A Joint Source/Channel Approach to Strengthen Embedded Programmable Devices against Flash Memory Errors

A Joint Source/Channel Approach to Strengthen Embedded Programmable Devices against Flash Memory Errors

Maurizio Martina, *Member, IEEE,* Carlo Condo, Guido Masera, *Senior Member, IEEE,* Maurizio Zamboni

***Abstract*—Reconfigurable embedded systems can take advantage of programmable devices, such as microprocessors and FPGAs, to achieve high performance and flexibility. Support to flexibility often comes at the expense of large amounts of non volatile memories. Unfortunately, non-volatile memories, such as multi-level-cell (MLC) NAND flash, exhibit a high raw bit error rate that is mitigated by employing different techniques, including error correcting codes. Recent results show that low-density-parity-check (LDPC) codes are good candidates to improve the reliability of MLC NAND flash memories especially when page size increases. This work proposes to use a joint source/channel approach, based on a modified arithmetic code and LDPC codes, to achieve both data compression and improved system reliability. The proposed technique is then applied to the configuration data of FPGAs and experimental results show the superior performance of the proposed system with respect to state of the art. Indeed, the proposed system can achieve bit-error-rates as low as about 10^{-8} for cell-to-cell coupling strength factors well higher than 1.0.**

***Index Terms*—arithmetic coding, LDPC coding, flash memories, FPGA**

I. Introduction

Modern embedded systems can take advantage of the reconfigurability offered by programmable devices, such as microprocessors and Field-Programmable-Gate-Arrays (FPGAs). Unfortunately, the amount of configuration data increases with both the size of programmable resources and the number of possible configurations, requiring a large non-volatile memory, such as flash memory, to store them. Besides, very high density Multi-Level-Cell (MLC) NAND-flash memories, implemented with highly scaled technologies, have a reduced noise margin and a high raw Bit-Error-Rate (BER) [1]. This unreliability is related to threshold voltage susceptibility, that, according to [2], is mainly caused by program disturb, read disturb and retention time limit. Different strategies are employed to face these phenomena, e.g. [3], [4], where Low-Density-Parity-Check (LDPC) codes and Data-Pattern-Aware (DPA) error prevention techniques are exploited to face cell-to-cell interference and vulnerable threshold voltage levels respectively. In particular, DPA error prevention techniques could be used to remap the output of the LDPC encoder to jointly improve system reliability. A further step to improve the reliability is reported in [5], where the authors increase the error correction capability of the LDPC code by using data compression. However, experiments shown in [5] rely on a negligible compression ratio (less than 5%) where the residual BER after LDPC decoding is nearly the same as the BER on decompressed data. On the contrary, when high compression ratios are employed, a small residual BER on compressed data leads to high BER on decompressed data. The novel contribution of this work is to go one step beyond the solution proposed in [5] and to propose a technique that improves the BER performance even when higher compression ratios are applied. Inspired by the idea developed in [6] for image transmission over wireless channels, in this work we propose a joint source/channel approach to combat MLC NAND-flash errors, induced by cell-to-cell interference, and to apply it to the compression of FPGA configuration data. The proposed system exploits a modified arithmetic code (MAC) not only to achieve high data compression but also to improve the BER performance of LDPC codes. Experimental results show the superior performance of the proposed system with respect to state of the art techniques based on LDPC codes. To the best of our knowledge this is the first work that tries to apply joint source-channel techniques to concurrently compress and protect against errors data to be stored in a flash memory.

II. Background and proposed technique

Efficient compression with arithmetic codes (ACs) is obtained when input symbols are mutually unrelated. Unfortunately, this is not always the case and a reversible decorrelation algorithm (DA) could be required. As an example, in [7], [8] AC is used to compress the bitstream of Xilinx Virtex and Virtex II pro FPGAs. In [7] a DA tailored on the structure of the bitstream is employed, namely the DA exploits the presence of different regions (e.g. Configurable Logic Blocks, I/O Blocks and Block RAMs), to find for each region the most probable bit-pattern. Each pattern is then applied to the proper region of the bitstream, that is performing a bitwise xor operation. Then data are coded by the AC. A similar approach with a proper DA can be used for other devices as well.

