# A Multi-bit Error Tolerant Register File for a High Reliable Embedded Processor

Siamak Esmaeeli, Morteza Hosseini, Bijan Vosoughi Vahdat, Bizhan Rashidian
Department of Electrical Engineering, Sharif University of Technology
Tehran, Iran
E-mail:{esmaeeli, smhosseini}@ee.sharif.edu, {vahdat, rashidia}@sharif.edu

*Abstract*— **The vulnerability of microprocessors to soft errors is increasing due to continuous shrinking in fabrication process. Recent studies show that 1-5% of the SEUs (single event upset) can cause MBUs (multiple bit upsets). The probability of MBU generation due to SEU is increasing because of the reduction in minimum energy required to flip a memory bit in modern technologies. Register file is the most sensitive component in a microprocessor. In this paper, we present an innovative way to protect registers in a 64-bit register file for a RISC processor using extended Hamming (8, 4) code (SEC-DED code) and narrow-width values. A narrow-width value can be represented by half number of bits of the register width. Two additional bits for each data register have been used to store the information for a narrow-width value. Each 64-bit data in register file has its unique 64-bit extended Hamming code that is stored in another register file in a bit-interleaved manner. Two copies of narrow-width values can be stored in one register and each copy has its unique extended Hamming code in other register file. Proposed method has been tested using fault injection simulation with SPEC2000 benchmarks. Error probability of a word that stores generated values for register file in SPEC2000 benchmarks and is protected with proposed method is less than the error probability of the same word that is protected with TMR or various extended Hamming codes. The implementation on a Xilinx Virtex-4 FPGA shows that the area overhead of a register file with 64-bit wide and more than 64-word entry that is protected with proposed method is less than the area overhead of the same register file that is protected with TMR. Error detection and correction is performed in parallel with execute stage to prevent performance degradation. More than 99% of errors in adjacent 32 bits in data or extended Hamming code registers can be corrected with the proposed method. Presented method employs pure combinational logics and can be used for 16-bit and 32-bit register files too.**

## I. INTRODUCTION

Energetic particles – such as neutron and alpha particles – lose energy and generate electron-hole pairs when they strike a semiconductor device. Charges can be collected on transistor source and diffusion nodes. Adequate amount of collected charge can change the state of a device by changing the logical value of its components such as SRAM, Flip-Flop etc. Changing the state of a device introduces a logical fault or single event upset (SEU) into the circuit [1].

Future microprocessors will be more vulnerable to soft errors because of improvement in silicon technologies [1] [2]. Increasing vulnerability of microprocessors is due to the decreasing logic depth, reducing capacitance of nodes, lowering supply voltage, increasing the clock frequency of the system and the high integration density in modern technologies [3].

1-5% of SEUs can cause MBUs [12], and depending on the technology and the particle that strikes the device several types of multiple bit errors may be generated [13] [14] [15]. Also the soft error rate of a constant area of SRAM array in modern technologies is more than soft error rate in old technologies [3]. Therefore, in future technologies the multiple bit errors will be significant. So an approach that can detect and correct many errors in a register while keeping the performance and area overhead in acceptable range is required.

The most sensitive component to soft errors in a microprocessor is the register file. The probability of data error generation due to SEUs in register file is high [4] [5]. In this paper we propose an approach that can improve the soft error protection of register file by employing narrow-width values and extended Hamming (8, 4) code. We have assumed that the register file has 64-bit registers, two read ports and one write port.

## II. RELATED WORKS

There are many fault tolerant techniques that are used in register file of embedded processors such as using parity check and Hamming code (extended Hamming code) [6] [7], and using narrow-width values [8]. Narrow-width values can be represented by half number of bits of full data width. The method introduced in [8] can correct limited errors in narrow-width values, and errors in normal-width values can just be detected and handled with operating system through an exception. Normal-width values should be represented by more than half number of bits of full data width. In [17] 4 parity bits have been used to detect up to 4 bit errors.

In [18] Triple Modular Redundancy (TMR) technique and various Hamming codes for soft error mitigation in a 64-bit wide, 32-word entry register file have been analyzed. Experimental results show that error probability and performance degradation for TMR are less than Hamming codes. But TMR technique incurs 204% area penalty.

In superscalar processors unused physical registers can be used as copy of actively used physical registers to increase the reliability of the register file [5]. But this technique and similar techniques cannot be employed in embedded processors.

## III. NARROW-WIDTH VALUES

Many operands and result values in datapath of a microprocessor can be represented with fewer bits than full data width [9] [10] [11]. In this paper a narrow-width value is a value that can be represented by 32 bits, and a normal-width value is a value that should be represented by more than 32 bits. If upper 32 bits of a value are all zeros or all ones, then it is called a narrow-width value. In other words, lower 32 bits of a 64-bit narrow-width value represent the whole value.

