA	-	PROGRAMMABLE LOGIC ARRAY
PAL	-	PROGRAMMABLE ARRAY LOGIC
CPLD	-	COMPLEX PROGRAMMABLE LOGIC DEVICES
CLB	-	CONFIGURABLE LOGIC BLOCKS
ISE	-	INTEGRATED SOFTWARE ENVIRONMENT
EDA	-	ELECTRONIC DEVICES AUTOMATION
FPGA	-	FIELD PROGRAMMABLE GATE ARRAY
PDA	-	POWER DELAY PRODUCT
EDD	-	ERROR DETECTION DELAY

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
1.1	SUMMARY OF VHDL DESIGN FLOW	4
1.2	FPGA ARCHITECTURE	6
1.3	EXAMPLES OF FPGA PACKAGES	7
4.1	SIMPLIFIED IMPLEMENTATION OF TCAM USING SRAM	16
4.2	SIMPLIFIED EXAMPLE OF THE PROPOSED ER- TCAM	18
4.3	PROPOSED ER-TCAM ARCHITECTURE FOR ERROR DETECTION	18
4.4	PROPOSED ER-TCAM ERROR-CORRECTION MODULE	20

LIST OF TABLES

TABLE No.	TABLE NAME	PAGE No.
5.1	DISCUSSION	39

CHAPTER 1

INTRODUCTION

INTRODUCTION TO VLSI DESIGN:

1.1 What is VLSI?

VLSI stands for "Very Large-Scale Integration". This is the field which includes pressing increasingly more rationale gadgets into more modest and more modest territories. VLSI, circuits that would have taken board folds of room would now be able to be placed into a little space not many millimeters across! VLSI circuits are all over the place ... your PC, your vehicle, your pristine best in class computerized camera, the cellphones, and what have you. This includes great deal of aptitude on numerous fronts inside a similar field, which we will take a gander at in later areas.

1.1.1 Dealing with VLSI Circuits

The manner in which ordinary squares like hooks and entryways are carried out is unique in relation to what understudies have seen up until this point, however the conduct stays as before. All the scaling down includes new interesting points. A great deal of thought needs to go into genuine executions just as plan.

Circuit Delays: Enormous confounded circuits running at exceptionally high frequencies have one major issue to handle - the issue of deferrals in proliferation of signs through entryways and wire in any event, for regions a couple of micrometers across! The activity speed is enormous to the point that as the postpones add up, they can really get tantamount to the clock speeds.

Power: Another impact of high activity recurrence is expanded utilization of force. This has two-overlay impact - gadgets burn-through batteries quicker, and heat dispersal increments. Combined with the way that surface zones have diminished, heat represents a significant danger to the Stability of the actual circuit.

Layout: Spreading out the circuit segments is task basic to all parts of gadgets. What's so unique for our situation is that there are numerous potential approaches to do this; there can be various layers of various materials on a similar silicon, there can be

various plans of the more modest parts for a similar segment and soon. The way we chose the format for the circuit segments determined our choice between the two. VLSI chip manufacturing may also be influenced by format, making it easier or more difficult to carry out the parts on the silicon.

1.1.2 Introduction to VHDL

A digital system can be described at different levels of the abstraction takes place and from different points of view. An HDL ought to steadfastly and precisely demonstrate and depict a circuit, regardless of whether previously fabricated or being worked on, from either the primary or social perspectives, at the ideal degree of deliberation. Since HDLs are designed as per the equipment, their semantics and use are altogether different from those of customary programming dialects.

1.1.2.1 Limitations of Traditional Programming Languages

There are several different types of PC programming languages, ranging from Frontend to C to Java. Lamentably, they aren't sufficient to show advanced equipment. To comprehend their constraints, it is helpful to analyze the advancement of a language. A programable language which is portrayed by its punctuation and semantics. The punctuation contains the linguistic guidelines used to compose a program, and the semantics is the "signifying" related with language develops. At the point when another scripting language is created, the creators first examination the attributes of the basic cycles and afterward create syntactic builds and their related semantics to demonstrate and communicate these qualities.

A sequential method is modelled in most common general-purpose programming languages, such as C. Operations are carried out in this method in a sequential order, one at a time. Since the outcome of one operation always depends on the outcome of another, the order in which operations are carried out cannot be changed at will. The sequential process model has two major benefits. At the abstract level, it helps the human thinking process to develop an algorithm step by step. The sequential method resembles the operation of a simple computer model at the implementation stage, allowing for efficient translation from an algorithm to machine instructions.

The attributes of advanced equipment, then again, are altogether different from

those of the successive model. A run of the mill advanced framework is regularly worked by more modest parts, with redid wiring that associates the info and yield ports of these parts. At the point when sign changes, the parts associated with the sign are actuated and a bunch of new tasks is started as needs be. These activities are performed simultaneously, and every activity will take a particular measure of time, which addresses the proliferation deferral of a specific part, to finish. After culmination, each part refreshes the estimation of the comparing yield port. In the unlikely event that the worth is changed, the yield sign will thus enact every one of the associated parts and start another round of tasks. This portrayal shows a few extraordinary attributes of advanced frameworks, including the associations of parts, simultaneous activities, and the idea of engendering deferral and timing. The successive model utilized the traditional programming languages are unable to capture the characteristics of computerized equipment, necessitating the use of unusual dialects (e.g., HDLs) that are designed to display advanced equipment.

VHDL incorporates offices for portraying legitimate construction and capacity of computerized frameworks at various degrees of deliberation, from framework level down to the entryway level. It is proposed, in addition to other things, as a demonstrating language for determination and recreation. We can likewise utilize it for equipment combination in the present event which we confine ourselves to a subset that can be naturally converted into equipment.

The Very High-Speed Integrated Circuits (VHSIC) software of the US government gave birth to VHDL. Over the course of the programme, it became clear that a common language for depicting the design and capability of synchronised circuits was needed (ICs). VHSIC Hardware Description Language (VHDL) was created as a result. It was consequently grown further under the support of the Institute of Electrical and Electronic Engineers (IEEE) and embraced as the IEEE Standard 1076, Standard VHDL Language Reference Manual, in 1987. This first standard variant of the language is frequently alluded to as VHDL-87.

Following the underlying delivery, different expansions were created to work with different plan and displaying necessities. These expansions are archived in a few IEEE principles:

- i. IEEE standard 1076.1-1999, VHDL Analog and Mixed Signal Extensions (VHDL-AMS): defines the extension for analog and mixed-signal modeling.
- ii. VHDL Mathematical Packages, IEEE standard 1076.2-1996, specifies additional mathematical functions for real and complex numbers.
- iii. Synthesis Packages, IEEE standard 1076.3- 1997, specifies Over a set of bits, arithmetic operations are performed.
- iv. VHDL Initiative towards ASK Libraries (VITAL), an IEEE standard 1076.4- 1995, describes a mechanism for adding precise timing information to ASIC cells
- v. VHDL Initiative towards ASK Libraries (VITAL), an IEEE standard 1076.6- 1999, describes a mechanism for adding precise timing information to ASIC cells.
- vi. 1 164- 1993 IEEE standard Standard Multivalve Logic System for VHDL Model Interoperability (std-logicJl64) introduces new data types for modelling multivalve logic.
- vii. IEEE standard 1029.1-1998, VHDL Waveform and Vector Exchange to Support Design and Test Verification (WAVES): defines how to use VHDL to exchange information in a simulation environment.

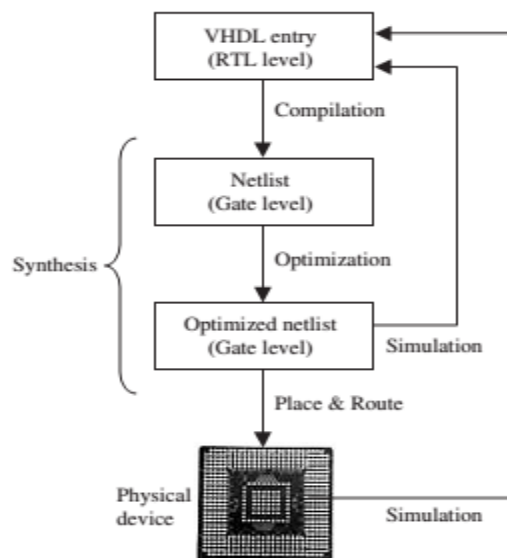


Fig:1.1:Summary of VHDL design flow

1.1.3 Electronic Design Automation

There are several EDA (Electronic Design Automation) tools available for circuit synthesis, implementation, and simulation using VHDL. A few devices (spot and

course, for instance) are requested as part of a seller's plan suite (for example, Altera's Quartus II, which allows VHDL code to be combined with other code). Altera's CPLD/FPGA chips, or Xilinx's ISE suite, for Xilinx's CPLD/FPGA chips). Different apparatuses (synthesizers, for instance), Aside from being requested as part of the plan suites, individual EDAs may also include it. organizations (Mentor Graphics, Synopsis, Simplicity, and so on are just a few examples.) Instances of the last gathering Leonardo Spectrum (a Mentor Graphics synthesiser), Simplify (a Simplicity synthesiser), and Modelsim (a Simplicity synthesiser) (a test system from Model Technology, a Mentor Graphics organization). The plans introduced in the provided book were combined onto CPLD/FPGA gadgets (informative supplement A) either from Altera else from Xilinx. The devices utilized were either ISE joined with ModelSim, MaxPlus II joined with Advanced.

Synthesis Software or Quartus II. Leonardo Spectrum was likewise utilized once in a while. Albeit distinctive EDA devices were utilized to carry out and test the models introduced in the plan, we chose to normalize the visual introduction of all reproduction charts. Because of its perfect appearance, the waveform manager of MaxPlus II was utilized. However, newer simulators like ISE p ModelSim and Quartus II, over a much broader set of features, which allow, for example, a more refined timing analysis. For that reason, those tools were adopted when examining the fine details of each design.

1.1.4 Field-Programmable Gate Array (FPGA)

A field-programmable gate array (FPGA) is a semiconductor device that can be arranged by the client or originator subsequent to assembling—thus the name "field-programmable". To program a FPGA, one should Using a rationale circuit map or source code in an equipment, they will show how they want the chip to work portrayal language (HDL). FPGAs can also be utilized to carry out any legitimate capacity that an application-explicit incorporated circuit (ASIC) could perform, however the capacity to refresh the usefulness subsequent to transportation offers benefits for some applications.

FPGAs contain programmable rationale segments called "rationale blocks", and the squares can be "wired together" in a pecking order of reconfigurable interconnects, similar to a one-chip programmable breadboard. Rationale squares may be built to perform complex combinational capabilities or to perform only simple rationale

entryways such as AND and XOR. In many FPGAs, the rationale hinders additionally incorporate memory components, which might be straightforward flip-flop or more complete squares of memory.

For some random semiconductor measure, FPGAs are generally slower than their fixed ASIC counterparts. They likewise draw more power, and for most of part accomplish less usefulness utilizing a given measure of circuit intricacy. However, their benefits incorporate a more limited chance to showcase, capacity to re-program in the field to fix bugs, and lower non-repeating designing expenses. Merchants can likewise take a center street by building up their equipment on normal FPGAs yet make their last form so it can at this point don't be altered after the plan has been submitted.

Field Programmable Gate Array (FPGA) devices were introduced by Xilinx in the mid-1980s. They differ from CPLDs in architecture, storage technology, number of built-in features, and cost, and are aimed at the implementation of high performance, large-size circuits.

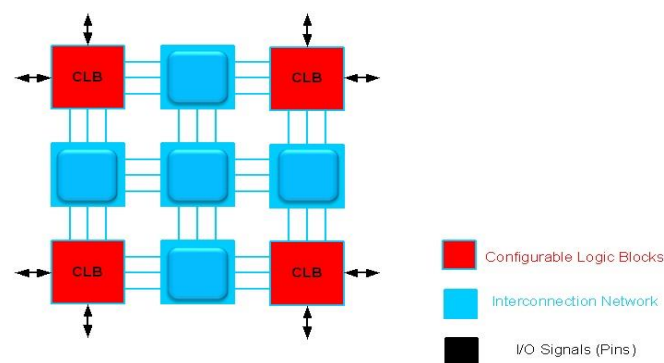


Fig:1.2: FPGA Architecture

The basic architecture of an FPGA is illustrated in figure 1.2. It consists of a matrix of CLBs (Configurable Logic Blocks), interconnected by an array of switch matrices.

The inside design of a CLB is unique in relation to that of a PLD. First, rather than executing SOP articulations with AND entryways followed by OR doors (like in SPLDs), its activity is ordinarily founded on a LUT (Lookup table). Moreover, in an FPGA the number of flip-flops is much more abundant than in a CPLD, thus allowing the construction of more sophisticated sequential circuits. Besides JTAG support and interface to diverse logic levels, other additional features are also included in FPGA

chips, like SRAM memory, clock multiplication (PLL or DLL), PCI interface, etc. Some chips also include dedicated blocks, like multipliers, DSPs, and microprocessors.

Another fundamental difference between an FPGA and a CPLD refers to the storage of the interconnects. While CPLDs are non-volatile (that is, they make use of anti-fuse, EEPROM, Flash, etc.), most FPGAs use SRAM, and are therefore volatile. This approach saves space and lowers the cost of the chip because FPGAs present a very large number of programmable interconnections, but requires an external ROM. There are, however, non-volatile FPGAs (with antifuse), which might be advantageous when reprogramming is not necessary.



Fig.1.3: Examples of FPGA Packages

FPGAs can be extremely refined. Chips produced with condition of-the-art 0.09mm CMOS innovation, with nine copper layers and more than 1,000 I/O pins, are at present accessible. A couple of instances of FPGA bundles are shown in figure A6, which shows one of the littlest FPGA bundles on the left (64 pins), a medium-size bundle in the center (324 pins), and an enormous bundle (1,152 pins) on the right. FPGAs are made by a number of firms, including Xilinx, Actel, Altera, and Fast Logic.

Since all Xilinx FPGAs store the interconnects in SRAM, they are reprogrammable and unpredictable (in this manner requiring outer ROM). Then again, Actel FPGAs are non-unstable (they utilize against meld), however are non-reprogrammable (aside from one family, which uses Flash memory). Since each approach has its own benefits and inconveniences, the genuine application will direct which chip design is generally proper.

