Error Masking Technique For Register File In Xilinx FPGA

Steffi Mary Peter[1], Dr. Susan R. J.[2]
[1]PG Scholar, [2]Professor
Dept. of Electronics and Communication Engg.
Rajiv Gandhi Institute of Technology Kottayam, India
[1]steffimp94@gmail.com, [2]susanrj100@gmail.com

***Abstract*—Due to their flexibility, re-configurability and capacity, SRAM based FPGAs are becoming increasingly suitable for avionics and space applications. But soft errors are a major concern for space electronics. This work focus on a novel protection technique for a 64-bit dual port register file which utilizes the duplicated memories in an efficient manner to convert the inherent register file duplication into a real time fault tolerant memory structure for the soft processors. It uses parity-based error detection and switching (multiplexing) logic. This error masking approach can handle single bit errors and reduce the cost in terms of hardware overhead compared to the error correcting approaches like Hamming codes and Decimal Matrix codes. The delay of this error masking approach is slightly more which can be reduced by replacing the parity based error detection by hamming codes. These are simulated in Xilinx ISE Design Suite 14.1.**

***Index Terms*—Soft processors, Soft errors, Fault injection**

I. Introduction

Re-programmable and re-configurable capabilities of FPGAs have made them increasingly attractive as alternatives to Application Specific Integrated Circuits (ASICs) in space applications. SRAM-based FPGAs are used in space applications over the past decades. SRAM is a volatile storage format that needs to be reprogrammed at power-up. The ability to reprogram and the elimination of the large overhead cost make them attractive. Altera and Xilinx are the two major manufacturers of SRAM-based FPGAs[1].

SRAM-based FPGAs have a major disadvantage that they are vulnerable to soft errors or single event upset (SEU). It is caused by striking of ionizing particles which result in a change of state of a memory bit or flip-flop. There are mainly two types of soft errors for SRAM based FPGAs: 1) errors on the FPGA circuit elements and 2) on SRAM configuration memory[2]. The first type affects the logic gates, flip-flops, and embedded memories. The second type can modify the circuit implement on the FPGA (for example, can change a logic function).

Radiation has a negative impact on the lifespan, performance and reliability of the device or digital system. Soft errors have an adverse effect on register files. The register file has been largely ignored compared to main memories and cache structures due to its small area footprint. But register file is accessed very frequently and the soft errors occurred in the register file can create severe system problems. To study the effects of configuration upsets on the operation of an FPGA, the fault injection technique is mainly used. In this, an incorrect state is inserted into a system so as to monitor the impact of the fault in the system.

In this work, we discuss a dual port register file protection technique called the Error Masking technique. It is then compared with two existing techniques (Hamming Codes and Decimal Matrix Codes). Finally, we propose a technique to reduce the delay of the Error Masking technique.

II. Literature Review

This section gives a review of the existing methods in the field of register file protection in soft processors.

Triple module redundancy (TMR) is a technique effective against SEU. The effectiveness of TMR techniques with different voting schemes are discussed in [3]. 1) One-voter TMR: The design is replicated thrice and at the circuit output a majority voter is placed for performing a bit by bit voting. 2) Partitioned TMR: The design is divided into different partitions and each partition is replicated thrice. Each partition's output has a majority voter. 3) X-TMR: A commercial tool provided by Xilinx is used for hardening. Voters are inserted across each replica of the circuit to keep the state of FSM synchronized. These techniques are effective when a single SEU occurs in the configuration memory. One-voter TMR fails if an error occurs to two different replicas and partitioned TMR fails if an error occurs to two different replicas of the same partition. This model increases the area overhead and confirms that when large accumulation of error occurs the simple TMR techniques are ineffective.

A comparison of two soft error mitigation solutions, TMR and Hamming codes, for register files in ASIC is presented in [4]. Probability analysis is performed which discuss the overall improvement in bit-error rate for TMR and Hamming codes. It indicates a very small difference for word error probabilities between TMR and Ham (7, 4).It is observed from the results that the minimum area solution can be obtained from the Hamming code on the entire word as a single block and if the latency of the register file is more important, then TMR is the best solution. But the read access latency and area increase for TMR and SEC Hamming codes when applied to a 64-bit wide register file which cannot be tolerated in register files used in extremely pipelined high-speed microprocessors.

A method to construct SEC and SEC-DED codes that have less encoding and decoding delay is presented in [5]. This can be done by minimizing the number of ones in each row and column of the parity check matrix. This method uses some additional parity bits to do so. Then, the derived SEC codes and SEC-DED codes are compared with the Hamming SEC codes and the optimized SEC-DED codes respectively which shows there is a significant reduction in delay and area for the encoder and decoder. But these codes require more redundant bits compared to the traditional codes which limit their applications to large memories.