978-1-4577-1846-5/11/$26.00 ©2011 IEEE

Fig. 1 shows the width distribution of the values of the SPEC2000 benchmarks that generated for registers in register file. On the average, about 85% of all values can be represented by 32 bits. In superscalar microprocessors values can be packed within a single 64-bit physical register [9] [10] [11].

Upper 32 bits in narrow-width values that contain 32 identical bits (zero or one), can be replaced with lower 32 bits. So we can store 2 copies of lower 32 bits of a narrow-width value in a register. Having 2 copies from the lower 32 bits can lead to a more efficient SEU protection. We can use 2 bits to distinguish between registers that contain narrow-width values and other registers. One bit shows that whether register contains a narrow-width value and another bit shows that the upper 32 bits are all zeros or all ones. If first bit is zero, we can ignore second bit and normally read the value from register.

We assume that the register file is used in a microprocessor with 5 pipeline stages: 1. Fetch, 2. Decode, 3. Execute, 4. Memory Access and 5. Write-back. Recognition of narrow-width values that are produced in execute stage can be done simultaneously with memory access stage without any performance degradation. We used a unit called "Narrow-width Value Recognition" to detect narrow-width values. If the input value of this unit has narrow width, the unit will produce a 64-bit output using lower 32 bits in a bit-interleaved manner. The output structure of this unit is illustrated in Fig. 3. This unit also produces 2 bits that first bit shows whether the value has narrow width and another bit shows that the upper 32 bits are all zeros or all ones. If the input of this unit isn't a narrow-width value, first bit will be zero and another bit should be ignored. Narrow-width values are reconstructed when they are read from register file in decoding stage. We employed a unit called "Narrow-width Value Reconstruction" to reconstruct the narrow-width values. Fig. 2 shows the datapath and the pipeline stages of a microprocessor with "Narrow-width Value Recognition" and "Narrow-width Value Reconstruction" units.

Narrow-width values should not be employed in bypass network. The "Narrow-width Value Reconstruction" unit can reduce the maximum frequency of the system. The structure of the "Narrow-width Value Reconstruction" and the "Narrow-width Value Recognition" units are illustrated in Fig. 3.

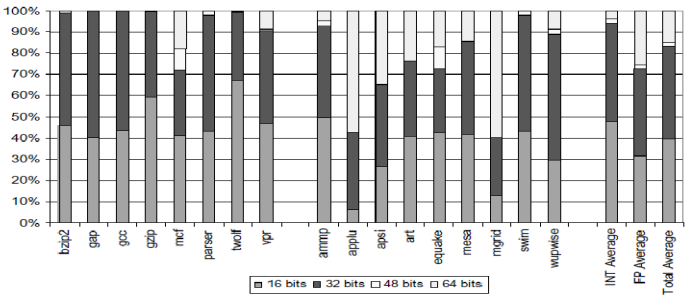


Figure 1. Width Distribution of Generated Register Values of The SPEC2000 Benchmarks.

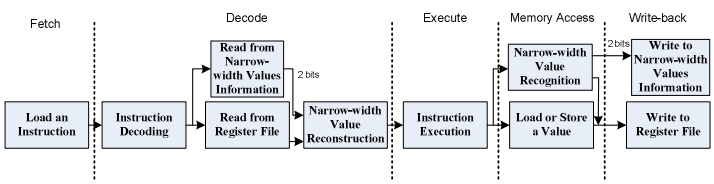


Figure 2. Datapath and The Pipeline Stages of a Microprocessor With Narrow-width Value Recognition and Reconstruction Units.

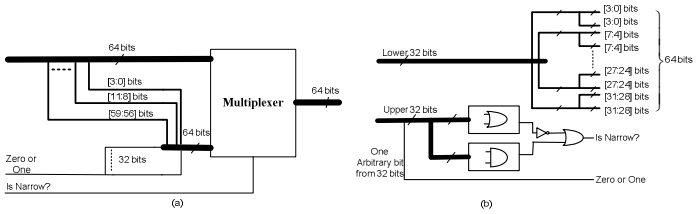


Figure 3. (a) Structure of "Narrow-width Value Reconstruction" Unit (b) Structure of "Narrow-width Value Recognition" Unit.

As depicted in Fig. 3, the logic circuit of the "Narrow-width Value Reconstruction" unit is simple. So the performance degradation due to propagation delay of this unit is not significant. If "Narrow-width Value Recognition" unit detects that its input has narrow width, the 64-bit output of this unit that is constructed in a bit-interleaved manner should be sent to write-back stage. In next sections we will show that using narrow-width values can improve the SEU protection efficiency of a register file.

