Eliminating speed penalty in ECC protected memories

Michael Nicolaidis, Thierry Bonnoit, Nacer-Eddine Zergainoh
TIMA Laboratory (CNRS, Grenoble INP, UJF)

Abstract— **Drastic device shrinking, power supply reduction, increasing complexity and increasing operating speeds that accompanying technology scaling have reduced the reliability of nowadays ICs. The reliability of embedded memories is affected by particle strikes (soft errors), very low voltage operating modes, PVT variability, EMI and accelerated circuit aging. Error correcting codes (ECC) is an efficient mean for protecting memories against failures. A major issue with ECC is the speed penalty induced by the encoding and decoding circuits. In this paper we present an effective approach for eliminating this penalty and we demonstrate its efficiency in the case of an advanced reconfigurable OFDM modulator.***)*

Keywords-Reliability, technoloy scalling, ECC, performance

I. INTRODUCTION *(HEADING 1)*

Drastic device shrinking, power supply reduction, increasing complexity and increasing operating speeds that accompany the technological scaling to the nanometric domain, have reduced the reliability of nowadays ICs. The reliability of embedded memories is affected by particle strikes (soft errors); very low voltage operating modes; process, voltage and temperature (PVT) variability; electromagnetic interferences (EMI); and accelerated circuit aging induced by failure mechanisms including NBTI, PBTI and HCI. These trends require new design approaches for enhancing SoC reliability [1-5] and require concurrent error detection and/or correction for logic (e.g. [16], [17] and Error control codes (ECC for memories). As embedded memories represent the largest and denser parts of modern SoCs, they concentrate the majority of failures. Thus, they are the first blocks that designers have to protect in order to meet their reliability ECC is a convenient mean for protecting memories against failures, as they usually achieve high reliability at moderate area and power penalties. One issue with ECC concerns the significant delays added by the encoding circuitry placed on the write side of the memory and by the decoding and error correction circuitries placed on its read side. These delays may induce significant reduction of the clock frequency, which may be inacceptable in many applications. In this paper we present an efficient approach for treating this issue. It eliminates the speed penalty induced by ECC at the cost of very low area and power penalties.

We demonstrate the effectiveness of the proposed approach in the case of an advanced reconfigurable OFDM (orthogonal frequency division multiplexing) modulator [6-8]. In our experiments we consider single error correcting codes (like Hamming and Hsiao), but also multiple error correcting codes (like BCH and Reed-Solomon). Protecting memories against single errors was sufficient in the past as multiple errors were very rare. Thus, codes with single-error correcting capabilities, like the Hamming and Hsiao codes [9] [10] were commonly used in industrial products [11] [12]. However, multiple cell upsets (MCUs) produced by neutron strikes are today very common, as a consequence of the drastic device shrinking in advanced technologies. The problem can be overcome by using column interleaving, which places further apart bits belonging to the same memory word. However, this approach has its limits as further scaling increases the spreading of MCUs [13] [14] and requires a higher interleaving factor. This requires placing large numbers of words in the same memory row, resulting in high row capacitance, low speed, and high power dissipation. Furthermore, in recent memories, designed for variability robustness, pairs of memory cells of the same word must be placed back to back. Also, using multiple error correcting codes is convenient in memories used for temporal storage (e.g. FIFO buffers) on the emission/write port and reception/read port of storage/transmission media/channels, as these media/channels are protected by such codes.

978-3-9810801-7-9/DATE11/©2011 EDAA

II. SPEED PENALTY ELIMINATION SCHEME

Implementing ECC in memories introduces extra delays:

- in the path of write data due to the circuitry used to compute the check-bits, and,
- in the path of read data due to the circuitry used to compute the check-bits and correct the error in read data.

It may be suitable in a design to remove the extra delay from one or both of these paths. Our scheme can do both for systematic codes (i.e. separate information bits and check bits).