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

- **"Partial Order Theory for Fast TCAM Updates"**, Peng He , Wenyuan Zhang, Hongtao Guan, Kavé Salamatian, and Gaogang Xie, **IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 26, NO. 1, FEBRUARY 2018**, Ternary content addressable memories (TCAMs) are as often as possible utilized for quick coordinating of packets against a given ruleset. While TCAMs can accomplish quick coordinating, they are tormented by high update costs that can make them unusable in a high beat rate climate. We present, in this paper, a methodical and top to bottom examination of the TCAM update issue. We apply fractional request hypothesis to determine major imperatives on any standard requesting on TCAMs, which guarantees right checking against a given ruleset. This hypothetical understanding empowers us to completely investigate the TCAM update calculations configuration space, to determine the ideal TCAM update calculation (however it probably won't be reasonable to be utilized by and by), and to acquire upper and lower limits on the presentation of functional update calculations. Having lower limits, we checked if the littlest update costs are viable with the stir rate saw by and by, and we saw that this isn't generally the situation. We hence built up a heuristic dependent on ruleset parting, with in excess of a solitary TCAM chip, that accomplishes huge update cost decreases (1.05~11.3×) contrasted and cutting-edge procedures.
- **"FAS: Using FPGA to Accelerate and Secure SDN Software Switches"**, Wenwen Fu, Tao Li, and Zhigang Sun, **Hindawi Security and Communication Networks Volume 2018**, Software Defined Networking (SDN) guarantees the vision of more adaptable and sensible organizations yet requires certain degree of programmability in the information plane to oblige distinctive sending deliberations. SDN programming turns running on product multicourse stages are programmable and are with low arrangement cost. Nonetheless, the presentation of SDN programming switches isn't good

because of the unpredictable sending procedure on bundles. Also, this may upset the presentation of ongoing security on programming switch. In this paper, we examine the sending strategy and distinguish the exhibition bottleneck of SDN programming switches. A FPGA-based system for speeding up and getting SDN switches, named FAS (FPGA-Accelerated SDN programming switch), is proposed to exploit the reconfigurability and superior benefits of FPGA. FAS improves the exhibition just as the limit against pernicious traffic assaults of SDN programming switches by offloading some practical modules. We approve FAS on a FPGA-based organization preparing stage. Test results exhibit that the sending pace of FAS can be 44% higher than the first SDN programming switch. Furthermore, FAS gives new freedom to upgrade the security of SDN programming switches by permitting the arrangement of knock in-the-wire security modules (like bundle locators and channels) in FPGA.

- **"Design and Characterization of SEU Hardened Circuits for SRAM-Based FPGA", Tianwen Li, Hongjin Liu, and Haigang Yang, IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION (VLSI) SYSTEMS, 2019 IEEE**, the mitigation of single-event upset (SEU) in SRAM-based field-programmable gate array (FPGA) is progressively significant as usage and interest for SRAM-based FPGA drastically expanded in radiation conditions like space. As D flip-flop (DFF) and memory [including block random access memory (BRAM) and configuration random access memory (CRAM)] are comprised as the critical components in a FPGA, it is in a general sense important to create radiation solidifying procedures focused for upgraded unwavering quality of DFF and memory. A tale SEU solidified memory plan for FPGA is proposed with capacities of multi cycle upset security. We further created two model FPGA chips, one with SEU and the other without SEU solidifying for correlation. The FPGA chips are manufactured in a standard 0.13- μm CMOS measure and have a volume of 3,000,000 comparable rationale doors. As far as SEU cross area, CRAM in the solidified FPGA configuration is around four significant degrees lower than in the unhardened FPGA plan, while BRAM exhibits a decrease by three significant degrees. On

the states of straight energy move being up to 14 MeV·cm²/mg, no SEU mistakes were seen from DFF in the solidified FPGA plan.

- **"Investigation into SEU Effects and Hardening Strategies in SRAM Based FPGA", Tianwen Li, Haigang Yang, He Zhao, Nan Wang, Yuanfeng Wei and Yiping Jia, 2017 IEEE**, the relief of single event upset (SEU) in SRAM based Field Programmable Gate Array (FPGA) is progressively significant as it is generally utilized in radiation conditions like space. As D flip-flop (DFF) and memory (counting Block RAM and Configuration RAM) are the vital components in FPGAs, it is pivotal to create radiation solidifying procedures for improved dependability of the DFF and memory. A tale solidified memory plan for FPGA is proposed with multi-cycle upset (MBU) security. We further created two model FPGA chips, one with and the other without SEU solidifying for examination. The FPGA chips are created in a standard 0.13Pm CMOS measure and have a volume of 3 million identical rationale doors. Rather than the base FPGA, the SEU cross part of the memory in the solidified FPGA is at any rate three significant degrees lower. Likewise, no SEU disturbs are seen in the DFF of the solidified FPGA.
- **"An ALU protection methodology for soft processors on SRAM-based FPGAs", Alexis Ramos, Ricardo G. Toral, Pedro Reviriego, and Juan Antonio Maestro, IEEE TRANSACTIONS ON COMPUTERS, 2018 IEEE**, the utilization of microchips in space missions suggests that they ought to be secured against the impacts of infinite radiation. Normally this goal has been accomplished by applying secluded repetition strategies which give great outcomes as far as unwavering quality yet increment essentially the quantity of utilized assets. Thus, new insurance strategies have showed up, attempting to build up a compromise among unwavering quality and asset use. In this paper, we propose an application-based procedure, to ensure a delicate processor executed in a SRAM-based FPGA, against the impact of delicate mistakes. This is finished making a library of versatile assurance setups, in view of the profiling of the application. This equipment setup library, joined with the reconstructing abilities of the FPGA, assists with making a versatile insurance for every application. We propose two incomplete TMR arrangements for the Arithmetic

Logic Unit (ALU) to act as an illustration of this philosophy. The proposed plot has been tried in a RISC-V delicate processor. An issue infusion crusade has been performed to test its dependability.

2.2 LITERATURE SUMMARY

The TCAM function in most of the SRAM-based FPGA solutions is defined by the content of the configured embedded memories [i.e., block RAM (BRAM), distributed RAM (distRAM)] [7], and a transient error may lead to a false match/mismatch and returns an incorrect match address. Accordingly, in the case of a soft error, the affected word of SRAM should be overwritten to retrieve the correct matching information during lookups. Be that as it may, the security of SRAM-based TCAM arrangements is trying without trading off the basic way delay and keeping up high search execution. This short presents an ease, low-response time and simple for integration method for the insurance of SRAM-empowered TCAMs without bargaining the search execution. This brief presents a low-cost, low-response time and easy for integration technique for the protection of SRAM-enabled TCAMs without compromising the search performance. The error detection is done in a straightforward manner with single-bit parity checking and very little delay and logic overhead. To fix soft errors, the proposed error-correction technique takes advantage of the redundant binary-encoded TCAM table retained in SRAM-based TCAM solutions for upgrade purposes. While the proposed error-correction mechanism is implemented, it retains a high search performance in the context, enabling multiple search operations to be carried out at the same time. The proposed error-correction technique has a low response time, ensuring a faultless TCAM design for lookups, during the entire (almost) processing time.

2.3 PROBLEM DEFINITION

The security of SRAM-based TCAMs against soft errors is difficult to achieve without sacrificing critical path delay or search results.

2.4 OBJECTIVES

- To reduce vulnerability to soft errors with high search performance and reduced critical path delay.
- To present SRAM based TCAM using error detection with error correction with different TCAM sizes 64x40, 512x28, 512x40, 1024x40, 2048x40.

2.5 EXISTING SYSTEM

- More Critical Path Delay
- More Cost and Complexity
- Low Search Performance

2.6 PROPOSED SYSTEM

- Less Critical Path Delay
- Low Cost and Complexity
- High Search Performance

CHAPTER 3

AIM AND SCOPE OF PRESENT INVESTIGATION

3.1 AIM

Static random-access memory (SRAM)-based ternary content-addressable memory (TCAM) on field-programmable gate arrays (FPGAs) is used for packet classification in software-defined networking (SDN) and Open Flow applications. SRAMs executing TCAM substance establish the significant piece of a TCAM plan on FPGAs, which are helpless against soft errors. The assurance of SRAM-based TCAMs against delicate errors is trying immediately and keeping a high search execution.

3.2 SCOPE

In a Software defined networking (SDN) in packet classification and open flow applications which using static random access (SRAM) based ternary content addressable memory (TCAM) on field programmable gate arrays (FPGAs). SRAM based TCAMs will using major part in all FPGA based applications, it will support to reduced vulnerable to soft errors with high search performance and reduced critical path delay. In brief SRAM based TCAM will reduce the cost and reduce the response time with single bit parity fault detection, which also covered minimal critical path overhead. This technique exploits the binary encoded TCAM table-maintained SRAM based TCAM for updated purpose to implement a low response time error correction. The proposed work of this paper will present SRAM based TCAM using error detection with error correction with different TCAM sizes 64x40, 512x40, 1024x40, 2048x40. Finally, this work was designed in Verilog HDL, Synthesized in Xilinx Vertex 5 FPGA and proved the performance of area, delay and power.

CHAPTER 4

METHODS AND ALGORITHMS USED

4.1 INTRODUCTION

Content-addressable memory (CAM) allows the stored content to be searched in parallel in a single cycle, achieving a high search performance. A binary CAM stores and searches data in only two states: “0” and “1,” whereas a ternary CAM (TCAM) represents data in three different states: “0,” “1,” and do not care state “x.” TCAMs are extensively used in network systems for packet classification and filtering. Modern static random-access memory (SRAM)-based field programmable gate array (FPGA) technology offers the flexibility and reconfigurability with high performance required in software defined networking (SDN) and Open Flow network accelerators for big data. Owing to the disturbances from high-energy neutron particles, circuits on SRAM-based FPGAs are susceptible to single-event upsets (SEUs). The on-chip embedded memory has been known as the most vulnerable to SEUs in advanced process technologies because of their increasingly small size and highly compact memory cells.

An SEU in embedded memory generates a transient error until the corrupted data is overwritten. SEUs can result in either single-bit upsets (SBUs) or multiple-bit upsets (MBUs). Cuckoo hashing offers a low-cost solution for implementing efficient binary CAMs on FPGAs. The TCAM function in most of the SRAM-based FPGA solutions is defined by the content of the configured embedded memories [i.e., block RAM (BRAM), distributed RAM (distRAM)], and a transient error may lead to a false match/mismatch and returns an incorrect match address. Accordingly, in the case of a soft error, the affected word of SRAM should be overwritten to retrieve the correct matching information during lookups. However, the protection of SRAM-based TCAM solutions is challenging without compromising the critical path delay and maintaining high search performance.

This brief presents a low-cost, low-response time and easy for integration technique for the protection of SRAM-enabled TCAMs without compromising the search

performance. The error detection is carried out in a simple way using single-bit parity checking at a minimal delay and logic overhead. The proposed error-correction technique exploits the redundant binary-encoded TCAM table maintained in SRAM-based TCAM solutions for update purposes to correct soft errors. It maintains a high search performance while the proposed error-correction mechanism is carried out in the background, allowing search operations to be performed simultaneously. The proposed error-correction technique has a low response time, ensuring a faultless TCAM design for lookups, during the entire (almost) processing time.

SRAM-BASED TCAM ON FPGAS:

On-chip SRAM memories in modern FPGAs are used for implementing TCAM solutions. For example, a 1×1 TCAM can be implemented using a 2×1 RAM such that the match knowledge for the presence of a “0” TCAM state is represented by storing a “1” at RAM[0], a value of “1” by storing a “1” at SRAM[1], and “x” state by storing a “1” at both the SRAM[0] and SRAM[1] locations. A C-bit TCAM pattern can be implemented using a 1-bit SRAM of 2^C positions. The address of an SRAM represents the C-bit TCAM pattern, whereas the words of the SRAM stores match/mismatch information for each word of the TCAM table against all the possible C-bit patterns. In this manner, a C-bit wide TCAM table of B words can be implemented using a B bits wide SRAM of 2^C positions.

Researchers divide the wide TCAM bit patterns into smaller chunks as they do not scale well in terms of required memory in SRAM-based TCAMs. The W-bit wide bit patterns of TCAM with depth D are divided into smaller chunks of C bits and then implemented using AND-cascaded $2^C \times D$ size SRAMs [8], [9]. This is explained using a simplified implementation of an SRAM-based TCAM shown in Fig. 1. The 4-bit patterns of a 4-word deep TCAM are divided into two partitions of 4×2 , which are then implemented using the two 4×4 SRAMs shown in Fig. 1(b). Let us consider a search key (1001) is applied for the search operation, the first two bits (10) would access the third word of the first SRAM (1100) and the last two bits (01) would access the second word of the second SRAM (1001). The SRAM words read are ANDed to get the final match result (1000) which represents a match for rule R0.

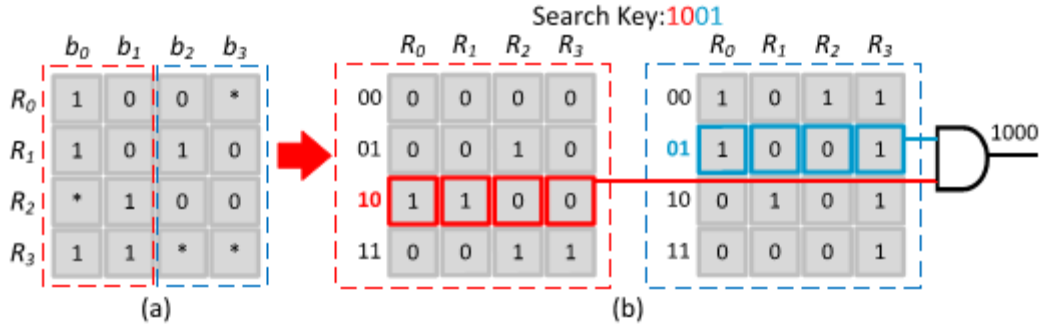


Fig:4.1: Simplified implementation of TCAM using SRAM. (a) 4x4 TCAM table.

(b) Implementing a 4×4 TCAM using two 4×4 SRAMs

There are different solutions for implementing TCAM on FPGAs. The flip flops (FFs) available in the logic resources of the FPGA are used for implementing small-size TCAMs. As mentioned before, existing FPGA-based TCAMs mostly employ BRAM or distRAM. Higher memory efficiency is achieved when shallow SRAMs are used to design TCAMs. The minimum depth limitation on the configuration of Xilinx BRAM is 512, which implies that 9 bits of TCAM are implemented using 29 BRAM bits. Hence, the optimal configuration size of 18- and 36-kb size BRAMs is 512×36 and 512×72 for which the width of the TCAM chunk is 9 bits. The distRAM-based TCAMs configure the four 64-bit LUTRAMs available in SLICEMs as a 32×6 simple dual-port RAM thereby implementing 30 TCAM bits per SLICEM. This implies that 256 distRAM bits are used to implement 30 bits of TCAM, and 8.53 distRAM bits are utilized for a single-bit of TCAM. Hence, the optimal depth of distRAM-based SRAMs is 32, for which the width of the TCAM chunk is 5 bits. In practice, it is recommended to realize TCAM using BRAMs, and utilizing LUTRAMs for implementing other parts of the system.

Generally, the TCAM designs that use more resources tend to be more sensitive to soft errors. The breakdown of BRAMs, LUTRAMs, and slice registers on a 28-nm FPGA device is 34%, 5%, and 1%, respectively, making SEUs in the smaller population much less likely [16]. Thus, the TCAM architectures implemented in BRAMs are more vulnerable to soft errors than others (distRAM-based and SR-based). The rate at which soft errors occur is often measured in terms of failures in time (FIT) and the real-time soft error rate per event in the BRAMs of the FPGA device used in the experiments of the proposed work Artix-7 is $73\% \pm 9\%$.