## IV. EXTENDED HAMMING CODE

Hamming code [16] is an error detecting and correcting code that can correct all single bit errors within a codeword. By adding an overall parity bit, it can also detect double-bit errors. The new code is referred to extended Hamming code (SEC-DED) [16]. If we consider parity bits and data bits as a single codeword, the position of each bit is described as follows: bit positions that are powers of two (1, 2, 4, 8, etc) are dedicated to parity bits, and other positions (3, 5, 6, 7, etc) belong to data bits. Each parity bit protects some bits of data and its position determines the subset of data bits that are to be protected.

In Hamming codes, $P_i$ protects $D_j$ if and only if the $\log_2(i)$-th bit of the binary representation of $j$ is one e.g. $P_4$ protects $D_5, D_6$, etc and $D_7$ is protected by $P_4, P_2$ and $P_1$.

Among the great family of Hamming codes, extended Hamming (8, 4) code shows symmetric behavior between subset of parity bits and subset of data bits. Information redundancy is obviously 100%. Separating parity bits and data bits into two subsets, one subset is a nibble of data and another is a nibble of parity. They can be assumed as a twin, and each of them alone is representative for original data. Equation (1) shows one to one relationship between two subsets of data and parity bits in extended Hamming (8, 4) code. $P_8$ is the overall parity bit.

$$\begin{cases} P_1 = D_3 \hat{} D_5 \hat{} D_7 \\ P_2 = D_3 \hat{} D_6 \hat{} D_7 \\ P_4 = D_5 \hat{} D_6 \hat{} D_7 \\ P_8 = P_1 \hat{} P_2 \hat{} D_3 \hat{} P_4 \hat{} D_5 \hat{} D_6 \hat{} D_7 = D_3 \hat{} D_5 \hat{} D_6 \end{cases} \Rightarrow \begin{cases} D_3 = P_1 \hat{} P_2 \hat{} P_8 \\ D_5 = P_1 \hat{} P_4 \hat{} P_8 \\ D_6 = P_2 \hat{} P_4 \hat{} P_8 \\ D_7 = P_1 \hat{} P_2 \hat{} P_4 \end{cases} \quad (1)$$

Fig. 4 shows the units related to the extended Hamming (8, 4) code. The bit address decoder unit provides the address of the bit that is faulty and a signal to report the occurring of double error.

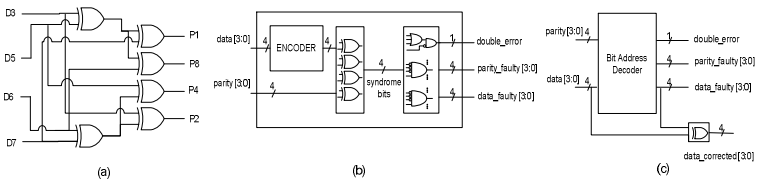


Figure 4. The Units Related to Extended Hamming (8, 4) Code: (a) Encoder Unit (b) Bit Address Decoder (c) Decoder Unit.

Equation (1) suggests that if the set of parity bits inferred by the data bits is given to the input of the encoder, the original data would be extracted. So, encoding a nibble of data produces a nibble of parity, and encoding a nibble of parity reproduces the data nibble.

Based on the above, when a double error occurs in the data nibble or the parity nibble, we can extract the original data from the non-faulty nibble.

## V. Proposed Method

We assume that the scope of MBU errors caused by heavy particles in our applications is as follows:

- The probability of error generation in adjacent bits in both data register and its extended Hamming code register at the same time is negligible.
- Most of errors are generated in adjacent bits in data register or its extended Hamming code register not in both of them at the same time.

This technique employs information redundancy and combines it with narrow-width value concept. Having a $4*N$-bit data encoded with extended Hamming (8, 4) results a 100% overhead in information bits. We will present a decoding mechanism that first seeks whether only one of parity or data registers has been exposed to error or not, and extracts non-faulty register. Otherwise if it realizes both parts are faulty, it checks whether the corrected version that is obtained by corrector circuit is valid to export or not. As we have 2 copies from lower 32 bits of narrow-width values, we can perform mentioned process to each copy and find the corrected data among 2 obtained results.

In conventional method that employs Hamming SEC-DED code for $N$ blocks of data bits, when a double error is detected, the method will not try more to correct the faulty data and simply reports the error detection. The conventional method can correct errors if only single errors are detected in data bits or parity bits.

### A. Encoding

The encoding scheme is identical to that of a conventional method. Given a 64-bit data divided into 16 nibbles that are 16-way interleaved ($D$_nibble[$i$] is $\{D[i], D[i+16], D[i+32], D[i+48]\} \mid i = 0$ to 15), encoding is done in parallel by means of 16 sub-blocks of extended Hamming (8, 4) encoder (Fig. 4). Each sub-block is typically composed of six 2-input XOR gates (or four 3-input XOR gates). Produced parity bits from the encoder are stored at 16-way interleaved positions similarly, within other 64-bit register (Parity register).