A technique called Self-Immunity[6] exploit the upper unused bits of a register by storing the corresponding SEC Hamming Code. For the value and its ECC to be stored together within the bit-width of a register, the required number of bits to represent the value should have an optimal value. The optimal value for 32-bit architectures is 26 and for 64-bit architectures is 57. Here the upper unused six bits of the register to used to store the corresponding ECC bits. This is called ”26-bit” values and register values which need more than 26 bits are ”over-26-bit” register values. To distinguish ”26-bit” register values from ”over-26-bit” register values, a self Q bit is associated with each register. Self Q bit is set to ’1’ indicating the existence of Self-Immunity and ’0’ indicating the absence of Self-Immunity. Due to the lesser ECC operations, the absence of additional storage for ECC and a less complex ECC generator, the power can be saved. However, if all the bits in the register are used, this technique can’t be used.

To protect registers, triple modular redundancy (TMR) is widely used. It reduces circuit delay but it requires thrice the original circuit area. An alternative to TMR, DMR+, is presented in [2]. This technique helps to achieve a reduction in the FPGA resources. This technique is used to protect flip-flops in Xilinx FPGAs. Also, a comparison between TMR and DMR+ is carried out. DMR+ gives a significant reduction in the number of flip-flops and LUTs needed. But this technique has a disadvantage that it increases the delay.

Hamming code is an error correcting code which is an improvement over the parity check method. They are widely used in computer memory because of their simplicity. In [7], the Hamming code is implemented in Verilog where 4-bit of information data is transmitted with 3 redundant bits. It also discusses the reason why Verilog language is preferred over VHDL for this method.

DMC[8] is used to protect memories from multiple bit errors. To obtain the maximum error detection capability, DMC is utilized. In this, divide-symbol and arrange the matrix

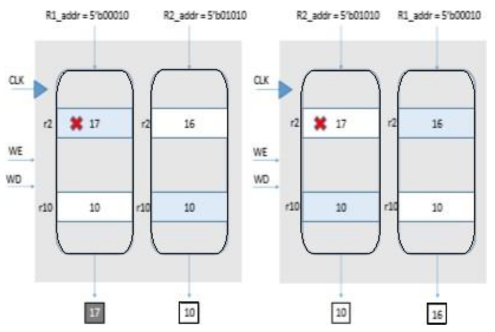


Fig. 1. Fault masking using switching of register file read addresses

is performed which is a logical implementation instead of a physical one. The main advantage of this technique is that it is able to detect even and odd number of errors. But its area and delay are slightly more than some of the existing techniques. The main drawbacks were increase in the delay and area overhead. The error masking technique for register file in Xilinx FPGA manages to protect the dual port register file of soft processors without significant time delay and with very little area overhead.

III. System Description

A. *Error Masking Technique*

To overcome the specified limitations, this work focus on a protection technique for a 64-bit dual port register file of a soft processor against single bit error. It utilizes parity-based error detection and a multiplexing logic to provide feed forward error-free results[9]. This technique protects a dual port register file of soft processors without significant time delay and with very little area overhead.

A dual port register file as two separate physical copies, RD1 and RD2, each one containing all the register values. If an instruction reads two operand registers and one of the parity is wrong, that means that the register associated with that parity check has an error. So the two operands are swapped in the read lines and their actual values are taken from the opposite register copy. Thus the error can be avoided. But it can only mask errors, it cannot be corrected. Since only a single error is considered, there will always be a valid copy of every register in the system. An example is shown in Fig. 1. Register r2 in RD1 has been affected by an error whose initial correct value was 16 changing its content to 17. But its second copy in RD2 is error free. An error detection logic is used to detect such an error and control the switching of read address lines. In this example, before the switching, r2 was read from the first copy RD1, but after switching r2 is now read from the error-free second copy RD2, thus avoiding the wrong value. Similarly, r10 was read before from RD2, but it is swapped to be read from RD1. Thus with this technique the system is able to get a good copy of both registers. As the register file is frequently updated and the corrupted content would be replaced by another value, this is not a problem.