A. Delay elimination in the write path

Figure 1 depicts the block diagram of our solution for eliminating extra delay in the path of the write data. We illustrate the solution for the Hamming code but it is similarly applied to any systematic code. As shown in figure 1, the data-bits and the check-bits are stored in two separate memories (data-memory and code-memory). We can see in this figure the block generating the check-bits from the data-bits to be written in the data-memory, as well as the block generating the check-bits from the data-bits read from the data-memory. The later check-bits are bit-wise XORed with the check-bits read from the code-memory by a stage of XOR gates (XOR1), which produces the error syndrome. This syndrome indicates the

position of the erroneous bit in a binary-code form. A combinational block transforms this form into a 1-hot code (1-hot block). Finally, a second block of XOR gates (XOR2) bit-wise XORs the 1-hot form of the syndrome with the data-bits read from the data-memory and the check-bits read from the code-memory to correct eventual errors affecting these bits. In this scheme, the data-bits are written in the data-memory as soon as they are ready. Thus, the frequency of the clock signal Ck is the same as in the case of a system which does not implement ECC. This eliminates the speed penalty on the write side. However, an extra delay is added on the inputs of the code-memory by the code generation block. To manage this delay, the code-memory uses a clock signal Ck+δ, which has the same frequency as signal Ck, but it is delayed with respect to this signal by a delay δ equal to or larger than the delay of the code generation block. As a matter of fact, the check-bits are ready when the signal Ck+δ enables writing these check-bits in the code-memory. Nevertheless, since the signal Ck+δ enables both the write and the read operations in the code-memory, then, during a read operation, the check-bits will be read with a delay δ with respect to the data-bits read from the data-memory. However, this delay does not increase the delay of the error detection/correction process, because the check-bits read from the code-memory are applied directly to the XOR block, while the data-bits read from the data-memory traverse the code generation block, which has a delay equal to δ (same as the code generation block used in the write side).

The delayed clock signal Ck+δ can be generated locally, by adding a delay element on the Ck signal of the data-memory. This local generation minimizes skews between the clock signals of the data-memory and the code-memory. Careful implementation of the delay element will consider worst-case delay of the code-generation block as well as best-case (minimum) delay of this element. Another possibility is to use a single clock signal for both memories, but use the one edge of this signal (e.g. the rising edge) as the active edge for the data-memory, and its second edge (e.g. the falling edge) as the active edge for the code-memory. This will work if the delay of the code generation block does not exceed the time interval that separates theses edges.

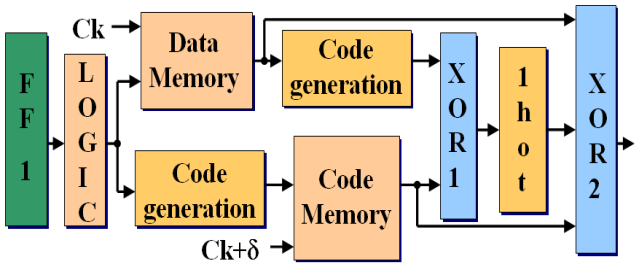

Fig. 1: Elimination of extra delay in the path of write data

Another implementation, useful when the delay for the code generation is very large, consists on adding a pipe-line stage in the code generation block. With this solution, the operations in the code-memory will be performed one clock cycle later than in the data-memory. Also, if the delay of the code generation block is larger than one clock cycle, more pipe-line stages are added. This means that a write in the code-memory will be performed more than one clock cycle after a write in the data-memory. This large delay is balanced by the delay of the code generation block placed on the read data, since, in this block, a similar number of pipe-line stages are added. Long delays in the later block will be handled by the scheme that removes speed penalty on the read side, as described next.

B. Delay elimination in the read path

Figure 2 depicts the block diagram of a system where the data read from a memory pass through a combinational logic block (Logic1) and are stored in a stage of flip-flops (FF1). Logic1 can be empty (i.e. the read data enter directly FF1).