The wild-card state “x” enables the TCAM words to represent the ranges. An input word might match several words when the ranges of two or more words overlap. In the case of a match with multiple words, the word with the lowest order is forwarded. It is mandatory to arrange the TCAM words in an order list such that higher priority words are stored in lower memory addresses. An update of a TCAM word might change the priorities and entails a rearrangement of the words in the TCAM table. During update operations, incoming words for the search operations remain suspended as cannot be matched correctly. Future networking systems like the SDN and Open Flow require frequent updates in the network switches. State-of-the-art SRAM-based TCAM solutions on FPGAs update a TCAM entry in the binary-encoded TCAM table which involves multiple TCAM entry movements to maintain the order constraint, further mapped onto SRAMs emulating the TCAM function. The proposed ER-TCAM exploits this system-level data redundancy offered by the stored binary-encoded TCAM content to correct soft errors in SRAMs emulating the TCAM function.

4.2 PROPOSED SYSTEM

In a Software defined networking (SDN) in packet classification and open flow applications which using static random access (SRAM) based ternary content addressable memory (TCAM) on field programmable gate arrays (FPGAs). Thus, SRAM based TCAMs will using major part in all FPGA based applications, it will support to reduced vulnerable to soft errors with high search performance and reduced critical path delay. In brief SRAM based TCAM will reduce the cost and reduce the response time with single bit parity fault detection, which also covered minimal critical path overhead. This technique exploits the binary encoded TCAM table-maintained SRAM based TCAM for updated purpose to implement a low response time error correction. The proposed work of this paper will present SRAM based TCAM using error detection with error correction with different TCAM sizes 64x40, 512x40, 1024x40, 2048x40. Finally, this work was designed in Verilog HDL, /synthesized in Xilinx Vertex 5 FPGA and proved the performance of area, delay and power.

Informational redundancy is essential to realize fault tolerance in SRAM-based TCAMs. In SRAM-based TCAMs, the $2C \times D$ bit data stored in SRAMs realizing TCAM functionality is generated from a $D \times C$ TCAM content, which is at most $2D \times C$ bit

information because of the three states of a TCAM cell can be represented using two SRAM bits (“0”→“00,” “1”→“01,” and “x”→“10”). As discussed before, in modern SRAM-based TCAMs, the original TCAM content is also maintained on-chip for update purposes, so the $2C \times D$ bit data of the design-configured SRAMs is completely redundant. ER-TCAM exploits this system-level data redundancy to correct soft errors in SRAMs. The basic idea of the ER-TCAM technique is illustrated in Fig. 4.1. The two SRAMs, shown in Fig. 4.2, implements the TCAM table partitions presented in Fig. 4.1. Fig.4.2 shows an 8×4 size SRAM storing binary-encoded contents of the TCAM table, providing redundancy of the matching information in SRAMs realizing TCAM.

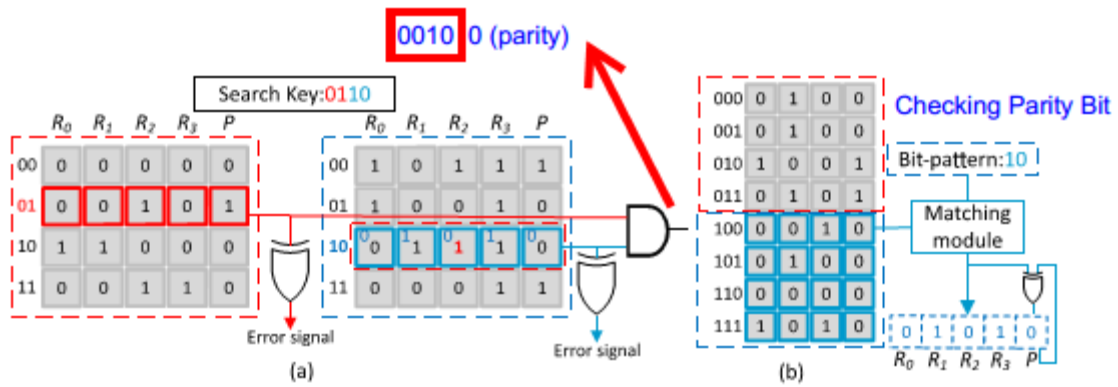


Fig.4.2: Simplified example of the proposed ER-TCAM. (a) Error detection in parity protected SRAMs implementing TCAM (b) Error correction vector generation using the binary encoded TCAM contents.

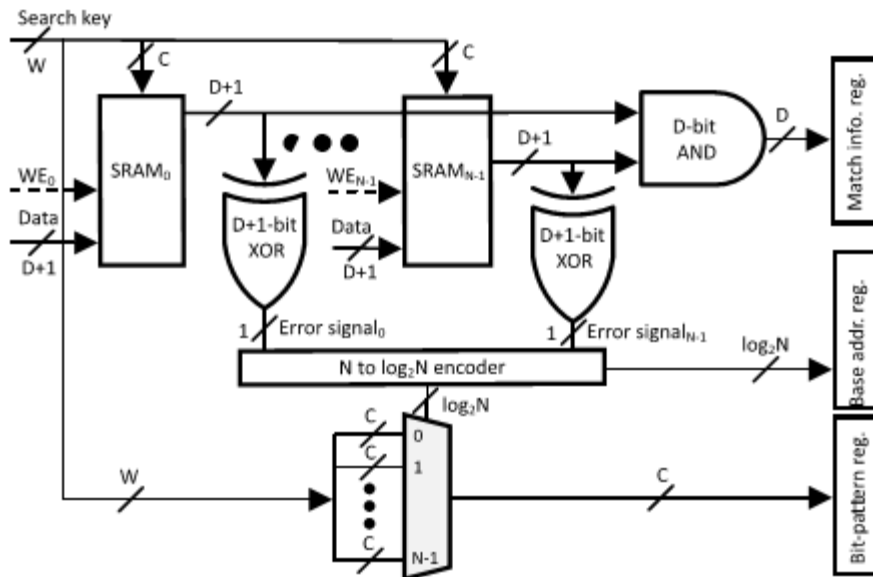


Fig.4.3: Proposed ER-TCAM architecture for error detection

To detect an SBU in SRAMs, the ER-TCAM adds a parity bit for each SRAM word as

shown in Fig. 4.2. Error detection is performed on the words of SRAMs accessed during lookups. Once an error is detected in a word, the ER-TCAM uses the redundant information stored in the binary-coded TCAM table for correction.

For example, when a search key (0110) is applied to SRAMs shown in Fig. 4.1, accesses the second (00101) and third words (01110) of the first and second SRAMs, respectively. The calculated parity for the match bits of the second SRAM word (01110) does not match the parity stored, indicating the occurrence of an SBU in SRAM as shown in Fig. 1.1. The ER-TCAM accesses the words of SRAM storing the binary-encoded contents of the related TCAM partition one by one. The SRAM words read are matched with the corresponding bit pattern (10) to compute match bits and associated parity (01010) collectively called error-correction vector (ECV) as shown in Fig. 4.2. The computed ECV is further written over the corrupted SRAM word. In this way, the ER-TCAM ensures 100% coverage against the SBUs. Moreover, the ER-TCAM is able to protect against a special case of MBUs when an odd number of bits flip in an SRAM word detected and corrected using the same aforementioned processes.

Fig. 4.3 shows the proposed ER-TCAM error-detection architecture. When an input search key is applied for lookup, the bits of the SRAM words read are EX-ORed to get an error signal. The error signals from the N SRAMs of the TCAM design are encoded to get a $\log_2 N$ -bit error code that uniquely identifies respective corrupted SRAM. The error code and related search-key bit patterns are forwarded to the error-correction module.

ARCHITECTURE OF PROPOSED ER-TCAM

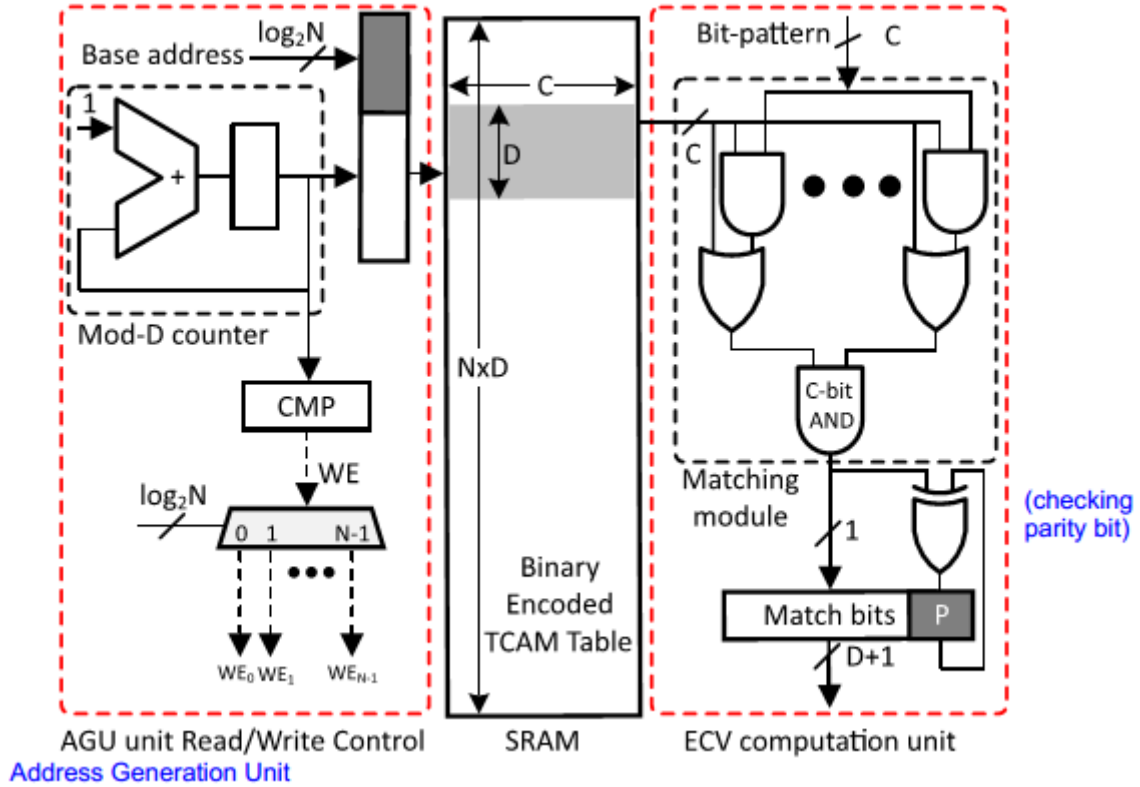


Fig:4.4: Proposed ER-TCAM error-correction module

Fig. 4.4 shows the ER-TCAM architecture for error correction which mainly comprises an SRAM storing binary-encoded contents of the TCAM table, an ECV computation unit, an address generation unit (AGU), and a read/write controller. The MOD-D counter generates a new sequence of $\log_2 D$ bits for every cycle. The SRAM address is executed such that $\log_2 N$ -bits of the SRAM ID constitute its most significant bits and points to the start of the corresponding sub-block in SRAM, and the lower $\log_2 D$ bits from the counter select SRAM words in the sub-block. In this way, the AGU accesses all the binary encoded words of the corresponding partition of the TCAM table. The TCAM words read are matched with the C-bit pattern to get a match bit each cycle, thus requiring D clock cycles to compute the match bits and the associated parity bit, composing the ECV. The read/write controller generates write enable high signal for the corresponding SRAM to write the computed ECV over the corrupted SRAM word. During the error-correction process, the ER-TCAM allows search operations as SRAMs realizing the TCAM function are available for lookup operations. The ER-TCAM configures these SRAMs as simple dual-port RAM that performs the read and

write in parallel at the same clock cycle. Once the ECV is computed, it is written using the write port of SRAM, thus, the error correction process completely overlaps the search operations in the ER-TCAM. Although a soft error can occur in the SRAM storing binary-encoded TCAM table; however, its error occurrence probability compared with that of SRAMs realizing TCAM is very low, owing to its relatively small size. Still, the ER-TCAM is able to protect SRAM storing the binary encoded TCAM table using ECCs at a very little cost of memory and error-correction latency overhead.

4.3 SOFTWARE REQUIREMENT

1.1.5 4.3.1 ModelSim – Altra

1.1.6 4.3.1.1 Assumptions

I expect that you know about the utilization of your working framework. You ought to likewise be comfortable with the window the board elements of your realistic interface: Open Windows, OSF/Motif, CDE, KDE, GNOME, or Microsoft Windows 2000/XP. We additionally expect that you have a functioning information on the language wherein your plan or potentially test seat is composed (i.e., VHDL, Verilog, and so on) In spite of the fact that ModelSim™ is a brilliant apparatus to utilize while learning HDL ideas and practices, this record isn't composed to help that objective.

1.1.7 4.3.1.2 ModelSim Introduction

ModelSim is a verification and simulation tool for VHDL, Verilog, System Verilog, and mixed language designs. This lesson provides a brief conceptual overview of the ModelSim simulation environment. It is divided into four topics, which you will learn more about in subsequent lessons.

1.1.8 4.3.1.3 Basic Simulation Flow

The following diagram shows the basic steps for simulating a design in ModelSim.

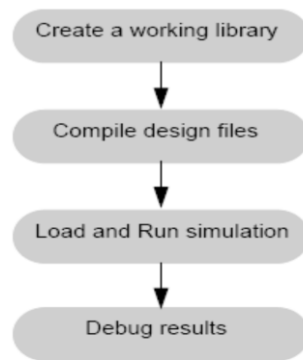


Fig4.5: Basic Simulation Flow

4.3.1.4 Basic Simulation Flow - Overview Lab [Creating the Working Library]

In ModelSim, all plans are ordered into a library. You normally start another recreation in ModelSim by making a functioning library called "work," which is the default library name utilized by the compiler as the default objective for assembled plan units.

- **Compiling Your Design**

In the wake of making the functioning library and accumulate your plan units into it. The ModelSim library design is viable across totally upheld stages. Its can recreate your plan on any stage without having to recompile your plan Stacking the Simulator with Your Design and Running the Simulation with the Plan Load the test device with your plan by conjuring it on a high-level module (Verilog) or a configuration or substance/engineering pair (VHDL).

Expecting the plan stacks effectively, the recreation time is set to nothing, and you enter a run order to start reproduction.

- **Debugging Your Results**

In the event that you don't get the outcomes you expect, you can utilize ModelSim's hearty investigating climate to find the reason for the issue.

1.1.9 4.3.2 Project Flow

A venture is an assortment component for a HDL plan under detail or test. Indeed, even thoughyou don't need to utilize projects in ModelSim, they may ease communication with the apparatus and are valuable for getting sorted out documents and indicating reenactment settings. The accompanying outline shows the

fundamental strides for reenacting a plan inside a ModelSim project.