In general AC-based techniques can be described as follows. Let $\mathbf{x} = \{x_0, x_1, \ldots, x_{K-1}\}$ be the binary array representing the data to be stored in the non-volatile memory (made of K bits) and $\mathbf{y} = \{y_0, y_1, \ldots, y_{K-1}\}$ the result obtained by applying an optional DA to achieve a stream of bits where the occurrence of 0 is very high ($P_0 \geq 0.9$) with respect to the occurrence of 1 ($P_1 = 1 - P_0$). The AC maps $\mathbf{y}$ onto the sequence $\mathbf{z} = \{z_0, z_1, \ldots, z_{K'-1}\}$, where $K' < K$ is the length of the compressed sequence $\mathbf{z}$. The compressed sequence $\mathbf{z}$ is obtained as the result of a recursive process and represents the probability interval of $\mathbf{y}$. The AC initializes the

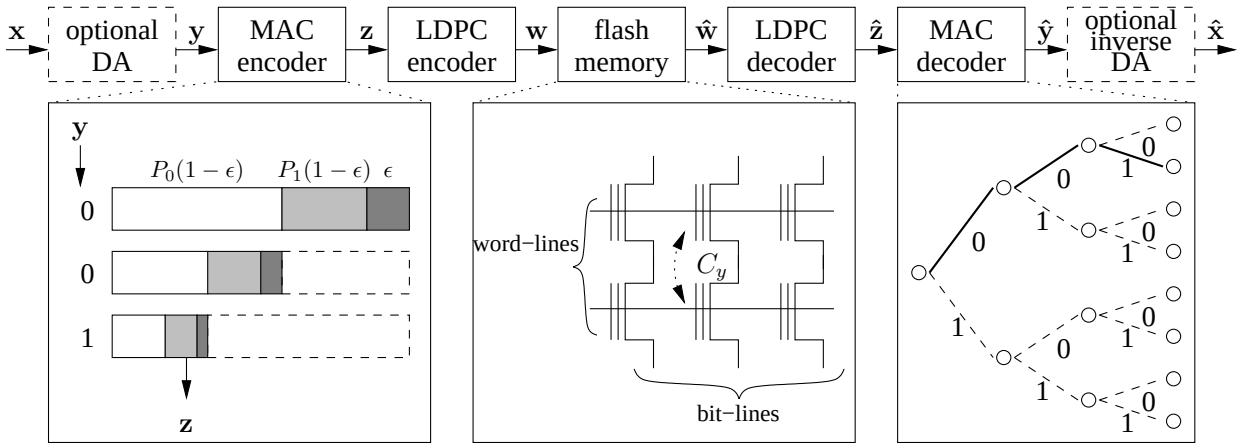

Figure 1. General block scheme of a Modified-Arithmetic-Coding-based (MAC-based) system for CD compression/error-correction with optional Decorrelation Algorithm (DA) and Low-Density-Parity-Check (LDPC) codes. Details: i) MAC encoding process and probability interval update; ii) flash memory structure and capacitance-coupling (C_y); iii) tree representation of the MAC decoding process.

probability interval as $[0, 1)$ and for each bit to be encoded selects the corresponding probability interval portion, P_0 for 0 or P_1 for 1. After K iterations an interval $I(\mathbf{y})$, whose width corresponds to the input sequence probability, is obtained and encoded by means of the shortest binary sequence belonging to it. This sequence represents the value $v(\mathbf{z}) \in I(\mathbf{y})$. On the other hand, the AC decoder progressively selects the intervals represented by $v(\mathbf{z})$ and obtains $\mathbf{y}$.

In this work we propose to use the MAC [6], namely to strengthen AC by using a ternary alphabet in the encoding. The basic idea of the MAC is to have a forbidden symbol μ, which is never encoded and whose probability is fixed to an arbitrary value $P_\mu = \epsilon$. The forbidden symbol perturbs the width of the intervals corresponding to 0 and 1 leading to $I_0 = P_0(1-\epsilon)$ and $I_1 = P_1(1-\epsilon)$ (see the left part of Fig. 1) and providing a form of coding redundancy per encoded bit