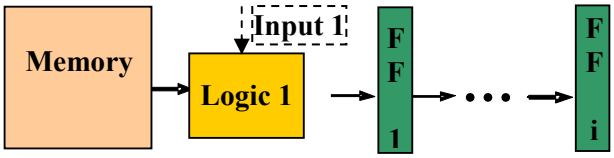

Fig. 2: A system with an unprotected memory

In figure 2, the memory is not protected by an ECC. If ECC is used, the error detection and correction circuitry will be placed between the memory and the combinational block Logic1, increasing significantly the signal delay and decreasing accordingly the clock frequency. In many applications, it may be required to maintain the clock frequency of the initial (unprotected) design. For doing so, one solution consists on adding one or more stages of flip-flops between the memory and FF1. This solution will allow reaching the desirable clock frequency. However, performance will still be impacted as the data read from the memory will reach the flip-flop stage FF1 one or more clock cycles later than in the unprotected design. This may not be desirable in many applications.

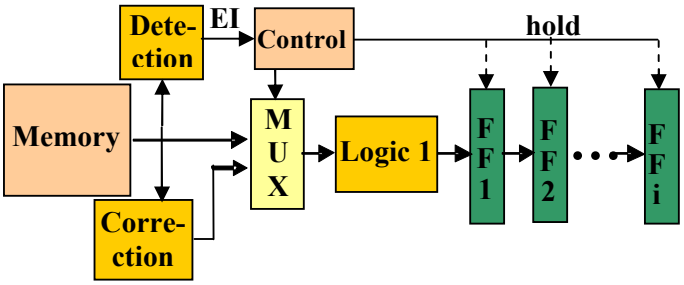

Fig. 3: Reduction of the speed penalty on the path of read data

Detecting an error is faster than correcting it. Based on this observation, the clock frequency can account only for the delay of the error detection signal. Thus, a second solution is shown in figure 3. The data read from the memory are supplied to the system through a MUX. The first inputs of the MUX come directly from the memory and the second inputs from the error correction block. In error-free operation, the first inputs of the MUX are supplied to the system. When the error indication signal (EI) detects an error, the system is halted for one or more clock cycles to provide extra time for performing the error correction. At the end of these cycles, the second inputs of the MUX are supplied to the system. While this

solution reduces the speed penalty, still the designer may not achieve its target speed due to the delay of error detection.

So, we are looking for a solution enabling eliminating clock speed reduction without adding extra pipe-line stages. The detailed implementation of the proposed solution depends on the relations between the delays of the detection circuitry and of the correction circuitry (i.e. in how many cycles after read the error detection signal and the corrected data are ready). Thus, several cases are considered. We start with the simpler ones to present the basic principles of the approach in simple way. Then we generalize them for more complex cases.

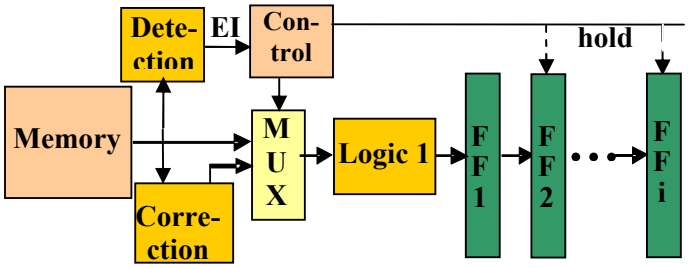

Fig. 4: Elimination of error detection and correction delays

Case a: The error detection signal and the corrected data are ready one clock cycle after FF1 latches the erroneous data. In this case, the proposed solution is illustrated in figure 4, for the design of figure 2. For easier illustration, the detection and correction blocks are shown to be distinct, but in reality they share the code generation block. The idea here is to supply, through a MUX, the data to the system as soon as they are read. Thus, we can use a clock period that does not account for the error detection and error correction delays. In this case, the system will work properly as long as no errors are present in the data read from the memory. During this time the MUX supplies to the system the data coming directly from the memory. However, when an error is present in the read-data, it is propagated and contaminates the system flip-flops before the error detection signal indicates its occurrence. To handle this issue, the activation of the error detection signal EI activates during the next clock cycle the hold signal of all flip-flops except the first stage of flip-flops (FF1). It also enables the MUX to supply the data coming from the error correction block. Thus, during one clock cycle all the flip-flop stages but FF1 are hold, maintaining their previous state. During the same clock cycle, the stage FF1 is decontaminated, since it receives the corrected data. Then, the hold signal is deactivated and the system resumes operation from a correct state.