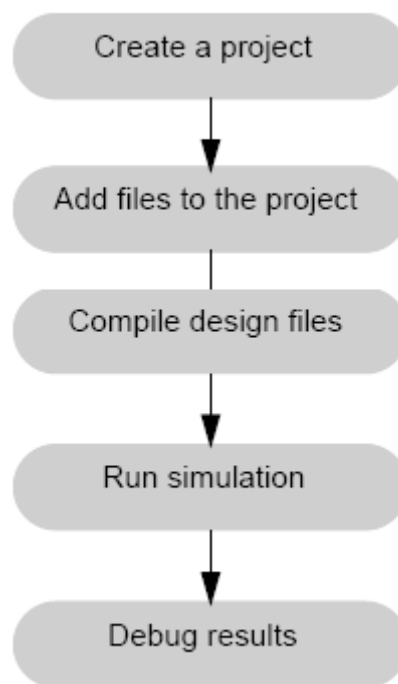


Fig:4.6: Project flow

As you can see, the flow is similar to the basic simulation flow. However, there are two important differences:

- Do not have to create a working library in the project flow; it is done for you automatically.
- Projects are persistent. In other words, they will open every time you invoke ModelSim unless you specifically close them.

1.1.10 4.3.3 Multiple Library Flow

ModelSim makes use of libraries in two ways: as a local working library containing the compiled version of the template and as a reference library. When you upgrade and recompile your template, the contents of your working library will change. A resource library is commonly static and fills in as a sections hotspot for your plan. It can make your own asset libraries, or they might be provided by another plan group or an outsider (e.g., a silicon merchant). It indicates which asset libraries will be utilized when the plan is assembled, and there are rules to determine in which request they are looked. A typical illustration of utilizing both a functioning library and an asset

library is one where your door level plan and test seat are incorporated into the functioning library, and the plan references entryway level models in a different asset library. The outline beneath shows the essential strides for recreating with various libraries.

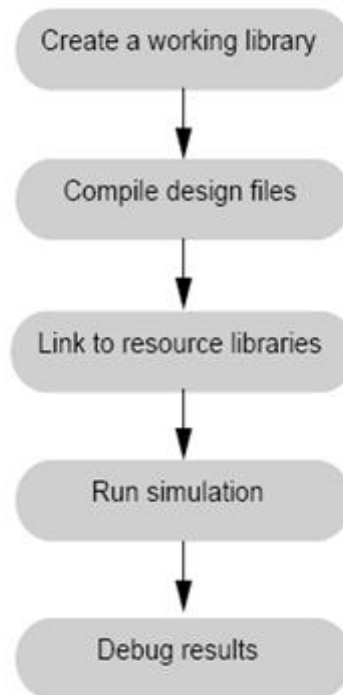


Fig:4.7: Multiple Library flow

1.1.11 4.3.4 Debugging Tools

Model Sim offers numerous tools for debugging and analyzing your design.

Following lessons will go into a few of these techniques, including:

- Using projects
- Using different libraries
- Using breakpoints and stepping into the source code
- Viewing waveforms and tracking time
- Viewing and initialising memories
- Using the Waveform Editor to provide stimuli
- Automating simulation

1.1.12 4.3.5 Basic Simulation

4.3.5.1 Introduction

In this lesson you will go step-by-step through the basic simulation flow:

1. Create the Working Design of the Library
2. Compile the Design Units
3. Load the Design
4. Let's run the Simulation.

4.3.5.2 Design Files for This Lesson

A basic 8-bit binary up-counter with an accompanying test bench is the example design for this tutorial. The below are the pathnames:

Verilog – `<install_dir>/examples/tutorials/verilog/basicSimulation/counter.v` and `tcounter.v`

VHDL – `<install_dir>/examples/tutorials/vhdl/basicSimulation/counter.vhd` and `tcounter.vhd`

This lesson uses the Verilog files `counter.v` and `tcounter.v`. If you have a VHDL license, use

`Counter.vhd` and `tcounter.vhd` instead. Or, if you have a mixed license, feel free to use the

Verilog test bench with the VHDL counter or vice versa.

4.3.6 Xilinx ISE

The means needed to perform conduct and post-course reproductions utilizing the Xilinx Integrated Software Environment (ISE) and Mentor Graphics ModelSim test system. Previously, Xilinx packaged the ISE with an authorized version of ModelSim, bringing about next to zero issues between the two programming projects. Notwithstanding, lately it shows up as though the two organizations are not, at this point on level standing. Therefore, Mentor Graphics doesn't give Xilinx licenses for ModelSim and Xilinx as far as it matters for its don't give pre-incorporated gadget libraries to ModelSim. This instructional exercise will tell you the best way to determine similarity issues, so you can keep on utilizing the highlights that these product's give.

Similarly, as with any product instructional exercise, there is in every case more than one method of getting things done. On the off chance that you run over better arrangements or errors or have any ideas for improving this archive kindly email the creator or contact Dr. Grantner.

As with any software tutorial, there is always more than one way of doing things. Please email the author or contact Dr. Grantner if you see better ideas, typos, or have any recommendations for improving this text.

1.1.13 4.4 PRE-REQUISITE

Before reading any further you should have the following software's installed on your computer.

1.1.14 4.4.1 Simulation Libraries

- Simulation Library Compilation Wizard
- Compile HDL Libraries through Project Navigator

1.1.15 4.4.2 Setting Up Environment Variables

Before we discuss the individual methods, let us first understand how ModelSim accesses these libraries. A list of the compiled libraries and their locations is stored in modelsim.ini file found in the installation directory. At the point when ModelSim begins a reenactment it looks for this ini record; first, in the nearby task envelope, at that point in the establishment catalog of ModelSim. In the event that a specific library isn't discovered, the reenactment stops with a blunder. Both the techniques examined above can't get to the modelsim.ini _le straightforwardly, so it is significant that we arrangement climate factors that highlight the area of this _le. Setting up climate factors requires regulatory advantages on your PC. In the event that you don't have these advantages jump to the following segment.

1. The first step is to change the modelsim.ini file's write permissions. Navigate to the ModelSim installation folder. This is c: modeltech64 10.0c in our setup (this will be different for other versions). Uncheck Read-only in modelsim.ini by right-clicking it and selecting Properties.
2. Make a reinforcement of the modelsim.ini document for security reasons.

3. The easiest way to view or modify the environment variables is by right-clicking on My Computer→Properties; click advanced system settings→Environment Variables.
4. Under System variables, click New; enter Modelsim for the variable name. For the Variable value enter: \modeltech64 10.0cnmodelsim.ini.
5. Also, add another variable named Path with the worth c:\modeltech64 10.0cnwin64. For our arrangement; win64 is the place where the executable (modelsim.exe) is found. On the off chance that a variable called way as of now exists, ensure you attach to the current qualities. Note that different qualities can be isolated by semi-colon.
6. Snap OK to save these settings.

1.1.16 4.4.3 Simulation Library Compilation Wizard:

This approach is highly recommended because it allows you to compile libraries for multiple computer families and/or languages. Open the Compilation Wizard for the Simulation Library.

Open the Simulation Library Compilation Wizard. This can be accessed from Start Menu→Programs→Xilinx ISE Design Suite 13.x→ISE Design Tools→32-bit Tools.

Note: Use the Simulation Library Compilation Wizard from 64-bit Tools if you're using the 64-bit version of ModelSim. Only the 32-bit version of ModelSim PE Student Version 10.x is available.

1. The window Select Simulator appears. Select the required simulator (ModelSim PE for Student Edition); for the executable position, type c:\modeltech64 10.0cnwin64; for the Compplib Configuration File, type compplib.cfg; and for the Compplib Log File, type compplib.log.
2. Choose the HDL that will be used for simulation. Pick both VHDL and Verilog if you're unsure. However, this will lengthen the compilation process and necessitate more disc space.
3. Then choose all of the system families for which you'll be operating. The more computers, the longer the compilation time and the more disc space available. Remember that you can still run the compilation wizard for additional devices at a later time.

4. Select libraries for Functional and Timing Simulation in the next window. Different libraries are required for different types of simulation (behavioral, post-route, etc.). We suggest that you select All Libraries as the default option. Interested users can find more detail in Chapter 6 of the Xilinx Synthesis and Simulation Design Guide.
5. Finally, the window for the compiled libraries' output directory appears. We recommend leaving the Xilinx default values alone. Then choose Start Compilation Process.
6. Be patient, as the compilation process will take a long time depending on your choices.
7. The compilation process may have produced numerous alerts, but it should have been error-free. We haven't looked into the explanations for these alerts, but they don't seem to have an effect on the simulation of any of our projects.
8. After the process is finished, open `c:\modeltech64\10.0\cnmodelsim.ini` and check for libraries pointing to the output directory specified in phase 6. This can only happen if the environment variables have been set. The library has now been completed. If the environment variables aren't set, the wizard generates a `modelsim.ini` file in the output directory you specified in step 6. This position is set to `c:` by default. Double-check this file and double-check that it contains the location of the recently compiled libraries. This file should be copied into every project you're working on create.