In general, for a $4*N$-bit data word, that employs $N$-way interleaved extended Hamming (8, 4) code, the encoding process is denoted by (2) that is inferred from (1). The second line of (2) denotes invert relationship that based on it one can obtain data bits from parity bits. It also suggests that for a $4*N$-bit parallel encoder, one needs $4*N$ 3-input XOR gates (or $6*N$ 2-input XOR gates).

$$\begin{cases} P_i = D_{(3*N-i) \bmod (4*N)} \wedge D_{(4*N-i) \bmod (4*N)} \wedge D_{(5*N-i) \bmod (4*N)} \\ D_i = P_{(3*N-i) \bmod (4*N)} \wedge P_{(4*N-i) \bmod (4*N)} \wedge P_{(5*N-i) \bmod (4*N)} \end{cases} (i = 0 \ \ to \ \ (4 \times N) - 1) \qquad (2)$$

$N = 16$ in our case.

### B. Decoding

The decoding scheme is based on divide and conquer, i.e. it processes information provided by 16 sub-blocks of extended Hamming (8, 4) decoders to decide between which of three buses to export. These three buses are described in the following. The first bus comes directly from data register (Direct Data in Fig. 5). Parity bits received in the decoder are internally passed through an encoder logic to provide a second version of original data (Data from Parity in Fig. 5). The corrector circuit tries to correct the errors and provides third version of original data (Modified Data in Fig. 5).

Each data nibble from 64-bit data and its affiliated parity nibble are given to a syndrome bit producer unit. This unit is basically a circuit like one we have in the encoder, except that one extra XOR stage is required for comparing between parity bits and re-encoded data bits. Four generated syndrome bit signals are given to a bit address decoder. This unit provides eight signals, based on the pattern of syndrome bits. Each of these signals represents a single faulty bit within the twin nibbles. Within the same circuit a double bit error signal per each twin is also produced that indicates a double bit error has taken place somewhere within two nibbles.

All 64 addressing signals that locate faulty data bits, (together with direct data bus) provide the modified version of data by means of a stage of 2-input XOR gates as depicted in Fig. 5. Whenever an addressing signal of one data bit is asserted, XOR gate inside the corrector complements the affiliated data bit.

As mentioned earlier, Selector logic chooses between the three named buses by comparing error status information that is provided by three main error signals. These signals are described as following; A general signal that is OR of all 64 addressing signals is produced to state whether at least one single error is taken place in data register or not. Another signal that states occurrence of single errors in parity register is produced likewise by using other 64 addressing signals that associate with parity bits. A general signal that is OR of all 16 double error report signals is also produced to indicate that at least one double error occurred somewhere within registers.

If at most two bursts of errors, each with length equal or less than 16 bits occur in just one of the two registers, the decoder provides the selector circuit with valid error reports. If at least one single error is reported for one register and no single error is reported for the other register (regardless of double error reports), the selector chooses the register with no single error as non-faulty register. In other words, the selector assumes that the double errors are occurred within the register that contains single errors. If number of errors in a twin is odd and bigger than one, the decoder provides wrong error reports.

In case that at least a single error in data register and at least a single error in parity register are reported at the same time, and no double error is reported, the modified data from corrector circuit is assumed to contain valid data.

UE (Uncorrectable Error) signal is asserted when general error signals report:

- double error and no single error;
- double error and single errors in both data and parity registers;

The decoding scheme can tolerate long burst errors with width of up to 32 bits within one register, while it is susceptible to random multi errors that occur in both registers at the same time. To increase random errors correction ability of the decoder, we employed narrow-width value concept. This concept very well fits inside the circuit with a quite fine modification in the proposed logic. In one aspect, the 64 bit decoder is composed of two 32 bit decoders that work in parallel. For narrow-width values, the selector circuit checks two groups of general error signals. If errors in first group are not correctable, the decoded

value of second group is exported. This method corrects all 2 random errors in a narrow-width value and related parity register. Narrow-width values information should be protected with triple modular redundancy (TMR). Fig. 5 illustrates the decoder unit of presented method that employs narrow-width values to improve the fault tolerance characteristics of the microprocessor.