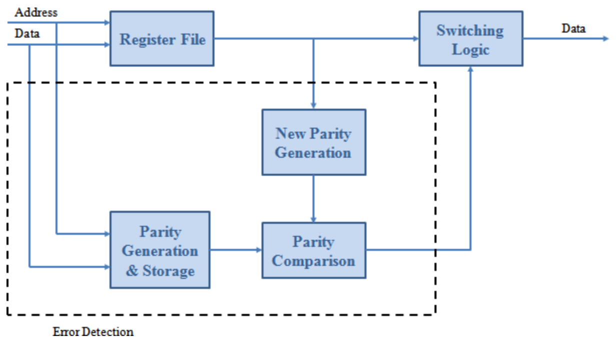


Fig. 2. Block diagram for Error Masking technique

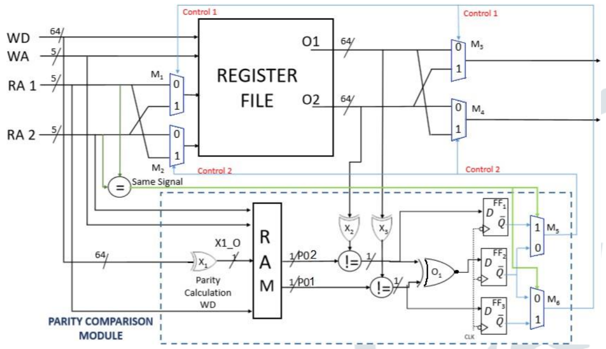
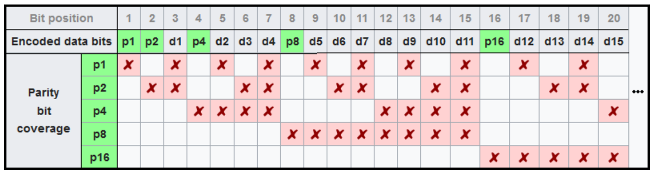


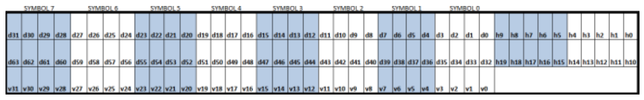
Fig. 3. Error Masking technique

The block diagram and circuit diagram for register file protection technique are presented in Fig. 2 and Fig. 3 respectively. It includes a parity generation, storage, and comparison module along with several multiplexers at the input and output of the register file. During the writing phase, the parity of the written word is generated by X1 and stored in the parity memory. The parity memory is not adding any resource overhead as it is implemented inside the same FPGA mapped register file memory structure. During the reading stage, the new parity, generated by X2 and X3, is compared with the stored parity bit. A parity check flag is generated by O1 if there is a mismatch. This parity check flag is then sampled by the negative edge triggered flip-flops followed by two multiplexers M5 and M6. The same signal will be logic 0 if different registers are to be read leading to the selection of FF2 output while the same signal will be logic 1 if same registers are to be read resulting in the selection of FF1 and FF3 outputs. If a fault is detected, switch the address and output lines using the multiplexers M1 and M2 at the inputs and M3 and M4 at the outputs of the memory. Thus the faulty copy is avoided by reading from the alternative copy. But a major drawback of this technique is that it cannot handle multiple bit errors in individual register contents as the parity-based detection is unable to detect errors that affect an even number of bits.



Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		...
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

Fig. 4. Parity bit calculation



SYMBOL 7				SYMBOL 6				SYMBOL 5				SYMBOL 4				SYMBOL 3				SYMBOL 2				SYMBOL 1				SYMBOL 0													
d31	d30	d29	d28	d27	d26	d25	d24	d23	d22	d21	d20	d19	d18	d17	d16	d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0
d63	d62	d61	d60	d59	d58	d57	d56	d55	d54	d53	d52	d51	d50	d49	d48	d47	d46	d45	d44	d43	d42	d41	d40	d39	d38	d37	d36	d35	d34	d33	d32	h19	h18	h17	h16	h15	h14	h13	h12	h11	h10
v31	v30	v29	v28	v27	v26	v25	v24	v23	v22	v21	v20	v19	v18	v17	v16	v15	v14	v13	v12	v11	v10	v9	v8	v7	v6	v5	v4	v3	v2	v1	v0										

Fig. 5. DMC logical organization of 64 bit data (k = 2 x 8 and m = 4)

B. Comparison With Existing Techniques

After the simulation of the error masking technique, a comparison of the error masking technique with two existing techniques is done. The two techniques taken for comparison are Hamming Codes and Decimal Matrix Codes.

1) Hamming Codes: Hamming codes are error correcting codes that can detect and correct single bit errors. This method was invented by Richard W. Hamming[7],[10]. The extended Hamming codes are single-error correcting and double-error detecting (SEC-DED) codes. The Hamming code technique uses redundant bits. The number of redundant bits depends on the number of data bits given by

$$2^p \geq p + m + 1 \qquad (1)$$

where p - no. of parity bits
m - no. of data bits

The parity bits are located at 2^x positions. The parity bits are calculated as shown in the Fig. 4. At first, check all of the parity bits for the presence of error. If all parity bits are the same as the previously calculated parity bits, then there is no error. Otherwise, the sum of the positions of the erroneous parity bits gives the erroneous bit. For example, if the parity bits in positions 1, 2 and 4 indicate an error, then bit 1+2+4=7 is the error bit. Suppose if only one parity bit indicates an error, then the parity bit itself is in error.