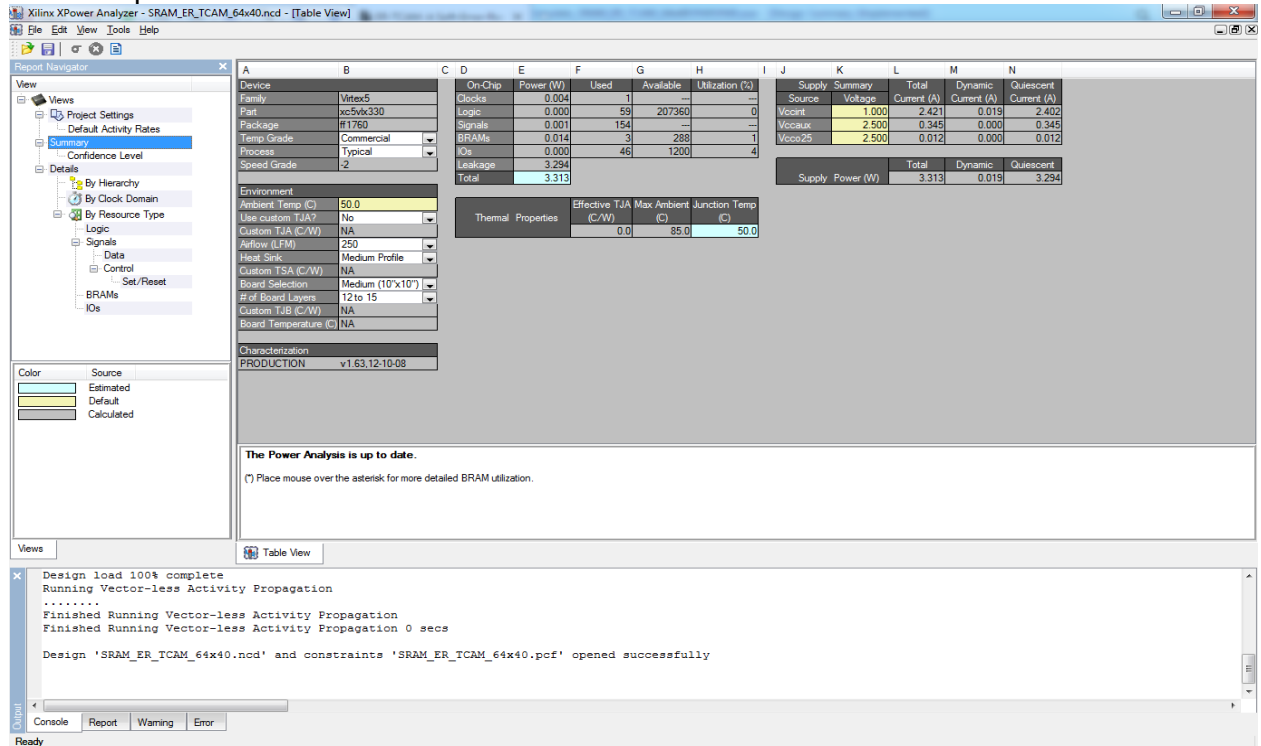
CHAPTER 5

RESULTS AND DISCUSSION

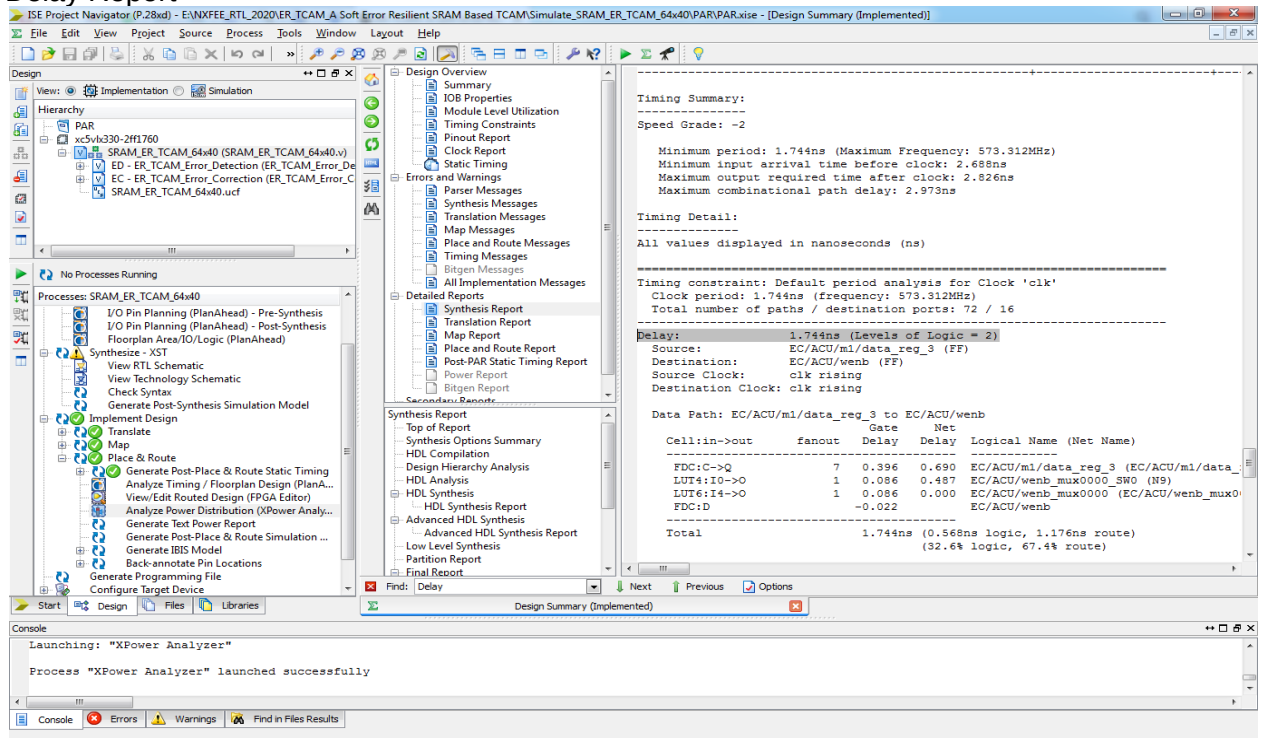
5.1 RESULT

5.1.1 TCAM 64x40

Power Report



Delay Report

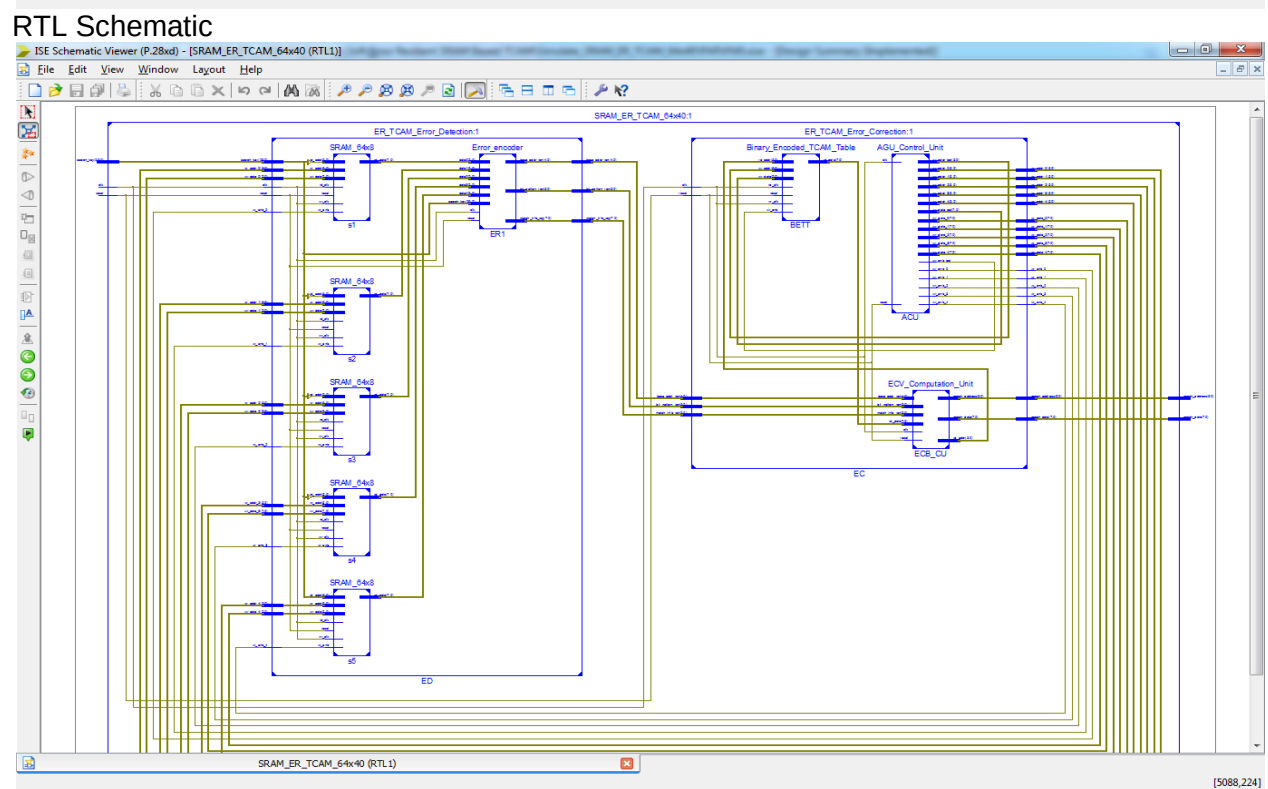


The screenshot shows the Xilinx Vivado IDE interface. The main window displays the "Design Summary (Implemented)" report for the project "SRAM_ER_TCAM_64x40".

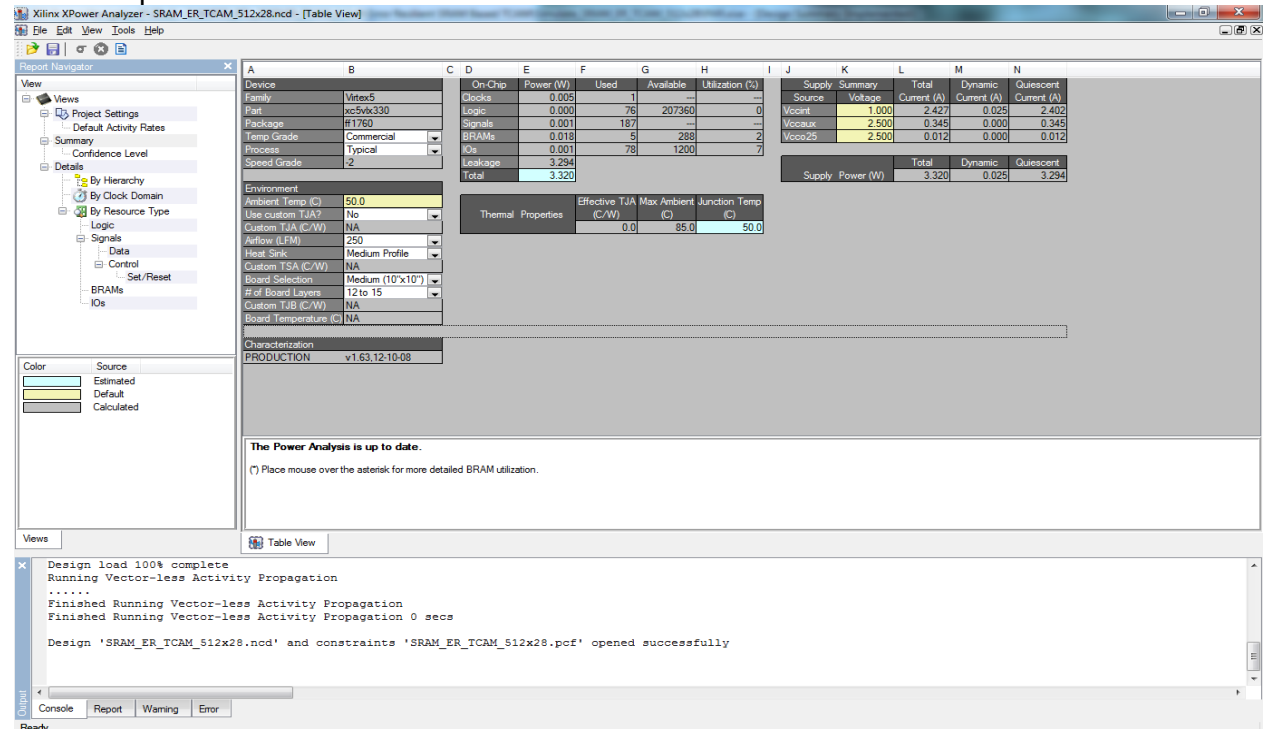
Project File: PAR.xise
Parser Errors: No Errors
Module Name: SRAM_ER_TCAM_64x40
Implementation State: Placed and Routed
Target Device: xc5vfx330-2ffl760
Errors: No Errors
Product Version: ISE 14.2
Warnings: 5 Warnings (0 new)
Design Goal: Balanced
Routing Results: All Signals Completely Routed
Design Strategy: Xilinx Default (unlocked)
Timing Constraints: All Constraints Met
Environment: System Settings
Final Timing Score: 0 (Timing Report)

Device Utilization Summary

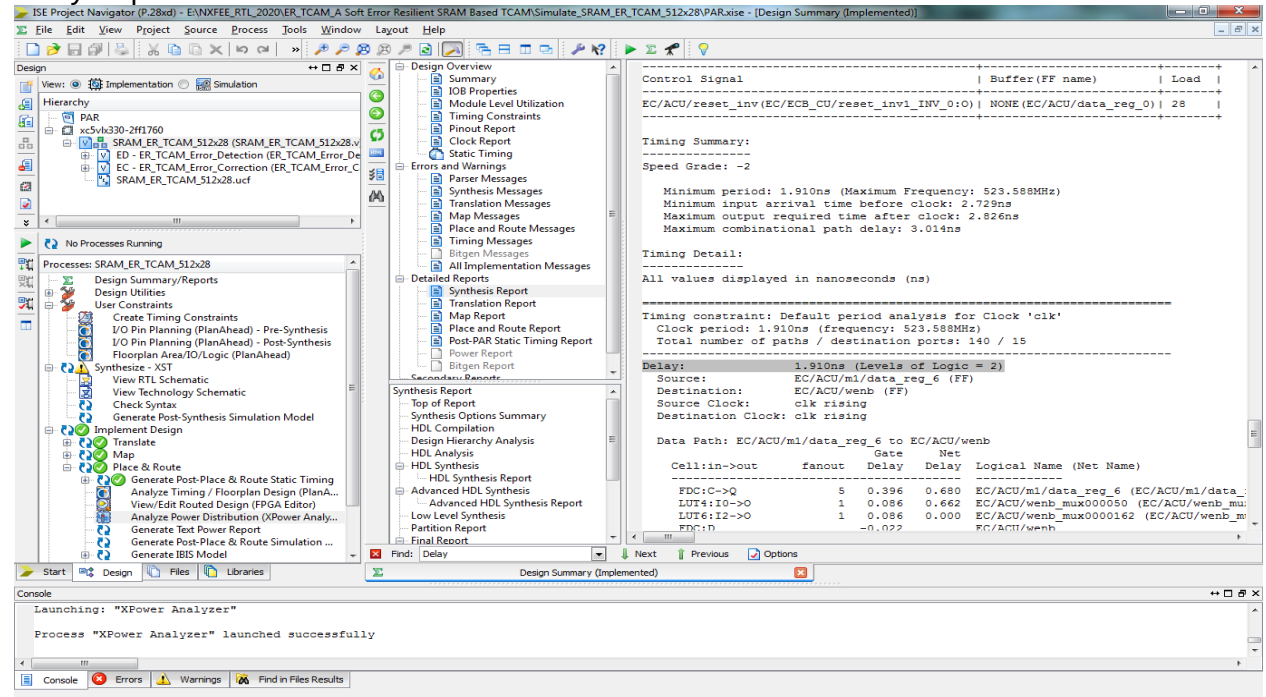
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	16	207,360	1%	
Number used as Flip Flops	16			
Number of Slice LUTs	59	207,360	1%	
Number used as logic	59	207,360	1%	
Number using O6 output only	59			
Number of occupied Slices	25	51,840	1%	
Number of LUT Flip Flop pairs used	59			
Number with an unused Flip Flop	43	59	72%	
Number with an unused LUT	0	59	0%	
Number of fully used LUT-FF pairs	16	59	27%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	0	207,360	0%	
Number of bonded IOBs	46	1,200	3%	
IOB Flip Flops	14			
Number of BlockRAM/FIFO	3	288	1%	
Number using BlockRAM only	3			



Power Report



Delay Report



Synthesized Report

ISE Project Navigator (P.28xd) - E:\NXP\FEE_RTL_2020\ER_TCAM_A Soft Error Resilient SRAM Based TCAM\Simulate_SRAM_ER_TCAM_512x28\PAR.xise - [Design Summary (Implemented)]

File Edit View Project Source Process Tools Window Layout Help

Design Overview

View: Implementation Simulation

Hierarchy

- PAR
 - xc5v330-2H1760
 - SRAM_ER_TCAM_512x28 (SRAM_ER_TCAM_512x28.v)
 - ED - ER_TCAM_Error_Detection (ER_TCAM_Error_Detection.v)
 - EC - ER_TCAM_Error_Correction (ER_TCAM_Error_Correction.v)
 - SRAM_ER_TCAM_512x28.ucf

No Processes Running

Processes: SRAM_ER_TCAM_512x28

- Design Summary/Reports
 - Design Utilities
 - User Constraints
 - Create Timing Constraints
 - I/O Pin Planning (PlanAhead) - Pre-Synthesis
 - I/O Pin Planning (PlanAhead) - Post-Synthesis
 - Floorplan Area/IO/Logic (PlanAhead)
 - Synthesize - XST
 - View RTL Schematic
 - View Technology Schematic
 - Check Syntax
 - Generate Post-Synthesis Simulation Model
 - Implement Design
 - Translate
 - Map
 - Place & Route
 - Generate Post-Place & Route Static Timing
 - Analyze Timing / Floorplan Design (PlanAhead)
 - View/Edit Routed Design (FPGA Editor)
 - Analyze Power Distribution (XPower Analyzer)
 - Generate Text Power Report
 - Generate Post-Place & Route Simulation ...
 - Generate IBIS Model

Design Overview

- Summary
 - IOB Properties
 - Module Level Utilization
 - Timing Constraints
 - Pinout Report
 - Clock Report
 - Static Timing
- Errors and Warnings
 - Parser Messages
 - Synthesis Messages
 - Translation Messages
 - Map Messages
 - Place and Route Messages
 - Timing Messages
 - Bitgen Messages
 - All Implementation Messages
- Detailed Reports
 - Synthesis Report
 - Translation Report
 - Map Report
 - Place and Route Report
 - Post-PAE Static Timing Report
 - Power Report
 - Bitgen Report
 - Secondary Reports
- Design Properties
 - Enable Message Filtering
- Optional Design Summary Contents
 - Show Clock Report
 - Show Failing Constraints
 - Show Warnings
 - Show Errors

Find: Delay

Next Previous Options

Design Summary (Implemented)

Console

Launching: "XPower Analyzer"

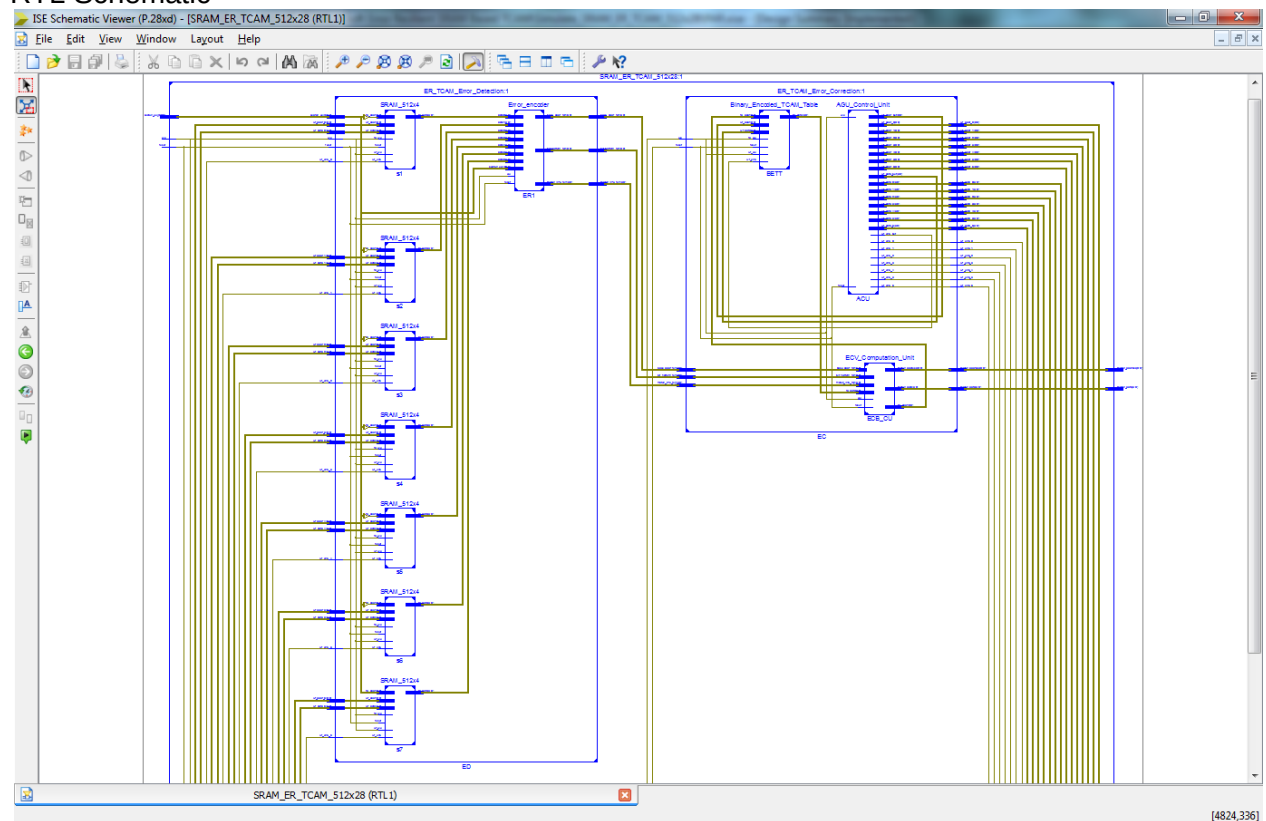
Process "XPower Analyzer" launched successfully

