Comparative Study Regarding two Implementations of an SEC-DED Code with FPGA Circuits

O. Novac, J. Sztrik, and C. Grava, Member, IEEE

Abstract—In this paper we compare two different implementation of an SEC-DED HSIAO code. This comparison is made to select which is the best method to achieve better memory, from reliability point of view. In this paper HSIAO code is used, because is efficient for single errors correction and double error detection that appear in cache memory hierarchy. Also, we have implemented the HSIAO code with FPGA Xilinx circuits for both implementations.

Index Terms—HSIAO code, SEC-DED code, cache, FPGA circuits.

I. INTRODUCTION

In current computer systems memory is the most sensitive part. When we write in memory or when we read the memory, errors may occur. This is why we decided to introduce a code for correcting errors that appear in our memory hierarchy. Cache memory is very sensitive in terms of reliability. In my previous work I have studied which are most suitable codes for error correction and error detection [1], [2], [3], [4], [13].

In this paper we will implement an SEC-DED (Single Error Correction and Double Error Detection) code to obtain a more reliable memory. Error detection methods normally provoke stopping the read/write operations with memory, before the erroneous data will be used in the system. Unfortunately, the correction is far more complicated. It requires first detecting the error, and then rebuilding the information based on redundant information, even during system operation. Depending on the process used for error correction or detection, we can repair many different types of errors that occur in the memory hierarchy, including those that appear in cache [5].

A simple method used for error detection relies on the usage of "parity" bits. This simple method can detect simple errors (change a single bit of the word), but not multiple errors because this method can’t correct the detected errors. Parity is a simple error detection method, which adds a single bit to each word (8 bits) of the memory module. This extra bit (the ninth bit), restrain in those 8 bits of each word is an odd or even number of 1 digits and is stored or transmitted together with the word. If we read or write the word, parity is recalculated and the result obtained is compared with the parity bit read or received. If the information does not match, then it means that there is an error either in transmission, or in memory. Multiple errors do not change the parity bit and therefore this type of errors can’t be detected through this method. Error correction methods are more advanced methods for detecting errors. These methods are more complex than parity methods, because with these methods we can correct errors as well [8].

For detection of multiple error and error correction of simple errors we can extend this simple parity method at block level. In this correction method, in addition to the parity bit of each word, we will use a parity bit at column level of word block. Because servers typically contain a huge capacity (GB of RAM’s) and runs 24 hours a day, the likelihood of errors in this RAM’s is relatively high and they require efficient methods of error correction (ECC - Error Correction Codes). In these cases, the memory must be protected with a very complex ECC code. This code can automatically correct any 1 bit error that appears in a 64-bit word. For this reason the memory stores each 64-bit word using a 72 bits code word (in which 8 bits are for ECC). The hardware checks at every memory access if the codeword is correct; if not, it automatically calculates the nearest codeword which has to be decoded afterwards. These operations are complicated; hence, a system with ECC memory operates with approximately 5% slower than one with parity memory [5], [6].

An important objective is to reduce the difference among memory and processor speed, so actual processors need to allocate almost all of their resources to cache [7], [8], [9].

To have a large and rapid memory we will use a memory hierarchy. In this hierarchy we have faster technologies with a...
At the bottom level of our hierarchy, we have most expensive, technology has a higher capacity for information storage. The cost-per-bit higher than slower technology, but the latest technology has a higher capacity for information storage. Figure 1 shows a hierarchy of such memories [5], [10], [11]. At the bottom level of our hierarchy, we have most expensive, slowest, and highest storage capacity memories.

II. APPLYING HSIAO CORRECTING CODE TO THE CACHE LEVEL OF HIERARCHY

A good method to increase the reliability of cache is the fault tolerant approach. Typically this is done by introducing redundant information. An efficient and used method for fault tolerant design is Hamming code. This error correcting code is based on parity bit generation. The HSIAO code is more efficient and is a modified version of Hamming code. The resulted design will be after that implemented with two types of FPGA circuits and the two implementations will be compared.

In computer systems design we can apply multiple error correction codes to cache level of the memory hierarchy. Single Error Correcting and Double Error Detection (SEC-DED) codes are the most used codes in high speed memories and are used to obtain better reliability of the hierarchy [11].

We choose for our cache implementation the HSIAO code. HSIAO is a correcting code used due to its ability to recover from multiple errors. In addition, it is stated that the H has a unit matrix array (I) on the first six rows and six columns and this matrix and is built of columns with an odd number of 1 bit. In our case, the number of 1 bit in each column is equal to 3 or 5. We also notice that in our matrix H we do not have two identical columns [12].

In this particular implementation of cache we have choose a (22,16,6) particular HSIAO code. So we have six control bits (named k), sixteen useful bits (named u) and twenty two code bits. To correct a single error (of only one bit) we must satisfy the following relation: u+k+1<2k. In a classic correcting code we need only 5 control bits, but we will add an extra bit and in final will have 6 control bits. With 6 control bits we can detect double bit error detection. Bellow, we present our matrix H. This matrix is the parity check matrix for our HSIAO code.

In matrix H, a codeword has the following form:

\[ u=(c_0c_1c_2c_3c_4c_5u_0u_1u_2u_3u_4u_5u_6u_7u_8u_9u_{10}u_{11}u_{12}u_{13}u_{14}u_{15}) \] (1)

HSIAO code has data bits from position 7 to position 22 and parities in the other six positions (from 1 to 6).