2) Decimal Matrix Code (DMC): DMC is able to correct multiple error bits. In this, divide-symbol and arrange-matrix are done[11], i.e., the N-bit word is divided into k symbols of m bits and these symbols are arranged in a k1 x k2 2-D matrix as shown below

$$N = k \times m \qquad (2)$$

where k = k1 x k2, k1 and k2 represent the numbers of rows and columns in the logical matrix respectively.

Fig. 5 shows the logical organization of 64-bit data where d0 to d63 are data bits, h0 to h19 are horizontal redundant bits and v0 to v31 are vertical redundant bits[12]-[14]. When data bits are fed into the DMC encoder, horizontal redundant

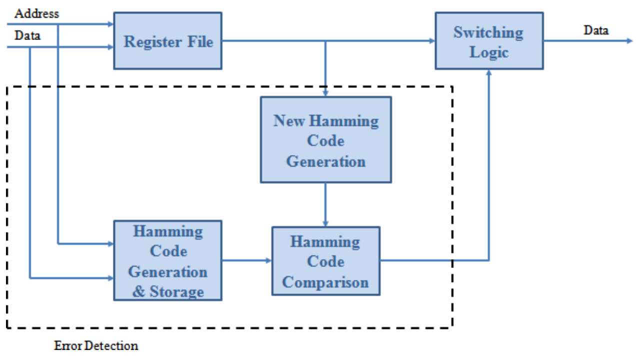


Fig. 6. Block diagram for Error Masking Technique using Hamming Codes

bits and vertical redundant bits are obtained from the DMC encoder as follows

$$h4h3h2h1h0 = d3d2d1d0 + d19d18d17d16 \quad (3)$$

$$h9h8h7h6h5 = d7d6d5d4 + d23d22d21d20 .. \quad (4)$$

$$v0 = d0 \oplus d32 \quad (5)$$

$$v1 = d1 \oplus d33 ..... \quad (6)$$

Let the received data be d0' to d63' and redundant bits be h0' to h19' and v0' to v31'. To check for errors, check all of the syndrome bits which can be calculated by

$$\Delta h4h3h2h1h0 = h4h3h2h1h0' - h4h3h2h1h0 \quad (7)$$

$$s0 = v0' \oplus v0 ..... \quad (8)$$

If all syndrome bits are zero, there is no error otherwise error is present which can be corrected by

$$d0_{correct} = d0' \oplus s0 \quad (9)$$

C. Proposed Method

The error masking technique when compared with Hamming codes and DMC is area efficient. But in terms of delay, this error masking technique has a delay slightly more than that of Hamming codes. In order to reduce its delay, some changes have made in the parity-based error detection.

The block diagram of the Error Masking Technique using Hamming Codes is shown in Fig. 6. It is proposed to reduce the delay of the error masking technique. It is done by replacing the parity-based error detection using Hamming codes. Instead of using parity bits for error detection, Hamming codes are used for error detection i.e. Hamming codes are stored in the RAM for error detection. Since a 64-bit dual port register file is to be protected, 71-bit Hamming code is used for error detection i.e. 71-bit Hamming code is stored in RAM.

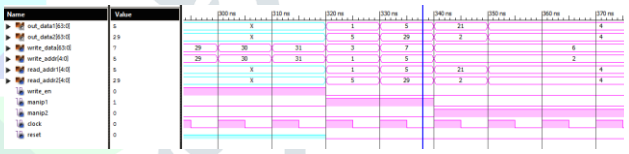
IV. RESULTS

This section discusses the results that were obtained in the Error Masking technique, its comparison with Hamming Codes and Decimal Matrix Codes and the Error Masking technique using Hamming Codes. These are carried out for a dual port register file with 64-bit width and depth of 32 registers. These were simulated in Xilinx ISE Design Suite 14.1.

A. Simulation and Synthesis Results of Error Masking Technique

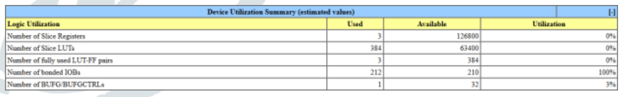
Fig. 7 shows the simulation result of the Error Masking technique for a 64-bit dual port register file. When write_en is high, the 64-bit data (write_data) are stored into the specified locations given by write_addr. A copy is also saved into another 32 registers. When write_en is low and manip1 is high, an error is introduced into the specified location in the first copy. Here an error is introduced in the first register in the first copy i.e. ”1” has now become ”3”. Even though there is an error at the first register of the first register file, the correct data ”1” is obtained at out_data1. Some other examples of this error masking technique are also shown in this figure.