### C. Using of The Proposed Method in a Microprocessor

For using proposed method in a microprocessor we should perform error detection and correction in parallel with execute stage to prevent performance degradation. Operands of an instruction should be read in decode stage and analyzed with error detection and correction unit and simultaneously, the instruction execution unit performs its operation. There are 2 scenarios when error detection and correction unit finds errors in an operand:

- If unit can not correct the errors, an exception will be raised. Operating system should handle the detected errors.
- If unit can correct the errors, a stall command should be sent to all stages of microprocessor. Simultaneously with stalling all stages of microprocessor, error detection and correction unit should exchange the corrupted value of the input of instruction execution unit with corrected value. On the other hand, errors can be accumulated in register file if we don't scrub it. So, we should write the corrected value to the register file at the same time with correcting the input value of instruction execution unit. If both operands are corrupted and can be corrected, the unit should stall the stages of the microprocessor twice.

Fig. 6 shows the datapath and the pipeline stages of a microprocessor with additional units for the proposed error detection and correction method.

## VI. Evaluation of The Proposed Method

In this section we compare proposed method to TMR and various extended Hamming codes. In TMR technique we should triple the hardware and use a majority voter to find the correct output [16]. Fig. 7 shows a register file with two read ports and one write port that is protected with TMR technique.

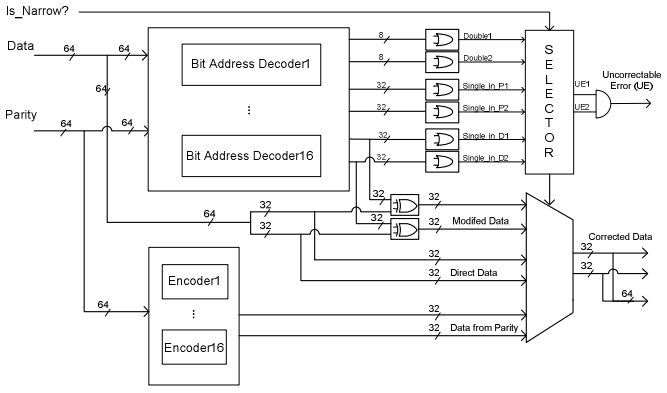


Figure 5. Decoder Unit of The Proposed Method.

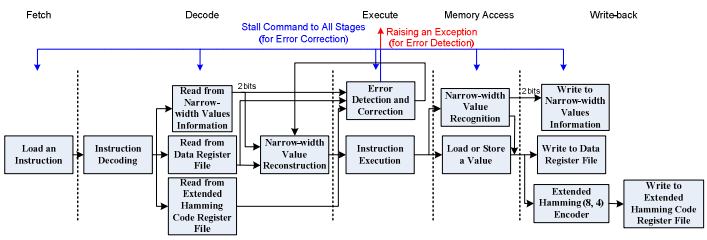


Figure 6. Datapath and The Pipeline Stages of a Microprocessor With Additional Units for Proposed Error Detection and Correction Method.

If the error probability of a single bit is $\varepsilon_e$, then the error probability of a word with $w$ bits that is protected using TMR is [18]:

$$P_{word-TMR} = 1 - (1 - 3\,{\varepsilon_e}^2(1 - \varepsilon_e) - {\varepsilon_e}^3)^w \qquad (3)$$

The error probability of a word with $w$ bits that is protected with extended Hamming $(n, k)$ code can be obtained using (4). The word with $w$ bits is divided into the blocks with $k$ bits that each block is protected with $(n - k)$ parity bits as described in the extended Hamming code section. Since detecting errors is not sufficient for many applications, we assume that the error probability for extended Hamming codes is the probability of not correcting errors. With this assumption the error probability of extended Hamming codes and Hamming codes are equal.

$$P_{word-ex-Ham} = 1 - \{(1 - \varepsilon_e)^n + n\varepsilon_e(1 - \varepsilon_e)^{(n-1)}\}^{\left(\frac{w}{k}\right)} \qquad (4)$$

The analysis of the proposed method using random fault injection simulation is shown in table 1. Since proposed method uses extended Hamming (8, 4) code, the results of fault injection simulation for the conventional method that uses extended Hamming (8, 4) code is shown in table 1 too. The obtained results for more than 4 random errors are not shown in this table. Equation (5) shows the error probability of the proposed method for normal-width values that is the probability of not correcting the errors in a word with $w$ bits that stores a normal-width value. In (5) $w$ should be multiple of 4.

$$\begin{cases} P_{word-normal} = \sum_{j=1}^{2w} (1 - C_j)\left(\frac{factorial(2w)}{factorial(2w-j)factorial(j)}\right){\varepsilon_e}^j(1 - \varepsilon_e)^{(2w-j)} \\ C_j = Correction\ Percentage\ for\ j\ random\ errors\ in\ normal - width\ values \end{cases} \qquad (5)$$

Equation (6) shows the error probability of the proposed method for narrow-width values that is the probability of not correcting the errors in a word with $w$ bits that stores a narrow-width value. In (6) $w$ should be multiple of 4.

$$\begin{cases} P_{word-narrow} = \sum_{j=1}^{2w} (1 - C_j)\left(\frac{factorial(2w)}{factorial(2w-j)factorial(j)}\right){\varepsilon_e}^j(1 - \varepsilon_e)^{(2w-j)} \\ C_j = Correction\ Percentage\ for\ j\ random\ errors\ in\ narrow - width\ values \end{cases} \qquad (6)$$

In (6) the error probability of 2-bit data that is mitigated with TMR and stores the narrow-width value information of the word is neglected.