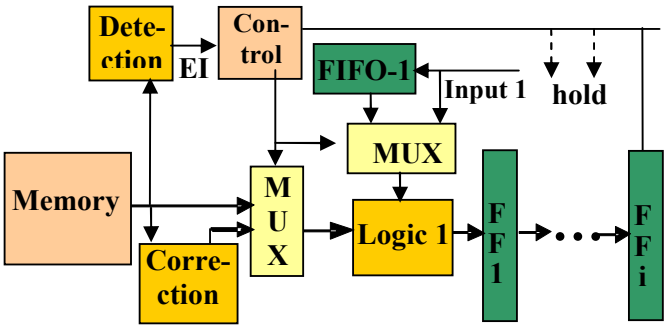

Fig. 5: Handling lateral inputs

The above implementation works when only data originated from the erroneous read value enter the pipe-line stages of the decontamination path. If some other inputs enter these stages (as "Input 1" signaled in figure 2 by a dashed arrow), then, during the decontamination phase the values applied on these inputs have to be the same as the ones applied during the error propagation phase. As a matter of fact, a FIFO is added on these inputs, to preserve their values and apply them later on the circuit through a MUX, as shown in figure 5. As in this figure the values coming from "Input 1" are propagated during one clock cycle before activating the hold signal, then, a single stage FIFO is used (i.e. FIFO-1 is just a stage of flip-flops).

Case b: The error detection signal is ready one clock cycle after the uncorrected data are latched in FF1 and the corrected data are ready m clock cycles after this time (the correction block will comprise m pipe-line stages to handle its delay, with m>1). The implementation of our solution is as in case *a* (figure 4). However, when an error is detected the control is different than the one corresponding to figure 4. In fact, in the present case, the hold signal is maintained active for m-1 cycles to give the time required for the data to be corrected and stored in FF1 m cycles later. During the same period the control signal of the MUX is maintained high. Thus, at the end of the m^{th} cycle, when the data are corrected, the inputs of FF1 come from the correction block. Thus, at the m^{th} cycle all FF stages contain correct data and the hold signal is released at the next cycle enabling all pipeline stages to resume operation.

Case c: The error detection signal and the corrected data are ready k cycles after the uncorrected data are latched in FF1 (the detection block and the correction block will comprise k pipe-line stages, with k>1). Note that, for holding the system, we need to take into account the delay of the detection block plus the delay of the interconnections distributing the hold signal. Thus, the k pipe-line stages of the detection block also account for the delay of these interconnections. In the present case (*c*), the erroneous data are propagated through k pipe-line stages before the system flip-flops are hold. Thus, the decontamination process will take k clock cycles. Furthermore, the number of cycles during which different FF stages are hold is variable: The very first flip-flop stage following the correction block is not hold at all; the FFs that are one stage further are hold for one clock cycle; the FFs that are two stages further are hold for two clock cycles; …; the FFs that are k stages further as well as all other FFs are hold for k clock cycles. In addition, as an extension of the principle of figure 5, if some data not originated from the erroneous read values enter the pipe-line stages of the decontamination path (like "Input 1" in figure 2), then, during the decontamination the values applied on these inputs have to be the same as the ones applied during the error propagation. Thus, FIFOs are added on these inputs to preserve their values and apply them (through a MUX) during the decontamination phase. The number of the stages of each FIFO depends on its position in the decontamination path. For inputs connected to the very first stage following the correction block the FIFO will have k stages; for inputs connected at one stage further the FIFO will have k-1 stages; …; for inputs connected k-1

stages further the FIFO will have 1 stage. Also, each FIFO will be hold during the same number of cycles as the flip-flops of the pipeline stage on which it is connected.

Case d: The error detection signal is ready k clock cycles after the uncorrected data are latched in FF1 and the corrected data are ready m cycles after this instant (the detection block comprises k pipe-line stages and the correction block comprises m pipe-line stages, with m>k>1). For this case, the implementation of our solution is as in case *c*. However, when an error is detected the control is different. In fact, as in the present case the corrected data are ready m-k cycles after the error detection signal activates the hold signal, we have to hold the system flip-flops for m-k cycles (to give the time required for the data to be corrected and stored in FF1), before starting the decontamination phase described in case *c*.