Fig. 8 and Fig. 9 shows the device utilization summary and delay report of the Error Masking technique for a 64-bit dual port register file respectively. From the device utilization summary, it is clear that the Error Masking technique is area efficient. The delay report shows that the Error Masking technique has a delay of 4.159ns.



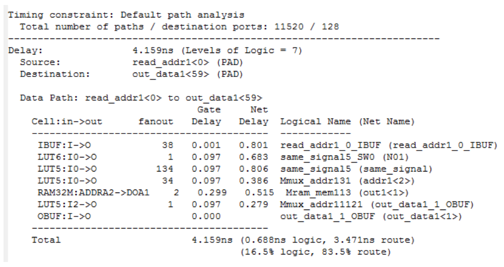
Name	Value
out_data1[63:0]	5
out_data2[63:0]	29
write_data[63:0]	7
write_addr[4:0]	5
read_addr1[4:0]	5
read_addr2[4:0]	29
write_en	0
manip1	1
manip2	0
clock	0
reset	0

Fig. 7. Simulation result of the Error Masking technique



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	3	126800	0%
Number of Slice LUTs	384	63400	0%
Number of fully used LUT-FF pairs	3	384	0%
Number of bonded IOBs	212	210	100%
Number of BUFG/BUFGCTRLs	1	32	3%

Fig. 8. Device utilization summary of the Error Masking technique



```
Timing constraint: Default path analysis
  Total number of paths / destination ports: 11520 / 128
---------------------------------------------------------------------------
Delay:                 4.159ns (Levels of Logic = 7)
  Source:              read_addr1<0> (PAD)
  Destination:         out_data1<59> (PAD)

  Data Path: read_addr1<0> to out_data1<59>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            38   0.001   0.801  read_addr1_0_IBUF (read_addr1_0_IBUF)
     LUT6:I0->O            1   0.097   0.683  same_signal5_SW0 (N01)
     LUT5:I0->O          134   0.097   0.806  same_signal5 (same_signal)
     LUT5:I0->O           34   0.097   0.386  Mmux_addr131 (addr1<2>)
     RAM32M:ADDRA2->DOA1    2   0.299   0.515  Mram_mem113 (out1<1>)
     LUT5:I2->O            1   0.097   0.279  Mmux_addr11121 (out_data1_1_OBUF)
     OBUF:I->O                 0.000          out_data1_1_OBUF (out_data1<1>)
    ----------------------------------------
    Total                      4.159ns (0.688ns logic, 3.471ns route)
                                       (16.5% logic, 83.5% route)
```

Fig. 9. Delay report of the Error Masking technique

B. Simulation and Synthesis Results of Hamming Codes

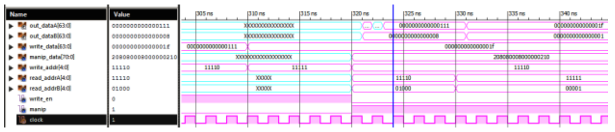
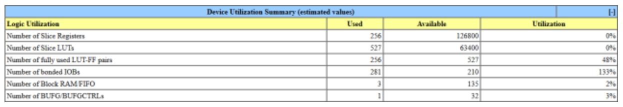
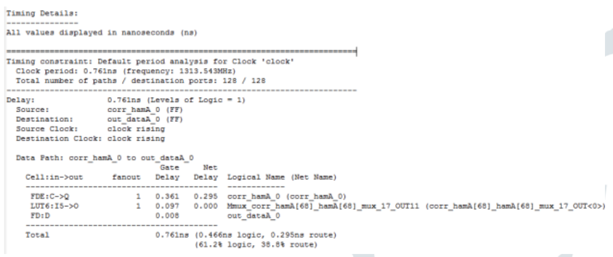


Fig. 10. Simulation result of the Hamming Code technique



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	256	126800	0%
Number of Slice LUTs	527	63400	0%
Number of fully used LUT-FF pairs	256	527	48%
Number of bonded IOBs	281	210	133%
Number of Block RAM/FIFO	3	135	2%
Number of BUFG/BUFGCTRLs	1	32	3%

Fig. 11. Device utilization summary of the Hamming Code technique



```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clock'
  Clock period: 0.761ns (frequency: 1313.543MHz)
  Total number of paths / destination ports: 128 / 128
-------------------------------------------------------------------------
Delay:               0.761ns (Levels of Logic = 1)
  Source:            corr_hamA_0 (FF)
  Destination:       out_dataA_0 (FF)
  Source Clock:      clock rising
  Destination Clock: clock rising

  Data Path: corr_hamA_0 to out_dataA_0
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q              1   0.361   0.295  corr_hamA_0 (corr_hamA_0)
     LUT6:I5->O            1   0.097   0.000  Mmux_corr_hamA[68]_hamA[68]_mux_17_OUT11 (corr_hamA[68]_hamA[68]_mux_17_OUT<0>)
     FD:D                      0.008          out_dataA_0
    ----------------------------------------
    Total                      0.761ns (0.466ns logic, 0.295ns route)
                                       (61.2% logic, 38.8% route)
```

Fig. 12. Delay report of the Hamming Code technique

In Fig. 10, when write_en is high and manip is low the 64-bit data(write_data) is stored in the specified locations given by write_addr. The error is introduced in the specified location as manip gets high. As the addresses to be read are given (read_addrA and read_addrB), the corrected output is obtained at out_dataA and out_dataB.

