Microelectronics Reliability 54 (2014) 2645–2648

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

Research note

Optimized parallel decoding of difference set codes for high speed memories

Mustafa Demirci [a], Pedro Reviriego [b,*], Juan Antonio Maestro [b]

[a] Aselsan, Mehmet Akif Ersoy Mahallesi 296, 16, 06370 Yenimahalle, Ankara, Turkey
[b] Universidad Antonio de Nebrija, C/Pirineos, 55 E-28040 Madrid, Spain

ARTICLE INFO

Article history:
Received 12 March 2014
Received in revised form 25 May 2014
Accepted 30 June 2014
Available online 27 July 2014

Keywords:
Difference-set codes
Error correction codes
Majority logic decoding
Memory

ABSTRACT

The interest in using advanced Error Correction Codes (ECCs) to protect memories and caches is growing. This is because as process technology downscales, errors are more frequent and also tend to affect multiple bits. For SRAM memories and caches, latency is a limiting factor and ECCs have to provide low decoding times that can in most cases be only achieved with the use of a parallel decoder. One important issue with parallel decoders is that they typically require large circuit area to be implemented. One type of ECCs that has been explored for memory protection is Difference Set (DS) codes. In this research note, an optimized parallel decoding scheme for DS codes is presented and evaluated. The results show that the circuit area and the decoding delay are reduced compared to a traditional implementation. In addition, the new scheme enables a reduction in the number of parity check bits thus reducing the memory size.

1. Introduction

Error Correction Codes (ECCs) are an attractive option to protect memories and caches from errors. This is due to a number of reasons. For example, as technology scales the percentage of radiation induced soft errors that affect multiple bits increases [1]. Another example is that in order to reduce power consumption, aggressive voltage reductions are proposed. These reductions increase the error rates and therefore advanced ECCs are needed to make sure that those errors do not compromise the integrity of the data stored in the cache or memory [2].

One key requirement for caches and SRAM memories is speed. Therefore, when used, advanced ECCs should minimize the impact on delay. The most complex task of an ECC is in most cases the decoding phase [3]. For many codes, decoding can be done serially or in parallel [4]. The serial decoder is typically much simpler and slower as it requires multiple cycles while the parallel decoder is faster but requires much more hardware resources. In the case of caches or high speed memories a parallel decoder can be an interesting option to meet the delay requirements. As an example, a parallel decoder for Double Error Correction (DEC) Bose–Chaudhuri–Hocquenghem (BCH) codes was proposed in [5]. The results show that low delay can be achieved at the cost of increased circuit area. Parallel decoders have also been explored for Orthogonal Latin Squares (OLS) codes to implement low power caches [6]. These codes are one-step majority logic decodable which enables fast and parallel decoding [4]. However, OLS codes require more parity check bits than other codes that can correct the same number of errors [7]. Other one-step majority logic decodable codes that have been explored for memory protection are a class of Euclidean Geometry (EG) codes. The implementation of a parallel decoder for EG codes was studied in [8] in the context of nano-scale memories and some optimizations were presented in [9]. Doubly Transitive Invariant (DTI) codes have also been recently considered to protect memories [3].

Difference Set (DS) codes are also one-step majority logic decodable and its use for memory protection has also been studied for example in [10]. Parallel decoders can also be directly implemented for DS codes. However, the properties and parameters of the DS codes enable optimizations of the parallel decoder implementation. In this research note, an optimized parallel decoding scheme for DS codes is presented and evaluated for FPGA implementations. The results show that significant reductions in decoder area and delay can be obtained. The scheme also provides a small reduction in the number of parity check bits. Therefore the proposed scheme can be useful for practical implementations of DS codes in high speed memories and caches.

* Corresponding author. Tel.: +34 914521100; fax: +34 914521110.
E-mail addresses: mdemirci@aselsan.com.tr (M. Demirci), previrie@nebrija.es (P. Reviriego), jmaestro@nebrija.es (J.A. Maestro).

http://dx.doi.org/10.1016/j.microrel.2014.06.017
0026-2714/

The rest of this research note is structured as follows, Section 2 presents a short overview of DS codes focusing on the decoder implementation and compares the DS codes with other codes that are also one step majority logic decodable. Then the proposed parallel decoder optimization is discussed in Section 3 and evaluated for FPGA implementations in Section 4. Finally the conclusions are summarized in Section 5.

2. Difference set codes

Difference Set (DS) codes are based on the concept of perfect difference set and have a complex mathematical construction [4]. They were proposed by Rudolph [11] and Weldon [12] and can be decoded with one step majority logic. Therefore, the decoder is easily implemented in parallel. The parameters of the DS codes for data block sizes smaller than 1 K bits are shown in Table 1. where k refers to the number of data bits and n to the total number of bits after encoding. It can be observed that the choices of block sizes and error correction capabilities are limited. This is the main disadvantage of DS codes. Another important point is that the block sizes are not a power of two. This however can be easily achieved by shortening the codes to the desired block length [4].