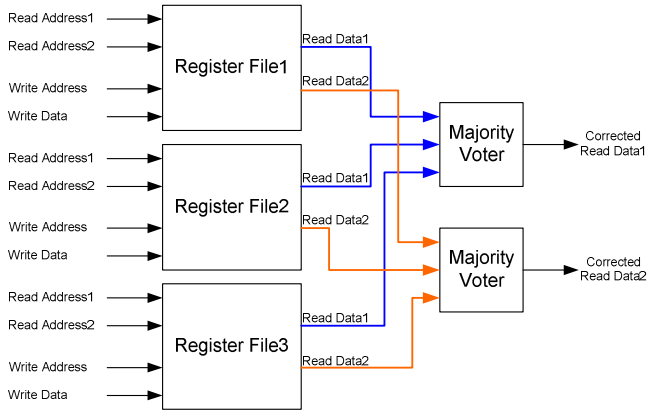


Figure 7. A Register File That Is Protected With TMR Technique.

TABLE I. ANALYSIS OF DIFFERENT ERROR DETECTION AND CORRECTION METHODS THAT USE EXTENDED HAMMING (8, 4) CODE WITH RANDOM ERRORS

| Number of Random Errors in Data and Extended Hamming Code registers | Conventional Method | | | Proposed Method For Normal-width Values | | | Proposed Method For Narrow-width Values | | |
|---|---|---|---|---|---|---|---|---|---|
| | *Correction* | *Detection* | *Wrong Correction* | *Correction* | *Detection* | *Wrong Correction* | *Correction* | *Detection* | *Wrong Correction* |
| 1 | 100% | 0% | 0% | 100% | 0% | 0% | 100% | 0% | 0% |
| 2 | 94.5% | 5.5% | 0% | 94.5% | 5.5% | 0% | 100% | 0% | 0% |
| 3 | 84% | 15.74% | 0.26% | 86.74% | 0% | 13.26% | 97% | 0% | 3% |
| 4 | 69.88% | 29.11% | 1.01% | 72.05% | 15.33% | 12.62% | 92.10% | 0.57% | 7.33% |

So the error probability of a word with $w$ bits that stores the generated values for register file in a benchmark and is mitigated with proposed method is:

$$\begin{cases} P_{word-proposed} = pct_{normal} \times P_{word-normal} + pct_{narrow} \times P_{word-narrow} \\ pct_{normal} = percentage\ of\ normal - width\ values\ in\ benchmark \\ pct_{narrow} = percentage\ of\ narrow - width\ values\ in\ benchmark \end{cases} \quad (7)$$

Fig. 8 shows the error probability curves of various protection techniques for a 64-bit word that stores generated values for register file in SPEC2000 benchmarks. The percentage of normal-width and narrow-width generated values for register file in SPEC2000 benchmarks is 15% and 85% respectively.

As depicted in Fig. 8 the error probability of the proposed method for a 64-bit word that stores the generated values for register file in SPEC2000 benchmarks is lower than the error probability of other techniques for the same word. The error probability of the proposed method for a benchmark can vary with respect to the percentage of generated narrow-width values for register file in that benchmark. Based on our experimental results, if the percentage of generated narrow-width values for register file in a benchmark is more than 60%, then the error probability of the proposed method for a word that stores the generated values for register file in that benchmark will be less than the error probability of TMR for the same word.

Table 2 shows the analysis of the conventional and the proposed method by simulating fault injection in adjacent bits. Results shown in this table are based on these assumptions that errors in adjacent bits can be random, and number of errors in $N$ adjacent bits varies from 1 up to $N$ errors. In fact in conventional methods that employ hamming SEC-DED codes for $N$ blocks of data bits (that are $N$-way interleaved), decoding algorithms suggest correcting burst errors with length less than $N$ bits. Presented scheme suggests correcting burst errors with length less than 2* $N$ (or 2 bursts of errors each with length less than $N$ bits) that take place within only data register or parity register. For presented scheme each block contains 4 data bits.