Console Errors Warnings Find in Files Results

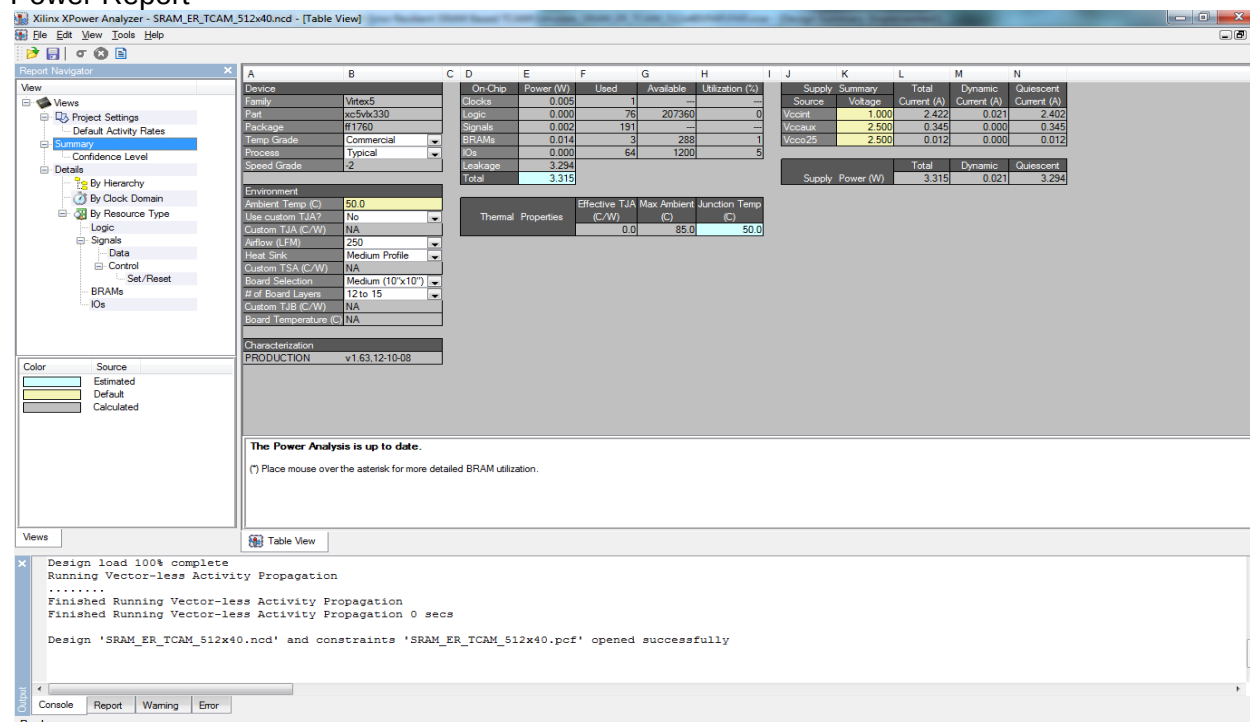
SRAM_ER_TCAM_512x28 Project Status (11/20/2020 - 12:31:21)			
Project File:	PAR.xise	Parser Errors:	No Errors
Module Name:	SRAM_ER_TCAM_512x28	Implementation State:	Placed and Routed
Target Device:	xc5v330-2H1760	Errors:	No Errors
Product Version:	ISE 14.2	Warnings:	5 Warnings (0 new)
Design Goal:	Balanced	Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	Timing Constraints:	All Constraints Met
Environment:	System Settings	Final Timing Score:	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	15	207,360	1%	
Number used as Flip Flops	15			
Number of Slice LUTs	76	207,360	1%	
Number used as logic	76			
Number using O6 output only	68			
Number using O5 and O6	8			
Number of occupied Slices	36	51,840	1%	
Number of LUT Flip Flop pairs used	76			
Number with an unused Flip Flop	61	76	80%	
Number with an unused LUT	0	76	0%	
Number of fully used LUT-FF pairs	15	76	19%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	1	207,360	1%	
Number of bonded IOBs	78	1,200	6%	
IOB Flip Flops	13			
Number of BlockRAM/FIFO	5	288	1%	

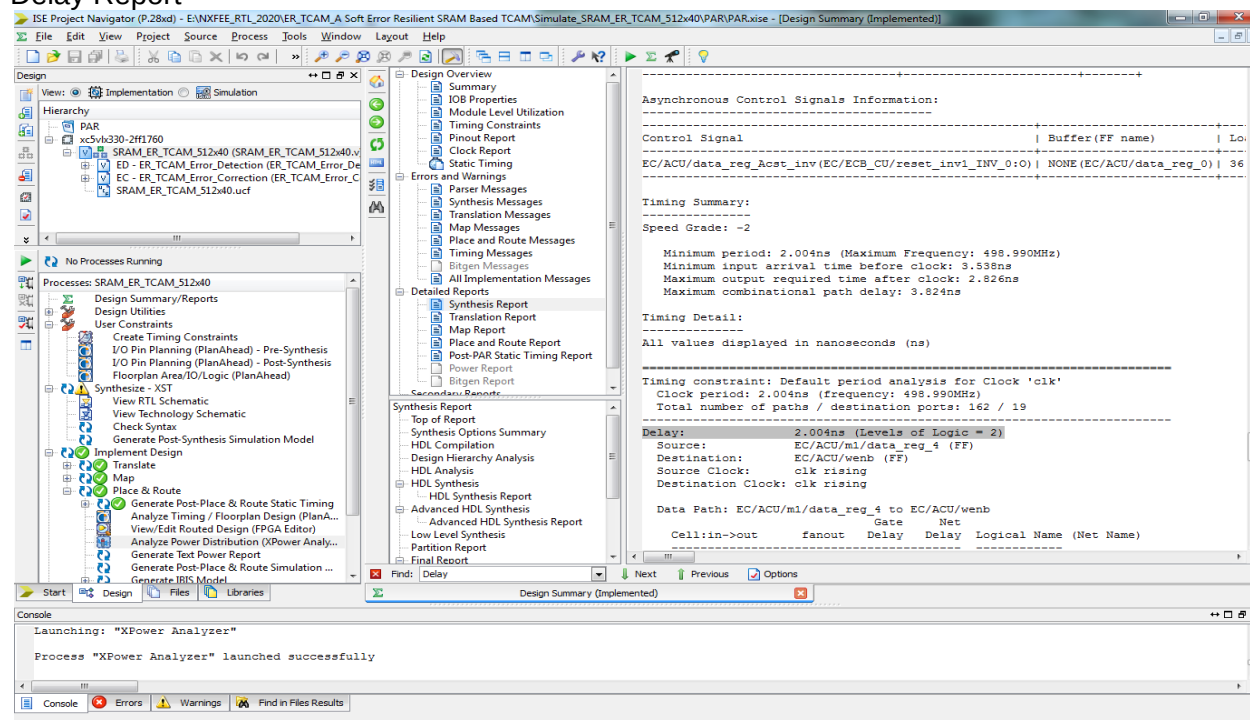
RTL Schematic

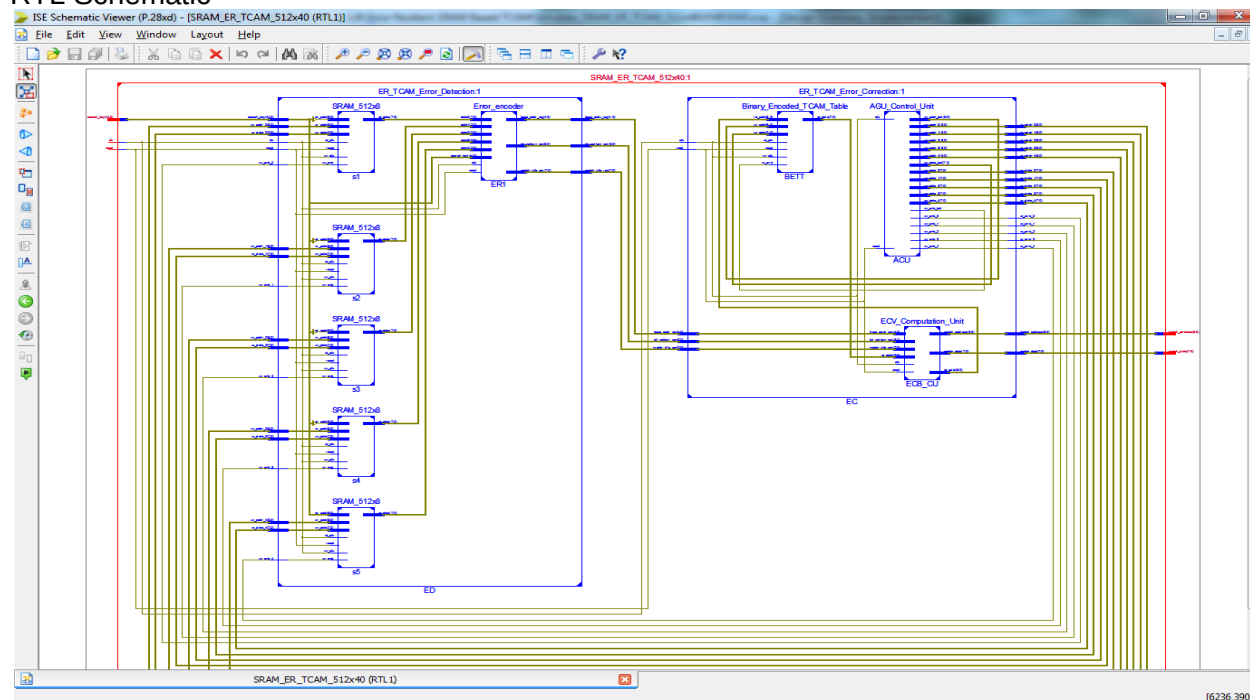
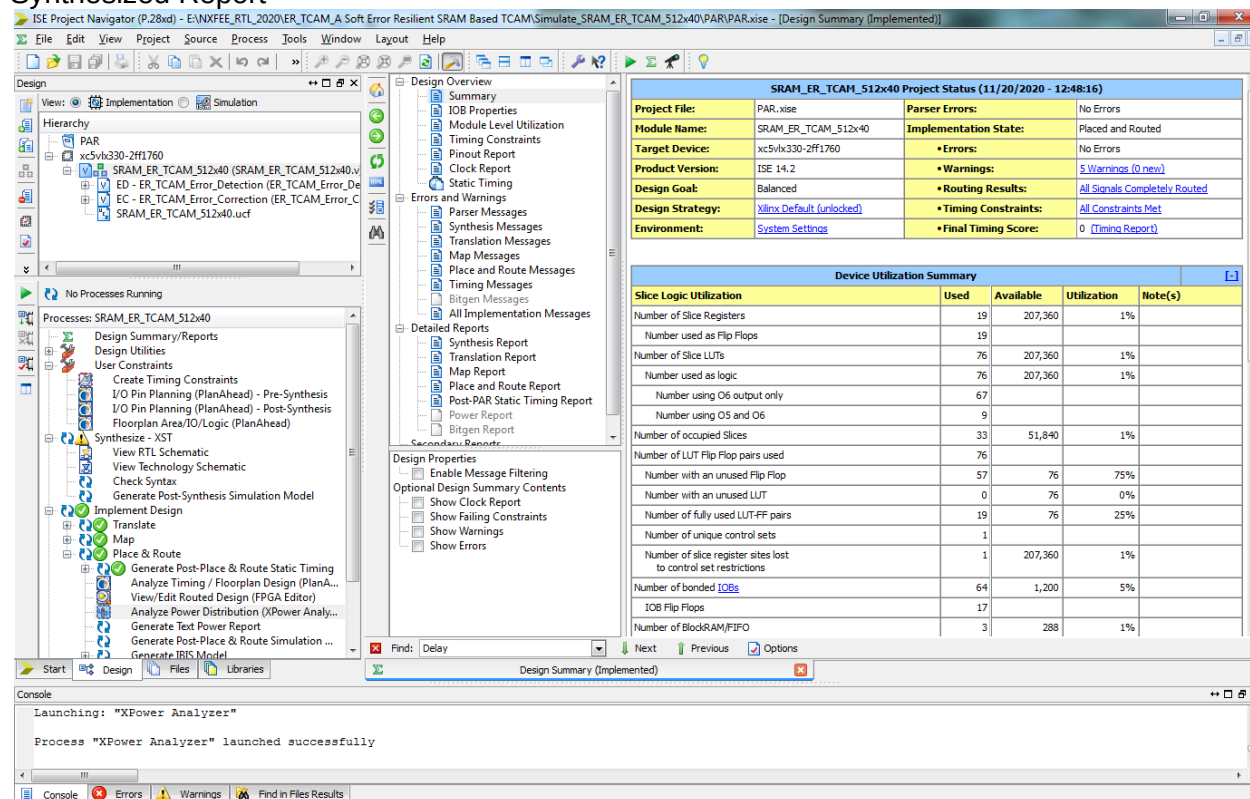