Fig. 11 and Fig. 12 show the device utilization summary and delay report of 64-bit dual port register file using Hamming Code respectively. The delay report shows that the error masking technique has a delay of 0.761ns.

C. Simulation and Synthesis Results of Decimal Matrix Code

In Fig. 13, when write_en is high and manip is low the 64-bit data (write_data) is stored in the specified locations given by write_addr. The error is introduced in the specified location as manip gets high. As the addresses to be read are given (read_addr1 and read_addr2), the corrected output is obtained at out_data1 and out_data2.

Fig. 14 and Fig. 15 show the device utilization summary and delay report of 64-bit dual port register file using Decimal Matrix Code respectively. The delay report shows that the error masking technique has a delay of 7.210ns.

D. Simulation and Synthesis Results of Error Masking Technique using Hamming Codes

Fig. 16 shows the simulation result of the Error Masking technique using Hamming Codes for a 64-bit dual port register file. When write_en is high, the 64-bit data (write_data) are

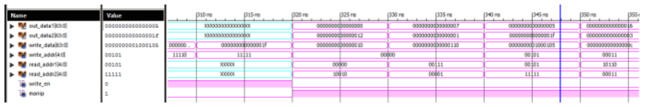
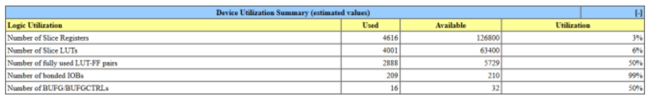
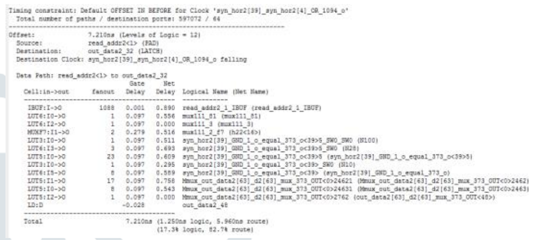


Fig. 13. Simulation result of the Decimal Matrix Code



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	4616	126800	3%
Number of Slice LUTs	4001	63400	6%
Number of fully used LUT-FF pairs	2888	5729	50%
Number of bonded IOBs	209	210	99%
Number of BUFG/BUFGCTRLs	16	32	50%

Fig. 14. Device utilization summary of the Decimal Matrix Code



```
Timing constraint: Default OFFSET IN BEFORE for Clock 'syn_hor2[39]_syn_hor2[4]_OR_1094_o'
  Total number of paths / destination ports: 597072 / 64
-------------------------------------------------------------------------
Offset:              7.210ns (Levels of Logic = 12)
  Source:            read_addr2<1> (PAD)
  Destination:       out_data2_32 (LATCH)
  Destination Clock: syn_hor2[39]_syn_hor2[4]_OR_1094_o falling

  Data Path: read_addr2<1> to out_data2_32
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O          1088   0.001   0.890  read_addr2_1_IBUF (read_addr2_1_IBUF)
     LUT6:I0->O            1   0.097   0.556  mux111_81 (mux111_81)
     LUT6:I2->O            1   0.097   0.000  mux111_3 (mux111_3)
     MUXF7:I1->O           2   0.279   0.516  mux111_2_f7 (h22<16>)
     LUT3:I0->O            1   0.097   0.511  syn_hor2[39]_GND_1_o_equal_373_o<39>5_SW0_SW0 (N100)
     LUT6:I3->O            3   0.097   0.693  syn_hor2[39]_GND_1_o_equal_373_o<39>5_SW0 (N28)
     LUT5:I0->O           23   0.097   0.609  syn_hor2[39]_GND_1_o_equal_373_o<39>5 (syn_hor2[39]_GND_1_o_equal_373_o<39>5)
     LUT3:I0->O            1   0.097   0.295  syn_hor2[39]_GND_1_o_equal_373_o<39>_SW0 (N10)
     LUT6:I5->O            8   0.097   0.589  syn_hor2[39]_GND_1_o_equal_373_o<39> (syn_hor2[39]_GND_1_o_equal_373_o)
     LUT5:I1->O           17   0.097   0.758  Mmux_out_data2[63]_d2[63]_mux_373_OUT<0>24621 (Mmux_out_data2[63]_d2[63]_mux_373_OUT<0>2462)
     LUT5:I0->O            8   0.097   0.543  Mmux_out_data2[63]_d2[63]_mux_373_OUT<0>24631 (Mmux_out_data2[63]_d2[63]_mux_373_OUT<0>2463)
     LUT5:I2->O            1   0.097   0.000  Mmux_out_data2[63]_d2[63]_mux_373_OUT<0>2762 (out_data2[63]_d2[63]_mux_373_OUT<48>)
     LD:D                     -0.028          out_data2_48
    ----------------------------------------
    Total                      7.210ns (1.250ns logic, 5.960ns route)
                                       (17.3% logic, 82.7% route)
```