Fig. 9 shows the area overhead of a 64-bit register file that is protected with different methods with respect to the size of the register file. The results shown in Fig. 9 have been obtained with implementation on a Xilinx Virtex-4 FPGA (XC4VLX25). We have assumed that narrow-width values information is protected with TMR. If the number of words in a 64-bit register file is bigger than 64, then the area overhead of the register file that is protected with the proposed method will be less than the area overhead of the same register file that is protected with TMR technique.

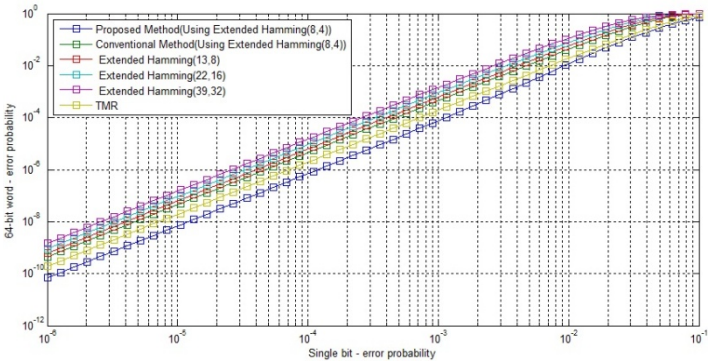


Figure 8. Error Probability Curves of Various Protection Techniques for a 64-bit Word That Stores The Generated Values for Register File in SPEC2000 Benchmarks.

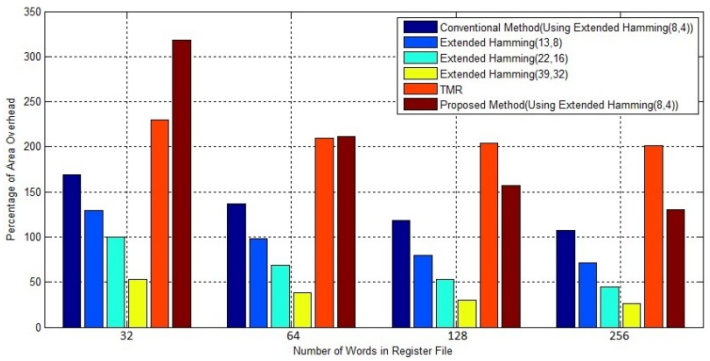


Figure 9. Area Overhead.

Table 3 shows the read and write latency overhead of different protection methods that is obtained with implementation on a Xilinx Virtex-4 FPGA (XC4VLX25). We have assumed that the error detection and correction of the Hamming codes and the proposed method is performed in parallel with execute stage.

Table 4 shows the propagation delay of the error detection and correction unit of the Hamming codes and the proposed method that is obtained with implementation on a Xilinx Virtex-4 FPGA (XC4VLX25). As shown in table 4, the propagation delay of the error detection and correction unit of the proposed method is 9.1ns. So if the propagation delay of execute stage of processor is bigger than 9.1ns, then no performance degradation will be incurred because of the propagation delay of the error detection and correction unit of the proposed method.

TABLE II. ANALYSIS OF DIFFERENT ERROR DETECTION AND CORRECTION METHODS THAT USE EXTENDED HAMMING (8, 4) CODE WITH ERRORS IN ADJACENT BITS

| Number of Adjacent Bits in Data or Extended Hamming Code Registers That Errors Can Occur Within Them | Conventional Method | | | Presented Method For Normal-width Values | | | Presented Method For Narrow-width Values | | |
|---|---|---|---|---|---|---|---|---|---|
| | *Correction* | *Detection* | *Wrong Correction* | *Correction* | *Detection* | *Wrong Correction* | *Correction* | *Detection* | *Wrong Correction* |
| 1-16 | 100% | 0% | 0% | 100% | 0% | 0% | 100% | 0% | 0% |
| 17-32 | 31% | 69% | 0% | 99% | 1% | 0% | 99.6% | 0.4% | 0% |

## VII. CONCLUSION

In this paper we proposed a new method using extended Hamming (8, 4) code to detect and correct soft errors in a 64-bit register file. The presented method can correct long burst errors. To increase the protection efficiency to random soft errors, the method employs narrow-width values. Narrow-width values can be represented by half number of bits of full data width. About 85% of generated values for the register file in SPEC2000 benchmarks are narrow-width values. The presented method has been analyzed using fault injection simulation. This method can correct more than 99% of errors that occur in adjacent 32 bits or 2 bursts of errors each with length less than 16 bits in data register or related extended Hamming code register (not in both of them at the same time). The error detection and correction unit of the proposed method performs its operation in parallel with execute stage to prevent performance degradation. Experimental results show that if in a benchmark the percentage of generated narrow-width values for register file is more than 60%, then the error probability of a word that stores the generated values for register file in that benchmark and is protected with proposed method will be less than the error probability of the same word that is mitigated using TMR or various extended Hamming codes. The implementation on a Xilinx Virtex-4 FPGA shows that the required area of a register file with 64-bit wide and more than 64-word entry that is protected using proposed technique is less than the required area of same register file that is protected using TMR.