Power Report

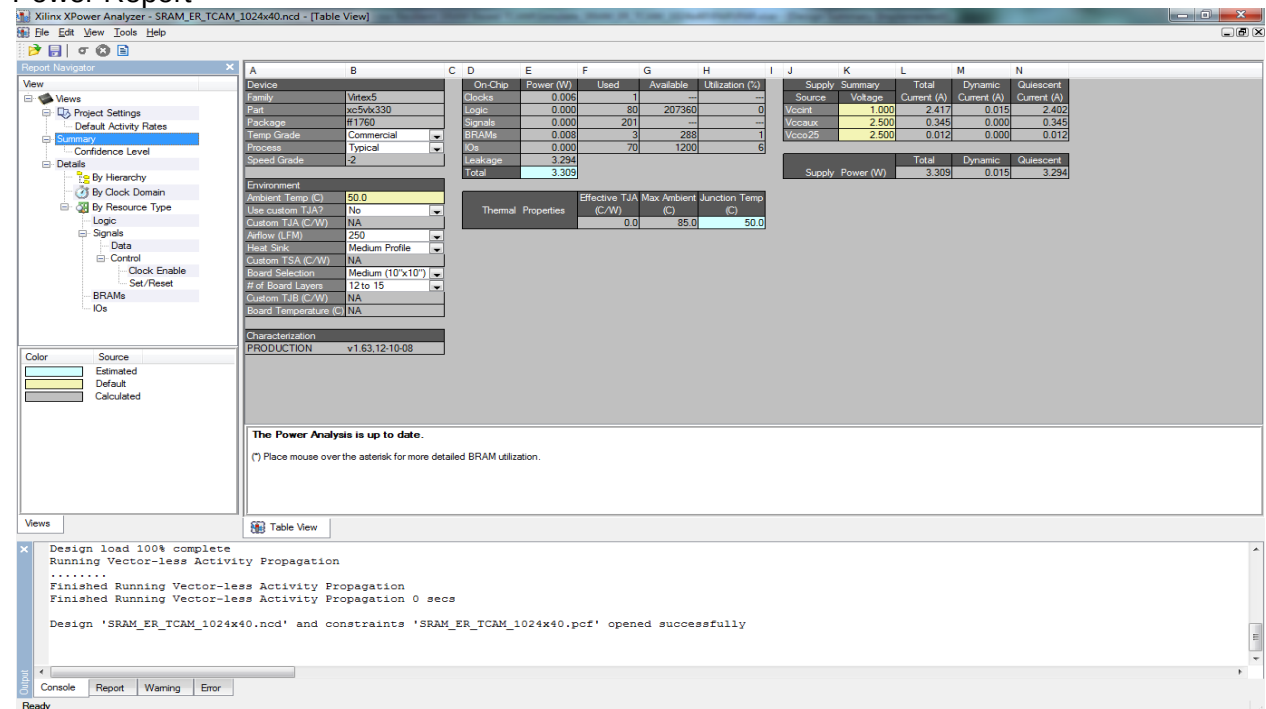


Delay Report

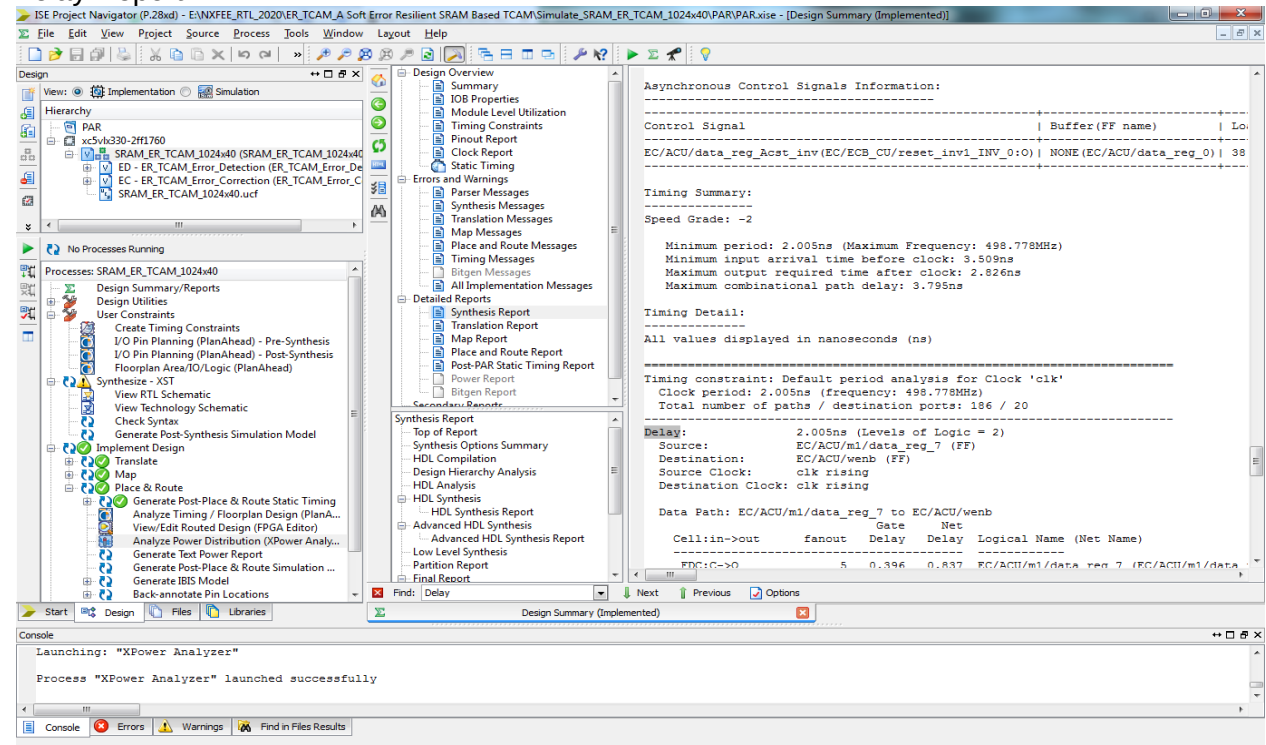




Power Report



Delay Report



Synthesized Report

ISE Project Navigator (P.28xd) - E:\NXXFEE_RTL_2020\ER_TCAM_A_Soft_Error_Resilient_SRAM_Based_TCAM\Simulate_SRAM_ER_TCAM_1024x40\PAR\PAR.xise - [Design Summary (Implemented)]

File Edit View Project Source Process Tools Window Layout Help

Design Overview

View: Implementation Simulation

Hierarchy

- PAR
 - xc5vfx330-2ff1760
 - SRAM_ER_TCAM_1024x40 (SRAM_ER_TCAM_1024x40)
 - EO - ER_TCAM_Error_Detection (ER_TCAM_Error_Detection)
 - EC - ER_TCAM_Error_Correction (ER_TCAM_Error_Correction)
 - SRAM_ER_TCAM_1024x40.ucf

Processes: SRAM_ER_TCAM_1024x40

No Processes Running

Design Summary/Reports

- Design Utilities
 - Create Timing Constraints
 - I/O Pin Planning (PlanAhead) - Pre-Synthesis
 - I/O Pin Planning (PlanAhead) - Post-Synthesis
 - Floorplan Area/IO/Logic (PlanAhead)
- Synthesize - XST
 - View RTL Schematic
 - View Technology Schematic
 - Check Syntax
 - Generate Post-Synthesis Simulation Model
- Implement Design
 - Translate
 - Map
 - Place & Route
 - Generate Post-Place & Route Static Timing
 - Analyze Timing / Floorplan Design (PlanAhead)
 - View/Edit Routed Design (FPGA Editor)
 - Analyze Power Distribution (XPower Analyzer)
 - Generate Text Power Report
 - Generate Post-Place & Route Simulation ...
 - Generate IBIS Model
 - Back-annotate Pin Locations

Design Overview

- Summary
 - JOB Properties
 - Module Level Utilization
 - Timing Constraints
 - Pinout Report
 - Clock Report
 - Static Timing
- Errors and Warnings
 - Parser Messages
 - Synthesis Messages
 - Translation Messages
 - Map Messages
 - Place and Route Messages
 - Timing Messages
 - Bitgen Messages
 - All Implementation Messages
- Detailed Reports
 - Synthesis Report
 - Translation Report
 - Map Report
 - Place and Route Report
 - Post-PAR Static Timing Report
 - Power Report
 - Bitgen Report
- Secondary Reports
 - Enable Message Filtering
 - Optional Design Summary Contents
 - Show Clock Report
 - Show Failing Constraints
 - Show Warnings
 - Show Errors

Find: Delay

Design Summary (Implemented)

SRAM_ER_TCAM_1024x40 Project Status (11/20/2020 - 12:53:50)

Project File:	PAR.xise	Parser Errors:	No Errors
Module Name:	SRAM_ER_TCAM_1024x40	Implementation State:	Placed and Routed
Target Device:	xc5vfx330-2ff1760	•Errors:	No Errors
Product Version:	ISE 14.2	•Warnings:	5 Warnings (0 new)
Design Goal:	Balanced	•Routing Results:	All Signals Completely Routed
Design Strategy:	Vlmx Default (unlocked)	•Timing Constraints:	All Constraints Met
Environment:	System Settings	•Final Timing Score:	0 (Timing Report)

Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	20	207,360	1%	
Number used as Flip Flops	20			
Number of Slice LUTs	80	207,360	1%	
Number used as logic	80	207,360	1%	
Number using O6 output only	70			
Number using O5 and O6	10			
Number of occupied Slices	37	51,840	1%	
Number of LUT Flip Flop pairs used	80			
Number with an unused Flip Flop	60	80	75%	
Number with an unused LUT	0	80	0%	
Number of fully used LUT-FF pairs	20	80	25%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	0	207,360	0%	
Number of bonded IOBs	70	1,200	5%	
IOB Flip Flops	18			
Number of BlockRAM/BIPO	3	288	1%	

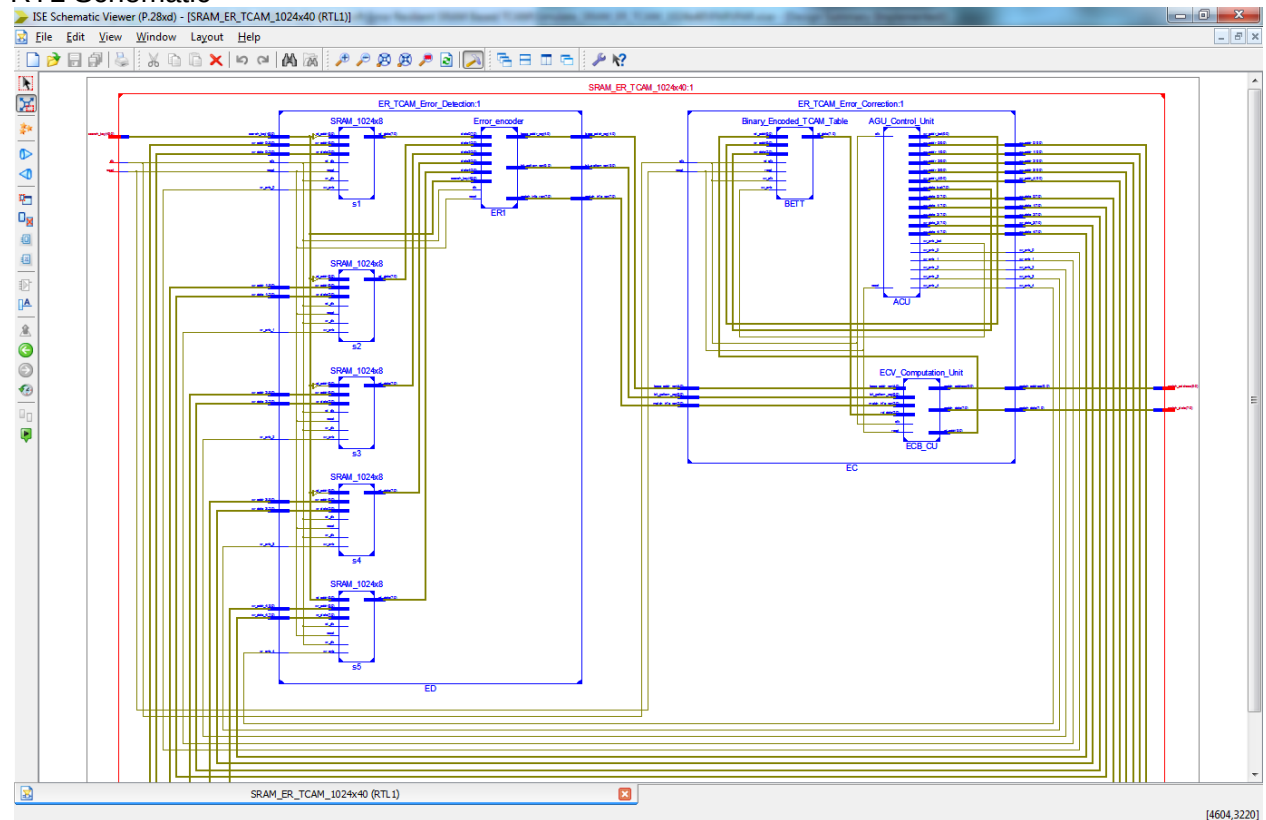
Console

Launching: "XPower Analyzer"

Process "XPower Analyzer" launched successfully

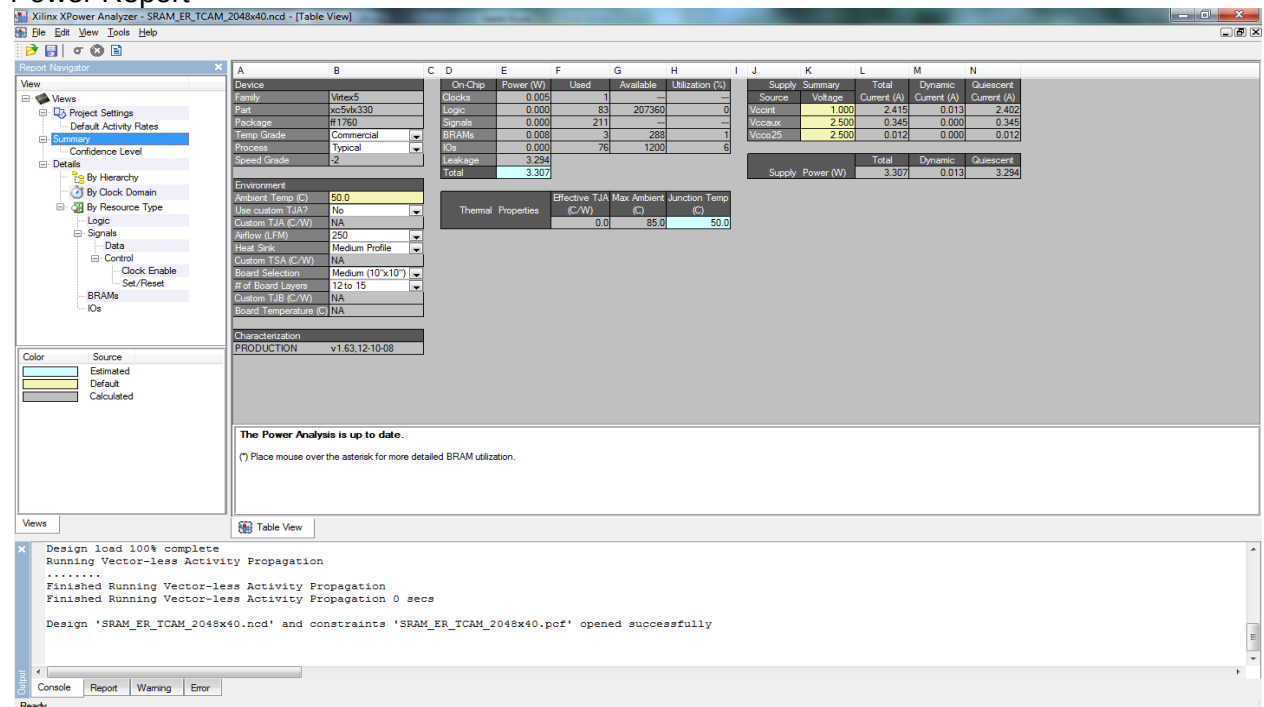
Console Errors Warnings Find in Files Results