In the above we present the principles of the proposed approach. However, numerous particular cases concerning the placement of the FIFOs, their depth and the number of cycles certain FF stages have to be hold are not presented here for space reasons. The complete algorithm was implemented in an automation tool which will be described in a further communication. This tool is interactive, as, in certain cases, it proposes several solutions concerning the placement and depth of FIFOs and leaves the designer to make the final choice. In some other cases, it requests the designer some functional information concerning the way certain paths are used, in order to determine the optimal solution.

The proposed approach allows implementing ECC in embedded memories without affecting the clock frequency of the design, neither introducing extra pipe-line stages. The counterpart is to pay some extra clock cycles each time an error is detected. However, the performance reduction introduced by these extra cycles is insignificant. As an illustration, let as consider a SoC comprising 50 Mbytes of embedded memories (which is on the upper limits of what is today possible with advanced technologies). Let us also consider that the SER (soft error rate) is 1000 FIT per Mbit (which is rather on the high side of SER encountered in current technologies - usually several hundreds FIT per Mbit). Then, the total SER will be 60x8x1000 FIT = $48x10^{-3}$ failures per hour, 1 FIT = 10^{-9} failures per hour) or about 1 failure every 115 days. This MTBF is totally unacceptable for numerous applications including networking, servers, automotive etc, and imposes using ECC to protect the embedded memories of the SoC. As ECC implementation may involve inacceptable performance penalty, the proposed approach can be used to handle this issue. Considering a rather large delay for error correction (e.g. three clock cycles), the described approach will induce loosing three clock cycles every 115 days. For a 200 MHz clock frequency we will have a performance reduction of 10^{-8} %, which is totally insignificant!

Note that, as the function of processors is to execute instructions, processors realize certain instruction manipulation functions such as bubbling, flushing, … Thus, in processors, reduction of ECC speed penalty during read can be done at the functional level. The similar technique as the one in figure 3, will be typically implemented by stalling the pipe-line. Complete elimination of ECC penalty can also be done at the functional level, but is more complex way. For instance, after error detection the pipeline is flushed - the instruction affected by the incorrect data and the instructions fetched after it are disregarded (similarly to branch misprediction recovery). Then, the discarded instructions are replayed, with the first of them using data coming from the error correction circuit. These approaches are limited to processor only designs. However, a SoC may comprise hundreds of memories, with several of the used by non-processor logic. Thus, the designers need a generic (preferably automated) approach, to implement fast ECC in any design. The approach described above is generic, as it works for any RTL design, and is automated. A non-processor design is consider in the next section as case study.

III. CASE STUDY AND IMPLEMENTATION

The case study consists in an advanced reconfigurable OFDM modulator [6-8], proposed in the context of Software Design Radio (SDR). SDR supports several standards with a unique terminal. OFDM modulation is particularly important for the development of SDR, because of the many standards using this kind of modulation (B3G, 4G, WIFI, WIMAX, WRAN). The reconfigurable architecture considered here is able to compute several modulations: OFDM QAM and OQAM. These modulations are based on FFT or IFFT using RADIX algorithms, and IOTA filtering for OQAM. IOTA stands for "Isotropic Orthogonal Transform Algorithm pulse shaping filtering". IOTA filtering improves performance against cosite interference, impulsive noise, and frequency-selective fading [6]. Consequently, it does not require guarding interval for the same performance. On the other hand, OFDM symbols have to be computed twice faster in average. The design supports a variable number of sub-carriers from 64 to 8192, and is able to compute up to 4 modulations in parallel.