TABLE III. READ AND WRITE LATENCY OVERHEAD

| Method | Read Latency Overhead | Write Latency Overhead |
|---|---|---|
| Proposed Method(Using Extended Hamming(8,4)) | 18% | 80% |
| TMR | 19% | 0% |
| Conventional Method(Using Extended Hamming(8,4)) | 0% | 80% |
| Extended Hamming(13,8) | 0% | 100% |
| Extended Hamming(22,16) | 0% | 150% |
| Extended Hamming(39,32) | 0% | 180% |

TABLE IV. PROPAGATION DELAY OF THE ERROR DETECTION AND CORRECTION UNIT

| Method | Propagation Delay of the Error Detection and Correction Unit |
|---|---|
| Proposed Method(Using Extended Hamming(8,4)) | 9.1ns |
| Conventional Method(Using Extended Hamming(8,4)) | 5.606ns |
| Extended Hamming(13,8) | 6.21ns |
| Extended Hamming(22,16) | 6.8ns |
| Extended Hamming(39,32) | 7.42ns |

## REFERENCES


[1] J.F.Ziegler, et al., "IBM experiments in soft fails in computer electronics (1978 – 1994)," *IBM Journal of Research and Development*, pp. 3 – 18, Volume 40, Number 1, January 1996.

[2] C. Weaver *et al.*, "Techniques to reduce the soft errors rate in a high performance microprocessor," in *Proc. of ISCA-31*, pp. 264- 275, 2004.

[3] P. Shivakumar *et al.*, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. International Conference on Dependable Systems and Networks*, pp. 389–398, June 2002.

[4] G.P. Saggese, N.J. Wang, Z.T. Kalbarczyk, S.J. Patel, and R.K. Iyer, "An Experimental Study of Soft Errors in Microprocessors", *IEEE Micro,* vol. 25, no. 6, pp. 30-39, Nov. 2005.

[5] G. Memik, M.T. Kandemir, and O. Ozturk "Increasing Register File Immunity to Transient Errors", *Proceedings of Design, Automation and Test in Europe (DATE'05),* Munich, Germany, vol. 1, pp. 586-591, Mar. 2005.

[6] J. Gaisler, "Evaluation of a 32-bit Microprocessor with Built-in Concurrent Error-Detection", *Proceedings of the 26th Annual International Symposium on Fault-Tolerant Computing (FTCS-27)*, Seattle, Washington, USA, pp. 42-46, Jun. 1997.

[7] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture", *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'02)*, Bethesda, Maryland, USA, pp. 409-415, Jun. 2002.

[8] Jie Hu; Shuai Wang; Ziavras, S.G., "On the Exploitation of Narrow-Width Values for Improving Register File Reliability," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.17, no.7, pp.953-963, July 2009.

[9] M. Lipasti, et.al., "Physical Register Inlining", *in Proc. Of ISCA, 2004*.

[10] D. Brooks, M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", *in Proc. of HPCA, 1999*.

[11] G. Loh, "Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth", *in MICRO-35*, 2002.

[12] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of Multibit Soft Error Events in Advanced SRAMs", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 519- 522, Dec. 2003.

[13] S. Satoh, Y. Tosaka, S.A. Wender, "Geometric Effect of Multiple-bit Soft Errors Induced by Cosmic-ray Neutrons on DRAMs", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 310- 312, Jun. 2000.

[14] A. Makihara, et al., "Analysis of Single-Ion Multiple-Bit Upset in High-Density DRAMS", *IEEE Trans. on Nuclear Science*, Vol. 47, No. 6, Dec. 2000.

[15] Y. Kawakami, et al., "Investigation of Soft Error Rate Including Multi-Bit Upsets in Advanced SRAM Using Neutron Irradiation Test and 3D Mixed-mode Device Simulation", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 945-948, Dec. 2004.

[16] W. PETERSON, "Error-correcting codes," 2nd ed., Cambridge : The MIT Press, 1980. 560p.

[17] Ricketts, A.J.; Madhu Mutyam; Vijaykrishnan, N.; Irwin, N.J.; , "Investigating Simple Low Latency Reliable Multiported Register Files," *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on* , vol., no., pp.375-382, 9-11 March 2007.

[18] Naseer, Riaz; Bhatti, Rashed Zafar; Draper, Jeff; , "Analysis of Soft Error Mitigation Techniques for Register Files in IBM Cu-08 90nm Technology," *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on* , vol.1, no., pp.515-519, 6-9 Aug. 2006.