RTL Schematic

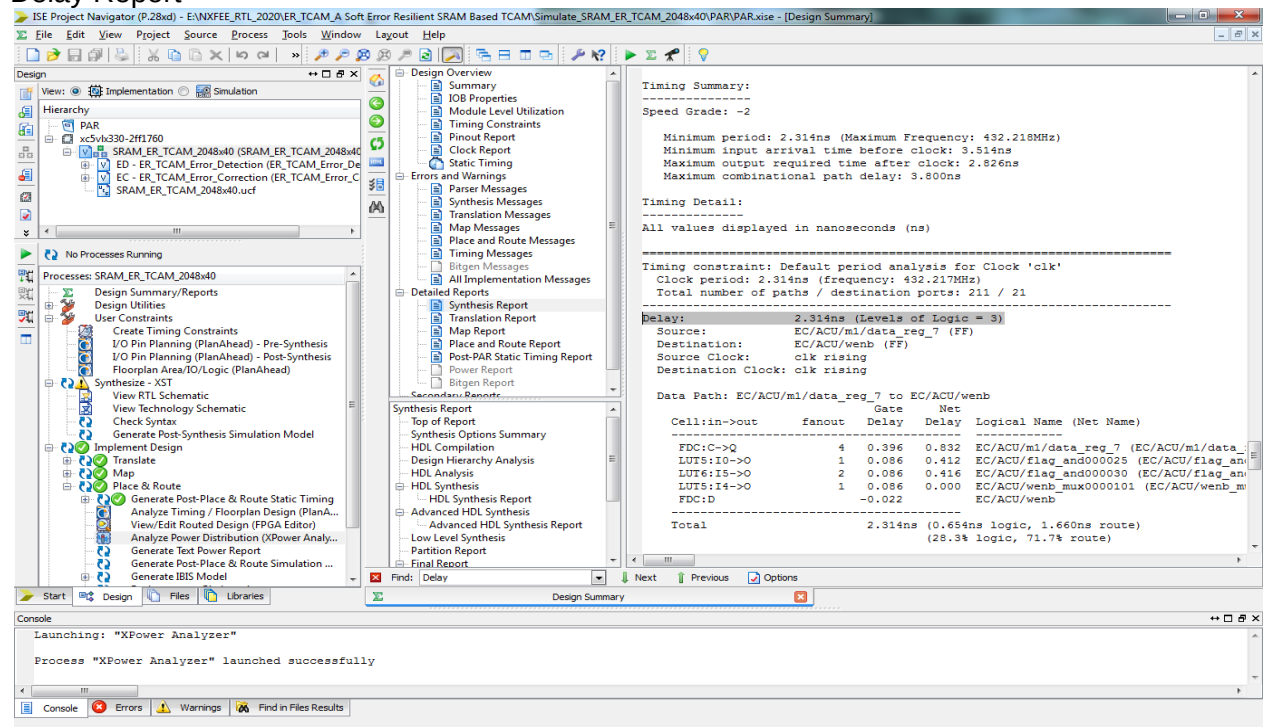


5.1.5 TCAM 2048X40

Power Report



Delay Report



Synthesized Report

ISE Project Navigator (P.28xd) - E:\NXP\FEE_RTL_2020\ER_TCAM_A_Soft Error Resilient SRAM Based TCAM\Simulate_SRAM_ER_TCAM_2048x40\PAR\PAR.xise - (Design Summary)

File Edit View Project Source Process Tools Window Layout Help

Design Overview

Design: Implementation Simulation

Hierarchy

- PAR
 - xc5vbx330-2ff1760
 - SRAM_ER_TCAM_2048x40 (SRAM_ER_TCAM_2048x40)
 - ED - ER_TCAM_Error_Detection (ER_TCAM_Error_Detection)
 - EC - ER_TCAM_Error_Correction (ER_TCAM_Error_Correction)
 - SRAM_ER_TCAM_2048x40.ucf

No Processes Running

Processes: SRAM_ER_TCAM_2048x40

- Design Summary/Reports
 - Design Utilities
 - User Constraints
 - Create Timing Constraints
 - I/O Pin Planning (PlanAhead) - Pre-Synthesis
 - I/O Pin Planning (PlanAhead) - Post-Synthesis
 - Floorplan Area/IO/Logic (PlanAhead)
 - Synthesize - XST
 - View RTL Schematic
 - View Technology Schematic
 - Check Syntax
 - Generate Post-Synthesis Simulation Model
 - Implement Design
 - Translate
 - Map
 - Place & Route
 - Generate Post-Place & Route Static Timing
 - Analyze Timing / Floorplan Design (PlanAhead)
 - View/Edit Routed Design (FPGA Editor)
 - Analyze Power Distribution (XPower Analyzer)
 - Generate Text Power Report
 - Generate Post-Place & Route Simulation ...
 - Generate IBIS Model

Design Properties

- Enable Message Filtering
- Optional Design Summary Contents
 - Show Clock Report
 - Show Failing Constraints
 - Show Warnings
 - Show Errors

Design Summary

SRAM_ER_TCAM_2048x40 Project Status

Project File:	PAR.xise	Parser Errors:	No Errors
Module Name:	SRAM_ER_TCAM_2048x40	Implementation State:	Placed and Routed
Target Device:	xc5vbx330-2ff1760	Errors:	No Errors
Product Version:	ISE 14.2	Warnings:	5 Warnings (0 new)
Design Goal:	Balanced	Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	Timing Constraints:	All Constraints Met
Environment:	System Settings	Final Timing Score:	0 (Timing Report)

Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	21	207,360	1%	
Number used as Flip Flops	21			
Number of Slice LUTs	83	207,360	1%	
Number used as logic	83	207,360	1%	
Number using O6 output only	72			
Number using O5 and O6	11			
Number of occupied Slices	35	51,840	1%	
Number of LUT Flip Flop pairs used	83			
Number with an unused Flip Flop	62	83	74%	
Number with an unused LUT	0	83	0%	
Number of fully used LUT-FF pairs	21	83	25%	
Number of unique control sets	2			
Number of slice register sites lost to control set restrictions	3	207,360	1%	
Number of bonded IOBs	76	1,200	6%	
IOB Flip Flops	19			
Number of BlockRAM/FIFO	3	288	1%	
Number using BlockRAM only	3			

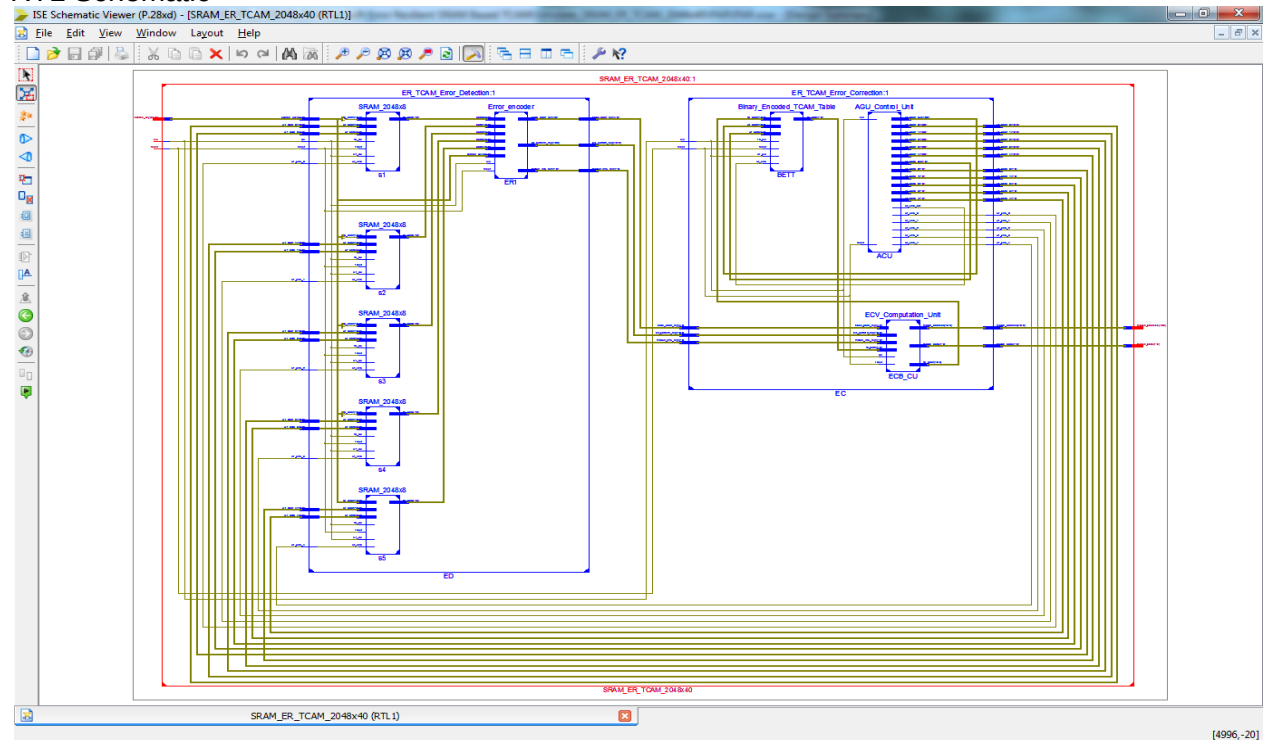
Console

Launching: "XPower Analyzer"

Process "XPower Analyzer" launched successfully

Console Errors Warnings Find in Files Results

RTL Schematic



5.2 DISCUSSION

5.1: Discussion Table

	ER-TCAM: A Soft-Error-Resilient SRAM-Based Ternary Content-Addressable Memory for FPGAs Synthesized on Xilinx Vertex-5 FPGA (xc5vlx330-2ff1760)				
TCAM Size	64x40	512x28	512x40	1024x40	2048x40
BRAM (18kb)	6	8	6	6	6
Slice Register	16	15	19	20	21
LUT	59	76	76	80	83
Occupied Slices	25	36	33	37	35
Number of IOBs	46	78	64	70	76
Delay (ns)	1.744	1.910	2.004	2.005	2.314
Power (W)	3.313	3.320	3.315	3.309	3.307

CHAPTER 6

SUMMARY AND CONCLUSIONS

This brief presents an error-detection and -correction technique for SRAM-based TCAMs, which makes the most of the redundant original TCAM content maintained on-chip for the update purposes to provide a low area, critical path delay, and response time protection. The proposed method uses single-bit parity to detect faults at a minimal cost of logic and critical path delay. The proposed error resiliency technique, called ER-TCAM, employs the binary-encoded TCAM table in SRAM-based TCAMs to correct errors. SRAMs implementing the TCAM function are available for search operations during the error-correction process carried out in the background, thus, the proposed error-correction technique does not affect the data path processing. The ER-TCAM achieved a search performance of up to 250 million searches per second with an EDD of 8 ns and a deterministic error-correction time of 260 ns when tested on the Artix-7 FPGA device. However, the error-correction time of other existing error-correction techniques is very high and non-deterministic. Compared with others, the ER-TCAM achieves at least two times better EDD. The ER-TCAM can be useful for network system or other system-on-chips that require SRAM-based TCAMs as a search engine for accelerating specific tasks.

REFERENCES

- [1] Afzaal. U, Lee. J. A, Ullah. I, and Ullah. Z, “DURE: An energy-and resource-efficient TCAM architecture for FPGAs with dynamic updates,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 6, pp. 1298–1307, Mar. 2019.
- [2] Baeg. S, Ilgon. K, and Ullah. Z, “Hybrid SRAM-based ternary material addressable partitioned memory,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 12, pp. 2969–2979, Dec. 2012.
- [3] Device Reliability Report, Xilinx, San Jose, CA, USA, 2018.
- [4] Fu. W, Li. T, and Sun. Z, “FAS: Using FPGA to accelerate and secure SDN software switches,” *Secur. Commun. Netw.*, vol. 2018, Jan. 2018, Art. no. 5650205.
- [5] Guan. H, He. P, Salamatian. K, Xie. G, and Zhang. W, “Partial order theory for fast TCAM updates,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 217–230, Feb. 2018.
- [6] Gupta. P, Shah. D and “Fast updating algorithms for TCAM,” *IEEE Micro*, vol. 21, no. 1, pp. 36–47, Jan. 2001.
- [7] Irfan. M and Ullah. Z, “G-AETCAM: Gate-based area-efficient ternary content-addressable memory on FPGA,” *IEEE Access*, vol. 5, pp. 20785–20790, 2017.
- [8] Jia. Y, Li. T, Yang. H, Wang. N, Wei. Y, and Zhao. H, “Investigation into SEU effects and hardening strategies in SRAM based FPGA,” in *Proc. 17th Eur. Conf. Radiat. Effects Compon. Syst. (RADECS)*, 2019, pp. 1–5.
- [9] Jiang. W, “Scalable memory with ternary content addressable implementation using FPGAs,” in *Proc. 9th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2013, pp. 71–82.
- [10] Keller, A. M and Wirthlin. M. J, “Impact of soft errors on largescale FPGA cloud computing,” in *Proc. ACM/SIGDA Int. Symp. FieldProgram. Gate Arrays*, 2019, pp. 272–281.
- [11] Lee. J. A, Ullah. I, and Ullah. Z, “Efficient TCAM design based on multipumping-enabled multiported SRAM on FPGAA,” *IEEE Access*, vol. 6, pp. 19940–19947, 2018.
- [12] Liu. H, Li. T, and Yang. H, “Design SEU hardened circuits for SRAM-based systems, as well as their characterization FPGA,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 6, pp. 1276–1283, Feb. 2019.
- [13] Maestro. J. A, Pontarelli. S, and Reviriego. P, “Parallel D-pipeline: A cuckoo hashing implementation for increased throughput,” *IEEE Trans. Comput.*, vol. 65, no.

1, pp. 326–331, Mar. 2015.

[14] Maestro. J. A, Ramos. A, Reviriego. P, and Toral. R. G, “An ALU security methodology for SRAM-based soft processors FPGAs,” *IEEE Trans. Comput.*, vol. 68, no. 9, pp. 1404–1410, Mar. 2019.

[15] Pontarelli. S, Reviriego. P, and Ullah. A, “Error detection and correction in SRAM emulated TCAMs,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 2, pp. 486–490, Feb. 2019.

[16] Pontarelli. S, Reviriego. P, and Ullah. A, “PR-TCAM: On Xilinx FPGAs, efficient TCAM emulation using partial reconfiguration,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1952–1956, Mar. 2019.

[17] Ternary Content Addressable Memory is a form of ternary content addressable memory (TCAM) that Search IP for SDNet V1.0; Xilinx Product Guide PG190, Xilinx, San Jose, CA, USA, Nov. 2017.

[18] Xilinx. (2016). 7 Series FPGAs Configurable Logic Block User Guide UG474 (V1.8). Accessed: Jun. 25, 2019. [Online]. Available: <http://www.xilinx.com>