$$\rho_\mu = -\log_2(1-\epsilon). \tag{1}$$

At the decoder side, the presence of μ can be used for error detection; if the MAC decoder detects the forbidden symbol, then the compressed data contain errors. Let $\hat{\mathbf{z}}$ be the received compressed data containing errors; the coding redundancy associated to the forbidden symbol can be used by the decoder to select the best estimate of the encoded sequence, received from the error prone flash memory. Thus, when errors occur, the decoder should find the path that does not contain the forbidden symbol in the tree of all the possible decoded sequences (see the right part of Fig. 1), namely the decoder behaves as a Maximum-Likelihood (ML) decoder. In other words, let $\tilde{\mathbf{y}}$ be a generic K bit sequence, the decoder selects $\hat{\mathbf{y}}$ among all possible $\tilde{\mathbf{y}}$ sequences such that $\hat{m} = P(\hat{\mathbf{z}}|\hat{\mathbf{y}}) \geq P(\hat{\mathbf{z}}|\tilde{\mathbf{y}}) = \tilde{m}$, where $\hat{\mathbf{z}}$ is the received stream of coded bits, $\tilde{\mathbf{z}}$ is the arithmetic coded version of $\tilde{\mathbf{y}}$ and contains $\tilde{K}' = K'$ bits. Since AC is a reversible process, $\tilde{\mathbf{y}}$ can be replaced by $\tilde{\mathbf{z}}$ and we can rewrite the problem in logarithmic form as

$$\ln[\tilde{m}] = \sum_{j=0}^{K'-1} \ln[P(\hat{z}_j|\tilde{z}_j)], \tag{2}$$

where $P(\hat{z}_j|\tilde{z}_j)$ is the transition probability of each bit of $\hat{\mathbf{z}}$. Finally, depending on the optional DA used at the encoder side, the decoded sequence $\hat{\mathbf{y}}$ is used to obtain $\hat{\mathbf{x}}$, the best estimate of $\mathbf{x}$. Unfortunately, the complexity of exploring the tree of all possible decoded sequences is prohibitive and sub-optimal search strategies, such as the M-algorithm, must be employed [6]. Indeed, the M-algorithm is a breadth-first technique to explore the tree that limits the search space to M paths at each depth in the tree. The ML decoder guides the tree exploration and, for each node, MAC decoding operations shown in Algorithm 1 are computed, where E_i is the left endpoint of interval I_i and endpoints initialization is $E_0 = 0$, $E_1 = P_0(1-\epsilon)$ and $E_2 = (1-\epsilon)$. In particular, lines from 1 to 7 correspond to the MAC decoding, namely to output '0' ('1') or to detect the forbidden symbol depending on the interval where $v(\hat{\mathbf{z}})$ lays in. Lines from 8 to 22 correspond to the interval renormalization, namely to keep limited the precision to represent the intervals. Finally, lines from 23 to 24 update E_i values. As shown in Fig. 1 the proposed system includes an LDPC code encoder and the corresponding decoder as well. Namely, the AC compressed sequence $\mathbf{z}$ is coded by an LDPC encoder that produces $\mathbf{w} = \{w_0, w_1, \ldots, w_{N-1}\}$ with $N > K'$. If errors occur then the sequence read from the flash memory is $\hat{\mathbf{w}}$ and the LDPC decoder produces $\hat{\mathbf{z}}$, that is the input of the MAC decoder.

III. Experimental setup and results

A. System parameter values

Let R_S, R_C and R_M be the code-rate of the whole system, the LDPC code and the MAC respectively, where

$$R_S = R_C \cdot R_M \qquad R_M = \frac{1}{H + \rho_\mu} \tag{3}$$

and H is the entropy of $\mathbf{y}$ [6]. Then, combining (1) and (3) we obtain

$$\epsilon = 1 - 2^{-(R_C/R_S - H)}. \tag{4}$$

As a possible application of the proposed technique, this work targets configuration data of FPGAs. Thus, we analyzed

several configuration data for Xilinx FPGAs, including the ones in [7], [8], and we observed that $0.93 \leq P_0 \leq 0.95$, corresponding to $0.29 \leq H \leq 0.37$. Moreover, since in [5] a 2 bit/cell NAND flash with 4 kB page ($K = 32768$) and a rate-15/16 LDPC code are considered, we fixed $R_S = 15/16$ and $N = 34952$ in the following tests to have a fair comparison with previous works. As shown in (4) the value of ϵ depends on R_C and vice-versa. It is worth noting that the minimum possible R_C is achieved when $\epsilon = 0$, that is $\rho_\mu = 0$. Thus, from (3) we infer that $R_C \geq R_S \cdot H$; when $P_0 = 0.95$ this leads to about $R_C \geq 1/4$. The opposite case is $R_C = 1$ (no LDPC code is used) that in (4) gives $\epsilon \approx 0.42$ when $P_0 = 0.95$. As a consequence, given an LDPC code $1/4 \leq R_C \leq 1$ the MAC is configured to obtain $R_S = 15/16$. However, fixing R_S limits the flexibility of the proposed system and the maximum compression ratio (K'/K) is about 11%. The LDPC codes employed in this paper are regular ones with $N = 34952$ and the column weight of $\mathbf{H}$ is 4 as in [3], [5]. The decoding algorithm is the layered normalized-min-sum with $\sigma = 0.75$ and performs at maximum 30 iterations. The data are represented in the form of Logarithmic-Likelihood-Ratios (LLRs) as 2's complement values on 8 bits and, when LDPC decoding is finished, are used to compute $\ln[P(\hat{z}_j|\tilde{z}_j)]$, that is required in (2), resorting to the Max-Log approximation routinely employed in turbo and LDPC code decoding:

$$\ln[P(\hat{z}_j|\tilde{z}_j = 1)] \approx \begin{cases} 0 & \text{if } \lambda[\hat{z}_j] \geq 0 \\ \lambda[\hat{z}_j] & \text{else} \end{cases}, \quad (5)$$

$$\ln[P(\hat{z}_j|\tilde{z}_j = 0)] \approx \begin{cases} -\lambda[\hat{z}_j] & \text{if } \lambda[\hat{z}_j] \geq 0 \\ 0 & \text{else} \end{cases}. \quad (6)$$

The maximum number of paths explored with the M-algorithm has been set to 1024.

The MLC NAND flash memory has been modeled as described in [3] for all-bit-line structures (shown in the middle part of Fig. 1). Each cell of an MLC NAND flash can be programmed to one of the possible L levels with a threshold voltage $V_t^{(k)}$ with $k = 0, \ldots, L-1$. The statistical distribution of $V_t^{(k)}$ changes depending on the considered level. When a cell is erased the threshold voltage $V_t^{(0)}$ can be modeled as a Gaussian random variable with μ_e and σ_e as mean and standard deviation respectively. On the contrary, a cell programmed at level k ($k \neq 0$) has a threshold voltage distribution that is uniform in the range $[V_p^{(k)}, V_p^{(k)} + \Delta V_{pp}]$, where $V_p^{(k)}$ is the verify voltage for level k and ΔV_{pp} is the incremental program voltage step.

Unfortunately, cell-to-cell interference caused by parasitic capacitance-coupling effects (see the middle part of Fig. 1) causes a distortion in the threshold voltage distribution of programmed cells. As argued in [3] this effect can be taken into account considering vertical capacitance-coupling among different word-lines only (C_y), and introducing i) the vertical coupling ratio $\gamma_y = C_y/C_{Tot}$, where C_{Tot} is the total capacitance of the victim cell, ii) the cell-to-cell coupling strength factor s to obtain $\gamma'_y = \gamma_y \cdot s$. Thus, the threshold voltage of a cell is perturbed when one vertical neighboring cell is programmed. Programming an erased cell to level k generates a threshold voltage shift $\Delta V_t^{(k)} = V_t^{(k)} - V_t^{(0)}$ and induces on a victim cell a cell-to-cell interference $\gamma'_y \cdot \Delta V_t^{(k)}$. Assuming $V_t^{(k)}$ and $V_t^{(0)}$ as independent random variables, and the probability that a cell is programmed to a certain level is uniform, the distribution of the total cell-to-cell interference can be derived [3] and used to compute $\lambda[\hat{w}_i]$, the LLR of the i-th bit stored in a cell. As in [3], [5] we target a 2 bit/cell NAND flash with the standard gray code mapping, namely 11, 10, 00 and 01 correspond to levels 0, 1, 2, and 3 respectively. The other parameters are equal to the ones used in [3].

Algorithm 1 MAC decoding operations.

if $v(\hat{\mathbf{z}}) \in [E_0, E_1)$ **then**
  $E'_0 \leftarrow E_0, \quad E'_1 \leftarrow E_1, \quad$ output 0
else if $v(\hat{\mathbf{z}}) \in [E_1, E_2)$ **then**
  $E'_0 \leftarrow E_1, \quad E'_1 \leftarrow E_2, \quad$ output 1
else
  μ detected, **exit**
end if
$done \leftarrow 0$
while $done = 0$ **do**
  if $[E'_0, E'_1] \in [0, 1/2)$ **then**
    $E'_0 \leftarrow 2E'_0, \quad E'_1 \leftarrow 2E'_1$
    $v(\hat{\mathbf{z}}) \leftarrow 2v(\hat{\mathbf{z}})$
  else if $[E'_0, E'_1] \in [1/2, 1)$ **then**
    $E'_0 \leftarrow 2\,(E'_0 - 1/2), \quad E'_1 \leftarrow 2\,(E'_1 - 1/2)$
    $v(\hat{\mathbf{z}}) \leftarrow 2\,(v(\hat{\mathbf{z}}) - 1/2)$
  else if $[E'_0, E'_1] \in [1/4, 3/4)$ **then**
    $E'_0 \leftarrow 2\,(E'_0 - 1/4), \quad E'_1 \leftarrow 2\,(E'_1 - 1/4)$
    $v(\hat{\mathbf{z}}) \leftarrow 2\,(v(\hat{\mathbf{z}}) - 1/4)$
  else
    $done \leftarrow 1$
  end if
end while
$E_0 = E'_0, \quad I' = E'_1 - E'_0$
$E_1 = E'_0 + P_0(1-\epsilon)I', \quad E_2 = E'_0 + P_1(1-\epsilon)I'$