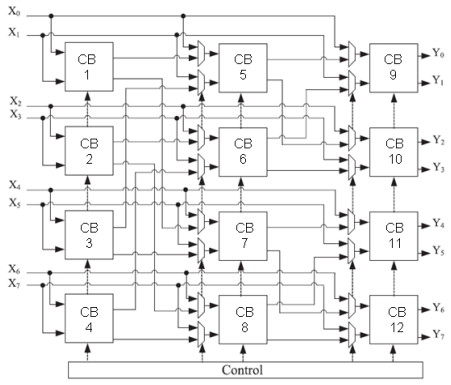

Fig. 6: Reconfigurable computing matrix

This design is a memory intensive parallel architecture and the protection of memories against failures is highly desirable. It is composed of reconfigurable data path computing matrix, divided in 12 blocks of complex multiplications and accumulations (Fig. 6). 2 ROMs store FFT and IOTA coefficients. 2 RAMs store samples for FFT and IOTA filtering computations. The RAM dedicated to FFT is divided in two blocks which are alternately used. Each block is composed of 8 sub-blocks of complex samples. Equivalently,

we may consider that each block has 16 input/outputs. Each of the 16 storages is a reconfigurable memory block, which configuration depends on the number of sub-carriers and MIMO. Regarding the computing matrix, the configuration depends on the algorithm chosen (Radix 2, 4 or 8), the number of MIMO, and IOTA filtering. The Calculus Blocks (CB), which are not involved in computation, can be deactivated. Registers can also be deactivated in each of the busy block, depending of the current calculus mode.

The approach for speed penalty elimination, presented in the previous section, was first implemented manually for the RAM blocks storing the FFT samples (FFT RAM blocks). As data can be written and read at the same time in these blocks, separate encoding and decoding circuits were used On the other had, the control block is common to all memory I/O in order to simplify the clock control.

The read clock has a phase difference of π with the clock of the rest of the design. Consequently, if we insert the detection and correction blocks in this design we will dispose less than a half clock cycle for detection and correction. This time is insufficient and will require either reducing the clock frequency or adding extra pipeline stages. In both cases, we will have a drastic throughput reduction. Thus, the approach described in the previous section is very suitable.

Subsequently we used the automation tool to implement our approach. The solution proposed by the tool, based on structural (RTL) information only was more expensive in hardware. However, the tool was able to specify the paths in which some functional information could allow cost reduction, and once we provided this information, the tool proposed the same solution as the one found by manual analysis. It is worth noting that the manual analysis took several months of work, while the response of the tool is virtually instantaneous.

The most complex task for the manual implementation was to analyze the complex architecture of the *reconfigurable* OFDM modulator and determine the places in the decontamination path where FIFOs and MUXes have to be inserted (positions where data not originated from the erroneous read values enter this path). As we have seen in the previous section, the length of the decontamination path and the size of the FIFOS depend on the delay of error detection. Thus, the analysis varies from one code to another. The detailed presentation is lengthy and goes along with a detailed presentation of the circuit. Thus, we only present the general principles that confirm the decisions of the RTL tool by the manual analysis: Outside the reconfigurable matrix, some signals have to be stored for saving the operating context needed during decontamination. The most important are the previous configuration of the matrix, and the identification of the concerned FFT RAMs (Fig. 7). According to the Fig. 6, each Calculus Block in the matrix can be deactivated on demand. For instance, if the first stage (CB 1, 2, 3 and 4) is involved in a correction, the other blocks can be stopped. If the second stage is involved (CB 5, 6, 7 and 8), the first stage is already deactivated, and the third stage has to be stopped. By deactivating CBs handling data that are not corrupted, their state is preserved. However, due to their complexity, the detailed description of CBs is required for presenting the details concerning the case when CBs are activated.

The most complex of the processing elements of the design has a pipelined architecture with feed-back loops. Thus, FIFOs and MUXes are used to save the context (Fig. 8).

The decisions made by this analysis confirm the decisions made by the tool. Furthermore the implementation was extensively validated by error injections on the read data.