Bellow in (2), we present the control bits, which are calculated with the following parity equations:

- \[ c_0 = u_{12} \oplus u_{13} \oplus u_{14} \oplus u_{15} \oplus u_{16} \oplus u_{17} \oplus u_{18} \oplus u_{19} \oplus u_{20} \oplus u_{21} \oplus u_{22} \] (2)

In (3), we present the syndrome equations that will decode a received vector:

\[ s_0 = u_{10} \oplus u_{11} \oplus u_{12} \oplus u_{13} \oplus u_{14} \oplus u_{15} \oplus u_{16} \oplus u_{17} \oplus u_{18} \oplus u_{19} \oplus u_{20} \oplus u_{21} \oplus u_{22} \] (3)

For implementation of equation (4), we use nine XOR gates with two inputs as is presented in figure 2. All the other control bits are generated in the same way. This control bits (from c_0 to c_5) are compared, using five XOR gates, with the control bits in each column is equal to 3 or 5. We also notice that in our matrix H we do not have two identical columns [12].

In this particular implementation of cache we have choose a (22,16,6) particular HSIAO code. So we have six control bits (named k), sixteen useful bits (named u) and twenty two code bits. To correct a single error (of only one bit) we must satisfy the following relation: u+k+1<2k. In a classic correcting code we need only 5 control bits, but we will add an extra bit and in final will have 6 control bits. With 6 control bits we can detect double bit error detection. Bellow, we present our matrix H. This matrix is the parity check matrix for our HSIAO code.

\[ H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \]
In figure 2 we present the scheme used for correction of a single error that can appear in the cache. In this scheme an (22,16,6) HSIAO code is implemented.

Data bits are corrected using sixteen XOR gates, as is illustrated in figure 2. In this figure is presented the implementation of (22,16,6) HSIAO code.

III. IMPLEMENTATION OF HSIAO CODE USING SPARTAN XS05 XILINX CIRCUITS TO THE CACHE MEMORY

We have chosen to design our HSIAO (22,16,6) code, first with Spartan XS05 FPGA Xilinx circuits. The (CLB) Configurable Logic Blocks and (PSM) Programmable Switch Matrices with the surrounding I/O pins, make up the internal structure of an FPGA circuit. This FPGA is programmable and includes two configurable elements: Input Output Blocks (IOB) and CLB. Figure 3 presents such a CLB circuit [1],[11],[13].

Fig. 3. CLB circuit from FPGA Spartan XS05 family

The role of IOB is to create an interface between internal signals and the outside of circuit using pins. CLB has functional elements that implement the designed logical structure. The logical function performed by CLB is implemented by static configuration memory [13].

This HSIAO code is used for single error correction but our scheme can also detect double error in the cache. This HSIAO code is implemented using FPGA programmable Spartan XS05 circuits, aspect illustrated in figure 4.

Fig. 4. Implementation with FPGA from Spartan XS05 family

Because our Hsiao code has a minimum number of 1’s in the matrix, we have optimal hardware and speed of the encoding / decoding circuit. From Xilinx Map Report file we can observe that from a total of 100 CLB circuits only 50 have been used. In conclusion, only half of CLB circuits are used.

IV. IMPLEMENTATION OF HSIAO CODE USING VERTEX 6 XILINX CIRCUITS TO THE CACHE MEMORY

We have chosen to implement the same Hsiao (22,16,6) code, with VERTEX 6 FPGA Xilinx circuits. Below, in figure 5 we present again a CLB circuit, but this time from VERTEX 6 family.

Fig. 5. CLB circuit from FPGA VERTEX 6 family

This time, we implement our HSIAO code with this VERTEX 6 (XC6VHX255T-2FFG1155C) FPGA Xilinx circuits, because these circuits are rapid and efficient. The programmable internal structure of a FPGA includes also, as in previous case, two configurable elements: CLB’s and IOB’s. In figure 6 we present the implementation of HSIAO code using VERTEX 6 Xilinx Circuits.
We analyze the second Map Report file and we can observe that from a total of 440 CLB only 40 circuits have been used. In conclusion, only 9% of the total bonded IOB’s and only 53 of Slice LUTs out of 158,400, meaning 1% of the total Number of Slice LUTs have been used. From this point of view this implementation is better because we have a smaller area occupied by IOB circuits than in first implementation with Spartan XS05 circuits. A second point of view is the speed. With the second implementation we will have a higher speed of because FPGA circuits from Vertex 6 family are faster circuits.

V. CONCLUSION

In this paper we have compared two different implementation of an HSIAO code. We can increase dependability through fault tolerance using an HSIAO code and this leads to lower costs and lower memory chip dimensions. The reason is that, through this method we can corrects single errors within the chip. Other advantages of this method are computing speed and increased storage capacity, with little investment in our configuration.

In the first implementation of HSIAO code, we have used FPGA from Spartan XS05 family. Our scheme was implemented in this case with a number of 50% of the total CLB circuits. In the second implementation we have implemented the HSIAO code using FPGA from VERTEX 6 family. In this second case, we can observe that from a total of 440 CLB only 40 circuits have been used, and this means that only 9% of the total bonded Input Output Blocks and only 53 of Slice LUTs out of 158,400, meaning 1% of the total Number of Slice LUTs have been used. The second implementation is better because we have a smaller area occupied by IOB circuits than in first implementation with Spartan XS05 circuits. Also in the second implementation we have a higher speed of because FPGA circuits from Vertex 6 family are faster circuits. As a final conclusion the second implementation is better than the first, because of the two reasons mentioned above.

ACKNOWLEDGMENT

This work was supported by the TAMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund. The research was also supported by the European Union with the help of the GOP-1.3.2-08-2009-0007 program.

REFERENCES