Fig. 15. Delay report of the Decimal Matrix Code

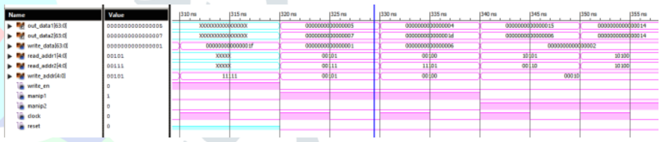
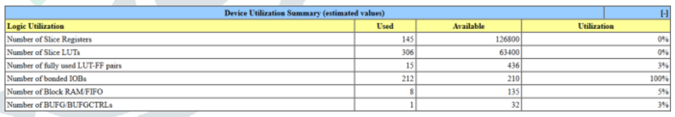
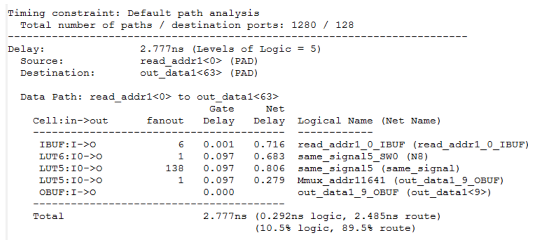


Fig. 16. Simulation result of the Error Masking Technique using Hamming Codes



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	145	126800	0%
Number of Slice LUTs	306	63400	0%
Number of fully used LUT-FF pairs	15	436	3%
Number of bonded IOBs	212	210	100%
Number of Block RAM/FIFO	8	135	5%
Number of BUFG/BUFGCTRLs	1	32	3%

Fig. 17. Device utilization summary of the Error Masking Technique using Hamming Codes



```
Timing constraint: Default path analysis
  Total number of paths / destination ports: 1280 / 128
-------------------------------------------------------------------------
Delay:               2.777ns (Levels of Logic = 5)
  Source:            read_addr1<0> (PAD)
  Destination:       out_data1<63> (PAD)

  Data Path: read_addr1<0> to out_data1<63>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             6   0.001   0.716  read_addr1_0_IBUF (read_addr1_0_IBUF)
     LUT6:I0->O            1   0.097   0.683  same_signal5_SW0 (N8)
     LUT5:I0->O          138   0.097   0.806  same_signal5 (same_signal)
     LUT5:I0->O            1   0.097   0.279  Mmux_addr11641 (out_data1_9_OBUF)
     OBUF:I->O                 0.000          out_data1_9_OBUF (out_data1<9>)
    ----------------------------------------
    Total                      2.777ns (0.292ns logic, 2.485ns route)
                                       (10.5% logic, 89.5% route)
```

Fig. 18. Delay report of the Error Masking Technique using Hamming Codes

TABLE I
COMPARISON SUMMARY

	EM	HC	DMC	EM using HC
No. of slice registers (out of 126800)	3	256	4614	145
No. of slice LUTs (out of 63400)	384	527	4001	306
No. of fully used LUT-FF pairs (out of)	3 (384)	256 (527)	2888 (5729)	15 (436)
No. of bonded IOBs (out of 210)	212	281	209	212
Total delay of the system (in ns)	4.159	0.761	7.210	2.777

stored into the specified locations given by write_addr. A copy is also saved into another 32 registers. When write_en is low and manip1 is high, an error is introduced into the specified location in the first copy. Here an error is introduced in the first register in the first copy i.e. ”5” has now become ”1”. Even though there is an error at the first register of the first register file, the correct data ”5” is obtained at out_data1. Some other examples of this error masking technique are also shown in this figure.

Fig. 17 and Fig. 18 shows the device utilization summary and delay report of the Error Masking technique using Hamming Codes for a 64-bit dual port register file respectively. From the device utilization summary, it is clear that the Error Masking technique using Hamming Codes is area efficient. The delay report shows that the Error Masking technique using Hamming Codes has a delay of 2.777ns which is lesser than the Error Masking technique.