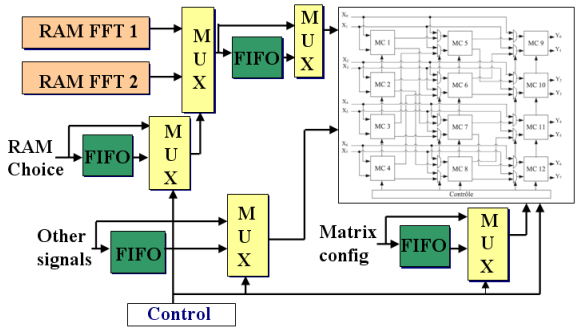

Fig. 7: Additional FIFOs and MUXs for saving context outside the matrix

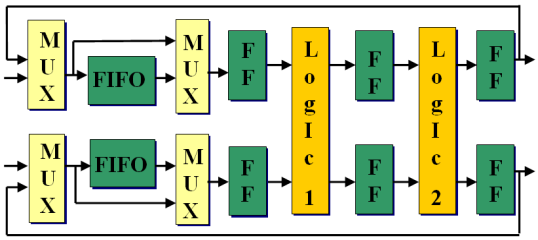

Fig. 8: Additional FIFOs and MUXs for saving context inside the matrix

IV. RESULTS

The circuit together with the various ECC implementations were synthesised on a 65 nm technology with Design Vision and simulated with Modelsim. The clock frequency of the original design (i.e. without introducing ECC), is 278 MHz. Several single-error correcting and multiple-error correcting codes were implemented: extended Hamming (22, 16, 1), Hsiao (22, 16, 1), BCH (31, 16, 3), Reed-Solomon (56, 16, 17), as well as Reed-Solomon (20, 8, 4) using the approach in [15] to reduce the complexity. In the parentheses, the first number gives the total number of bits (data plus check bits), the second number gives the data bits and the third the number of the correctable errors. The data width in the design is 16 its. The 8 data bits in Reed-Solomon (20, 8, 4) mean than the 16 bits are divided in two parts and each part is checked by a (20, 8, 4) Reed-Solomon code. The circuit operation under incorrect data was verified by error injections at the read data.

Table 1 presents the results for the different implementations. Column 1 presents the different codes that were evaluated. Column 2 presents the maximum operating frequency obtained with the standard implementation of these codes, and column 3 the maximum operating frequency obtained with the proposed implementation. Columns 4 and 5

present the area and power penalties induced by the proposed ECC implementation in comparison with the standard ECC implementation. These penalties are computed on the basis of the area and power of the memories and of the standard implementation of the ECC circuitry and not on the basis of the area and power of the whole design. These overheads will be lower if we compute them on the basis of the area and power of the whole design, but such a computation will not be fair as the proposed approach improves the performance related to the ECC insertion.

In all cases, the frequency allowed by the new implementation is equal to 278 MHZ, which is the frequency of the unprotected OFDM circuit (no ECC for memories). We observe drastic speed improvements with respect to the standard implementations of the different codes (40% for Hamming, 42% for HSIAO, and up to 329% for Reed-Solomon (56, 16, 17)), while the area and power penalties with respect to the same implementations are very low.

As concerning the extra cycles induced in case of error detection: For the Hamming and HSIAO codes, the erroneous data contaminate one flip-flop level before error detection and activation of the hold signal. Thus, we used one clock cycle for performing decontamination. For BCH (31, 16, 3) and RS (20, 8, 4), the erroneous data contaminate two flip-flop levels before the activation of the hold signal. Thus, we used two clock cycles for performing decontamination. Finally, for RS (56, 16, 17), the erroneous data contaminate three flip-flop levels before the activation of the hold signal. Thus, we used three clock cycles for performing decontamination. In addition, for this code, error correction delay requires inserting one wait cycle before starting decontamination. This means that according to the code used, at each error detection, computations will take one, two or four extra clock cycles. However, as shown in the previous section this represents insignificant performance penalty, even for complex codes like BCH or Reed-Solomon. Thus, even such complex codes can be implemented without performance penalty.