B. Simulation results

In Fig. 2 BER versus cell-to-cell coupling strength factor s performance is shown for different cases. The solid star-marked and diamond-marked curves, referred to as "LDPC only", show the performance of the rate-19/20 ($K = 32794$, $N = 34520$) and the rate-15/16 ($K = 32768$, $N = 34952$) LDPC codes used in [3] and [5] respectively. On the other hand, the solid curves with square and circle marks represent the case $R_C = 9/10$, $M = 1024$ for $P_0 = 0.93$ and $P_0 = 0.95$ respectively. As it can be observed, these test cases achieve significantly better BER performance than the LDPC only case. Even better results are achieved with $R_C = 5/6$ (solid curves with cross and plus marks). Since M represents the maximum number of paths explored at each depth in the tree of all the possible decoded sequences, it is directly related to the speed and the amount of memory and complexity of the MAC decoder. In order to increase the speed and to reduce the complexity of the MAC decoder, the same test cases have been considered with $M = 128$ (dashed curves in Fig. 2). As it can be observed, even if there is a performance loss

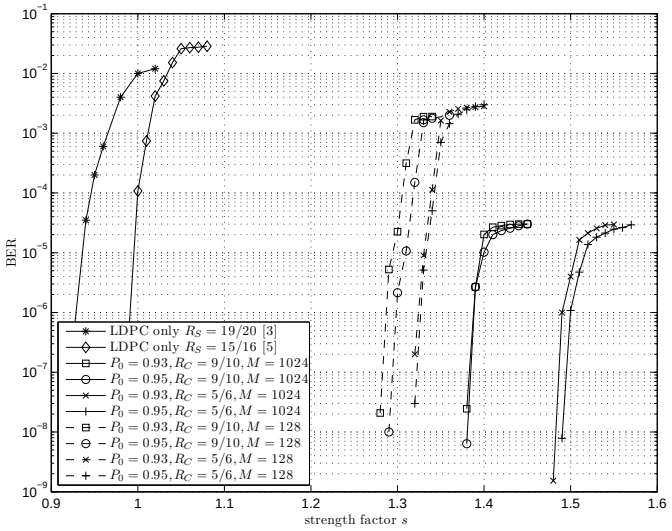

Figure 2. BER versus cell-to-cell coupling strength factor s comparison.

with respect to the case with $M = 1024$, the proposed system still performs better than the LDPC only case. Experimental results for $R_C = 1/4$, corresponding to $\epsilon = 0$, and $R_C = 1$, corresponding to $\epsilon = 0.42$ are not shown as they achieve a BER in the order of 10^{-4} in all the considered cell-to-cell coupling strength factor range. It is worth noting that these results are still valid for other applications, given that system parameters belong to the same ranges.

C. Implementation area and throughput discussion

Several LDPC decoder architectures described in the literature can be used to implement the proposed system, e.g. [9], [10]. On the contrary, to the best of our knowledge, the only work addressing an implementation of the MAC decoder is [11]. However, in [11] the MAC decoder acts as MAP decoder, whereas the proposed technique relies on an ML decoder. Thus, we simplified the architecture used in [11], by removing the backward unit required to implement the BCJR algorithm. As in [11], the MAC decoder architecture contains two AC decoders to perform 0 and 1 extension in the tree exploration, a forward unit to implement (2), two FIFOs to store the status of the AC in the M explored nodes. As argued in [11], since the first FIFO stores the metric obtained from 0 extension and the second FIFO the metric belonging to 1 extension, they are sorted. Thus, with a comparator one can compute the M candidates avoiding the use of a large sorter.