E. Area and Delay Comparison

Table I shows the area and delay comparison between the Error Masking technique, Hamming Code technique, Decimal Matrix Code technique and Error Masking technique using Hamming Codes. From the table, it is clear that the Error Masking technique is area efficient but has a delay greater than that of the Hamming Codes. The delay of the Error Masking technique can be reduced by the Error Masking technique using Hamming Codes. This technique is also area efficient.

V. CONCLUSION

An error masking technique for a 64-bit dual port register file is discussed here. This technique uses parity based error detection and switching logic. A literature survey on the related work was carried out. This technique was simulated in Xilinx ISE Design Suite 14.1 and was used for a register file with 64-bit width and depth of 32 registers. The results show that the single bit error in the register file had been masked and the correct copy was obtained at the output. A comparison of this error masking technique was done with two existing error correcting techniques, Hamming codes and DMC. The comparison result shows that the error masking technique is efficient in terms of area but has delay slightly more than that of Hamming code. So in order to reduce the delay, the parity-based error detection in the error masking technique was replaced by Hamming codes for error detection. From the results, it is clear that error masking using hamming codes is efficient in terms of both area and delay.

REFERENCES

[1] F. Brosser and E. Milh, "SEU Mitigation Techniques for advanced Reprogrammable FPGA in Space," *Master's thesis in Embedded Electronic System Design, Department of Computer Science and Engineering, Chalmers University Of Technology, Gothenburg, Sweden*, 2014.

[2] Reviriego P., Demirci M., Tabero J., Regadio A., and Maestro J. A., "DMR +: An efficient alternative to TMR to protect registers in Xilinx FPGAs," *Microelectronics Reliability*, vol. 63, pp. 314-318, 2016.

[3] A. Manuzzato, S. Gerardin, A. Paccagnella, L. Sterpone, and M. Violante, "Effectiveness of TMR-based techniques to mitigate alpha-induced SEU accumulation in commercial SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 1968-1973, Aug 2008.

[4] R. Naseer, R. Z. Bhatti, and J. Draper, "Analysis of soft error mitigation techniques for register files in IBM cu-08 90nm technology," *in 2006 49th IEEE International Midwest Symposium on Circuits and Systems*, vol. 1, pp. 515-519, Aug 2006.

[5] P. Reviriego, S. Pontarelli, J.A. Maestro, and M. Ottavi, "A method to construct low delay single error correction codes for protecting data bits only," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 3, pp. 479-483, Mar 2013.

[6] H. Amrouch and J. Henkel, "Self-immunity technique to improve register file integrity against soft errors," *in 2011 24th International Conference on VLSI Design*, pp. 189-194, Jan 2011.

[7] N. Shep and P.H. Bhagat, "Implementation of Hamming code using VLSI," *International Journal of Engineering Trends and Technology (IJETT)*, vol. 4, Issue 3, pp. 186-190, 2013.

[8] E. Abinaya, Dr. D. Somasundareswari, and S. Mathan Prasad, "Implementation of a Decimal Matrix Code for Correcting Multiple Cell Upsets in SRAM based FPGA Memories," *International Journal of advanced research trends in engineering and technology*, pp. 87-93, Feb 2015.

[9] A. Ramos, A. Ullah, P. Reviriego, and J. A. Maestro, "Efficient Protection of the Register File in Soft-processors Implemented on Xilinx FPGAs," *IEEE Transactions on computers*, 2017.

[10] Anoop H. K., S. K. Panda and Vasudeva G., "Implementation of single bit Error detection and Correction using Embedded hamming scheme," *International Journal of Modern Trends in Engineering and Research (IJMTER)*, Vol. 3, Issue 5, pp. 88-94, May 2016.

[11] T. Maheswari and P. Sukumar, "Error Detection and Correction in SRAM Cell Using Decimal Matrix Code," *IOSR Journal of VLSI and Signal Processing*, Vol. 5, Issue 1, pp. 09-14, Jan - Feb 2015.

[12] Y. V. Subbarao and C. H. Kanakalingeswararao, "Design and Implementation of Decimal Matrix Code for Correcting Cell Upsets in Memories," *International Journal of Emerging Technology in Computer Science and Electronics*, Vol. 9, Issue 3, pp. 36-42, Sep 2014.

[13] H. K. Padi and L. Narayanarao, "Implementation of 64-bit Decimal Matrix Code for correcting Cell Upsets in Static Random Access Memories," *International Journal of Emerging Engineering Research and Technology*, Vol. 2, Issue 5, pp. 179-187, Aug 2014.

[14] K. Madhuri and V. Thrimurthulu, "Implementation of Decimal matrix code for correcting cell upsets in static random access memories," *International Journal of Electrical, Electronics and Data Communication*, Vol. 2, Issue-10, pp.72-76, Oct 2014.