OFDM design	Fr standard	Fr New	Area New	Power New
Hamming (22,16)	198 MHz	278MHZ	+2,64%	+6,01%
Hsiao (22, 16)	196 MHz	278MHZ	+2,69%	+3,62
BCH (31, 16, 3)	151 MHz	278MHZ	+2,15%	+9,04
RS (56, 16, 17)	64,7 MHz	278MHZ	+2,24%	+8,27
RS (20, 8, 4)	171 MHz	278MHZ	+1,91%	+6,75

Table 1

CONCLUSION

In this article we have presented an approach for eliminating the speed penalty related with the implementation of ECC in embedded memories. The approach is generic (works for any systematic code and for any kind of design), it completely eliminates the speed penalty induced by the ECC circuitry, and is shown by means of a practical case study (an advanced reconfigurable OFDM modulator) to require very low area and power cost. We show that these results are valid even for complex codes like BCH or Reed-Solomon, which could otherwise induce very high speed penalty.

REFERENCES

[1] R.C. Baumann, "Soft Errors in Advanced Computer Systems," IEEE Design and Test of Computers, vol. 22, no. 3, pp. 258-266, May/June 2005.

[2] K. A. Bowman et al, "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance", IEEE J. of Solid State Circuits, Vol. 44, No. 1, Jan. 2009, pp. 49-63.

[3] M. Nicolaidis, "Design esign for soft error mitigation", IEEE Transactions on Device and Materials Reliability, Vol. 5 No 3, pp 405 – 418

[4] C. Metra, "Trading Off Dependability and Cost for Nanoscale High Performance Microprocessors: The Clock Distribution Problem" 2009 Workshop on Dependable and Secure Nanocomputing, June 29, 2009, Lisbon Portugal.

[5] S. Lin, Y.B. Kim, F. Lombardi, "A novel design technique for soft error hardening of Nanoscale CMOS memory", 52nd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS '09), PP. 679 - 682 AUGUST 2-5, 2009, CANCUN, 2009.

[6] M. Muck, J-P Javaudin. "Advanced OFDM Modulators considered in the IST-WINNER Framework for Future Wireless Systems", 14th IST Mobile and Wireless Communications Submit, 2005.

[7] C. Sahnine, J.-P. Javaudin, G. Degoulet, B. Jahan, "OFDM/OQAM Transceiver Implementation", Design and Architectures for Signal and Image Processing (DASIP 2007), Grenoble, France, November 27-29, 2007.

[8] C. Sahnine, "Architecture de circuit intégré reconfigurable, très haut débit et basse consommation pour le traitement numérique de l'OFDM avancé", Ph.D. diss., Grenoble INP, France, 2009.

[9] C. L. Chen, and M.Y. Hsiao. "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review", IBM J. Res. Devel., vol. 28, no. 2, pp. 124-134 (1984).

[10] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SECDED codes," IBM J. Res. Devel., vol. 14, pp. 395-401, (1970)

[11] K. Gray, "Adding Error-Correcting Circuitry to ASIC Memory", IEEE Spectrum, pp. 55-60, Apr. 2000.

[12] S. Ghosh, S. Basu, N.A. Touba, Selecting Error Correcting Codes to Minimize Power in Memory Checker Circuits , J. Low Power Electronics 1, pp.63-72(2005).

[13] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, T. Toba, "Scaling Effects on Neutron-Induced Soft Error in SRAMs Down to 22nm Process".3rd Workshop on Dependable and Secure Nanocomputing,June 2009, Lisbon, Portugal.

[14] E. Ibe, S. Chung, S. Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto, and T. Akioka, "Spreading Diversity in Multi-cell Neutron-Induced Upsets with Device Scaling," IEEE Custom Integrated Circuit Conference, pp. 437-444, 2006.

[15] S. M. Jahinuzzaman, J. Singh Shah, D. J. Rennie, M. Sachdev. "Design and Analysis of A 5.3-pJ 64-kb Gated Ground SRAM With Multiword ECC", IEEE Journal of Solid-State Circuits, Vol. 44, No. 9, pp 2543-2553,(2009)

[16] M. Nicolaidis, "Shorts in Self-checking Circuits", Journal of Electronic Testing, Springer, Vol. 1, No 4, pp. 257-273, (1991)

[17] M. Nicolaidis, R.O. Duarte, "Fault-secure parity prediction Booth multipliers", Design & Test of Computers, IEEE, Vol.: 16 Issue: 3, pp. 90-101, (2002)