According to [5] an LDPC decoder for NAND flash memories with $R_S = 15/16$, as the one we considered, requires 1.32 mm^2 of area on a 65 nm standard cell technology [12] and, with a clock frequency of 300 MHz, can sustain a throughput of few Gb/s. Moreover, in [5] it is shown that adding rate adaptivity comes at a negligible area overhead (1.36 mm^2 for a dual-rate decoder). The implementation of the MAC architecture proposed in [11] for $M = 128$ works with a sliding-window-based scheduling. When configured as an ML decoder with a window of 125 data, the MAC architecture requires nearly 2 mm^2 on a 90 nm standard cell technology (about 1.5 mm^2 on a 65 nm technology). Given that a 300 MHz clock frequency is employed for the MAC architecture, it sustains a throughput of 1.36 Mb/s. As a consequence, the proposed system recovers a 10 MB bitstream in about a minute. This value is compatible with the reconfiguration time measured in [13] for Xilinx Virtex II Pro devices. Indeed, in [13] it is shown that reconfiguring a Xilinx Virtex II pro FPGA with a 14.6 kB bitstream requires 101 ms, corresponding to about 70 s for a 10 MB bitstream.

IV. Conclusions

In this work a joint source/channel technique to improve the robustness of configuration data of programmable devices against error occurrence in MLC NAND flash memories has been presented. Inspired by the joint source/channel paradigm this work shows that concatenating a MAC and an LDPC code both data compression and improved error correction capability can be achieved with a limited area overhead. The proposed systems features superior BER performance with respect to state of the art techniques relying on LDPC codes and achieves BER values of about 10^{-8} for cell-to-cell coupling strength factors s well higher than 1.0.

References

[1] C. Zambelli, D. Bertozzi, A. Chimenton, and P. Olivo, "Non volatile memory partitioning scheme for technology-based performance-reliability trade-off," *IEEE Embedded Systems Letters*, vol. 3, no. 1, pp. 13–15, Mar 2011.

[2] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit error rate in NAND flash memories," in *IEEE International Reliability Physics Symposium*, 2008, pp. 9–19.

[3] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error correction codes in NAND flash memory," *IEEE Transactions on Circuits and Systems I*, vol. 58, no. 2, pp. 429–439, Feb 2011.

[4] J. Guo, Z. Chen, D. Wang, Z. Shao, and Y. Chen, "DPA: A data pattern aware error prevention technique for nand flash lifetime extension," in *Asia and South Pacific Design Automation Conf.*, 2014, pp. 592–597.

[5] N. Xie, G. Dong, and T. Zhang, "Using lossless data compression in data storage systems: Not for saving space," *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 335–345, Mar 2011.

[6] M. Grangetto, B. Scanavino, G. Olmo, and S. Benedetto, "Iterative decoding of serially concatenated arithmetic and channel codes with JPEG2000 applications," *IEEE Transactions on Image Processing*, vol. 16, no. 6, pp. 1557–1567, Jun 2007.

[7] M. Martina, G. Masera, A. Molino, F. Vacca, L. Sterpone, and M. Violante, "A new approach to compress the configuration information of progammable devices," in *Design Automation and Test in Europe conference*, 2006, pp. 6–10.

[8] L. Sterpone and M. Violante, "A new decompression system for the configuration process of SRAM-based FPGAs," in *ACM Great Lakes Symposium on VLSI*, 2007, pp. 241–246.

[9] J. Kim and W. Sung, "Rate-0.96 LDPC decoding VLSI for soft-decision error correction of NAND flash memory," *IEEE Transactions on VLSI*, vol. 22, no. 5, pp. 1004–1015, May 2014.

[10] C. Condo, M. Martina, and G. Masera, "VLSI implementation of a multi-mode turbo/LDPC decoder architecture," *IEEE Transactions on Circuits and Systems - I*, vol. 60, no. 6, pp. 1441–1454, Jun 2013.

[11] S. Zezza, S. Nooshabadi, and G. Masera, "A 2.63 Mbit/s VLSI implementation of SISO arithmetic decoders for high performance joint source channel codes," *IEEE Transactions on Circuits and Systems I*, vol. 60, no. 4, pp. 951–964, Apr 2013.

[12] A. Pulimeno, M. Graziano, and G. Piccinini, "UDSM trends comparison: From technology roadmap to UltraSparc Niagara2," *IEEE Trans. on VLSI*, vol. 20, no. 7, pp. 1341–1346, Jul 2012.

[13] K. Papadimitriou, A. Anyfantis, and A. Dollas, "An effective framework to evaluate dynamic partial reconfiguration in FPGA systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 6, pp. 1642–1651, Jun 2010.