The parameters of other majority logic decodable error correction codes that are similar in terms of data block size and error correction capabilities to DS codes are given in Table 2. The optimized DS code that will be introduced in the next section is also included in the Table. From the table, it can be observed that DS codes have larger code rates (k/n) than EG–LDPC (Euclidian Geometry–LDPC), DTI (Doubly Transitive Invariant) and especially Orthogonal Latin Square (OLS) codes. For OLS codes, the number of parity check bits is equal to the number of data bits. Therefore, OLS codes require a much larger memory overhead for data widths greater than 16 bits. The larger code rates make DS codes attractive in terms of memory overhead.

As mentioned before, DS codes are one step majority logic decodable. This means that each bit can be decoded by taking the majority vote of J parity equations. A serial implementation of the decoder for the (21, 11) DS code is shown in Fig. 1. The codeword is placed on a shift register and at each clock cycle the contents are shifted to the right and the bit in position C_{20} is decoded by taking the majority vote of equations w_0 to w_4. The Majority Logic (ML) equations are such that any bit except the bit being decoded participates in exactly one ML equation. The bit being decoded participates in all J equations. This ensures that the bit will be decoded correctly when the number of errors is equal or smaller than $J/2$. For example, when a bit is in error and there are other $l-1$ errors, the worst case for the majority vote will be $J-l+1$ which is a majority when $l \leqslant J/2$. Conversely if a bit is correct and there are l errors the bit will not be miscorrected when $l \leqslant J/2$. Therefore as long as the number of errors is equal to or smaller than $J/2$ the code can correct the errors.

As the decoding for each bit is independent, a parallel decoder can be implemented in a direct way by replicating the majority voter and ML equations for each bit. In fact, since the ML equations are obtained by shifting, many of them appear in different bits and in total only n ML equations are needed for a parallel decoder as noted in [9] for EG codes. For example, for the parallel decoder version of the serial decoder shown in Fig. 1, the number of ML equations would be 21, and also 21 ML voters and 21 correction gates are needed. In general, for a DS code the direct parallel implementation of the decoder requires n ML equations, n J-inputs majority voters and n correction gates.

Table 1
Difference-Set codes parameters.

n	k	Correctable errors	Majority logic equations (J)
21	11	2	5
73	45	4	9
273	191	8	17

3. Optimized parallel decoders

For DS codes, the number of ML equations is always odd [4]. Therefore the codes can correct exactly $(J-1)/2$ errors. For example, for the (21, 11) code $J = 5$ and it can correct two errors. This is different from EG codes where the number of ML equations is even [8]. When a serial decoder is implemented, the odd number of ML equations can be used to detect some errors that cannot be corrected [13]. For example, for the (21, 11) code, the majority voter can be changed to a voter that requires majority of four, this ensures that three errors do not cause miss-corrections. Then the uncorrected errors can be detected by performing some additional iterations in the serial decoder. For a parallel decoder, delay is the critical parameter and performing additional error detection would increase it. Compared to serial decoder, that corrects t errors and detects $t+1$ errors, there is a trade-off between delay and error detection. Parallel decoder needs n times less clock cycles than serial decoder and corrects t errors, but cannot detect $t+1$ errors as original serial decoder does. Therefore, instead of enhancing error detection, the odd number of MLD equations can be used to simplify the parallel decoder. This is explained in the following.

A possible modification is to remove one entry from all the majority voters so that the number of majority logic equations for each bit is effectively reduced to $J-1$. For example, for the (21, 11) code four equations would be used so that still two errors can be corrected. This simplifies the majority voters (as they have less inputs) and also enables reductions in the number of overall equations and parity check bits. To explain in detail the modification, an example will be used. Consider again the (21, 11) DS code whose serial decoder is shown in Fig. 1. Assuming a systematic encoder, the last bits (C_{11} to C_{20}) will be parity check bits. To reduce the number of inputs to the majority voters, one option is to remove the equations used to decode C_{20} (w_0 to w_4 in Fig. 1). This will remove J equations. Since each bit participates in exactly one of those equations except for bit C_{20} which participates in all, the reduction in the number of inputs to the voter is achieved. In addition, parity check bit C_{20} is no longer needed as the equations in which it participated have been removed. This means that bit C_{20} can also be removed. To illustrate the scheme, the parallel decoding for bit C_0 is shown in Figs. 2 and 3 for a traditional decoder and the modified decoder respectively. The procedure described can be applied to a general DS code. In that case the number of inputs to the voter is reduced by one, the number of equations by J and the number of parity check bits by one. In the following, the modification will be named Reduced Majority Logic (RML).

The amount of hardware needed for the direct and optimized parallel decoders is summarized in Table 3. It can be observed that the proposed RML scheme reduces the implementation cost. The reductions compared to a direct implementation will be larger when n is small. For example, for the (21, 11) code one parity check bit is a 10% reduction and $J = 5$ equations is more than a 20% reduction. On the other hand, for the (273, 191) code, one parity check bit is less than a 2% reduction and $J = 33$ equations is slightly more than a 3% reduction. Therefore, the benefits of the proposed scheme are expected to be larger for smaller block sizes. In the next section a detailed evaluation of the proposed parallel decoders is presented based on their HDL implementations and mapping to an FPGA. The price paid to obtain this reduction is that the proposed decoders cannot detect $t+1$ errors as serial decoders [13].