## POLITECNICO DI TORINO

Repository ISTITUZIONALE

## Design and Optimization of Adaptable BCH Codecs for NAND Flash Memories

Original
Design and Optimization of Adaptable BCH Codecs for NAND Flash Memories / Fabiano, Michele; Indaco, Marco; DI CARLO, Stefano; Prinetto, Paolo Ernesto. - In: MICROPROCESSORS AND MICROSYSTEMS. - ISSN 0141-9331. STAMPA. - 37:4-5(2013), pp. 407-419. [10.1016/j.micpro.2013.03.002]

## Availability:

This version is available at: $11583 / 2506420$ since

Publisher:
Butterworth Heinemann Publishers:Linacre Editore attuale..ELSEVIER SCI LTD, THE BOULEVARD,

Published
DOI:10.1016/j.micpro.2013.03.002

Terms of use.

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
(Article begins on next page)

# Design and Optimization of Adaptable BCH Coaedes for 

 NAND Flash MequiriesAuthors: S. Di Carlo, M. Fabiano, M. Mdado, Jand P. Prinetto Published in the Microproceroms and Microsystems Vol. 37 ,Issues. 4-5, 2013, pp. 407-419.

## N.B. This is a copy of theACCEPTED version of the manuscript. The final PUBLISHED manuseriptis available on SienceDirect:

URL: http://wnw.sciencedirect.com/science/article/pii/S0141933113000471
DOI: 10.1016xi.micoro.2013.03.002
© 2013 E-kevier. Personal use of this material is permitted. Permission from Elsevier must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

# Design and Optimization of Adaptable BCH Codecs for NAND Flash Memories 



NAND flash memories represent akey storage techmpiey for solid-state storage systems. However they suffer from seriows ryliability and endurance issues that mustbe mitigated by the use pif proper error correction codes. This paper proposes the design and impornentation of an optimized Bose-Chaudhuri-Hocquenghem iardware codec core able to addpt its correction capability in a range phodefined values. Code adaptability makes it possible to efficientry trade-Qf, in-field relionility and code complexity. This feature is reey important considering that the reliability of a NAND flash memory continuously decreases over time, meaning that the required correction capability is not fixed during the life of the device. Experimental resfilts show that the proposed architecture enables to save resources when the device is in the early stages of its lifecycle, while introducing a limited overhead in terms of area.

Key words:
Flash memories, Error correcting codes, memory testing, BCH codes

## 1. Introduction

NAND flash memories are a widespread technology for the development of compact, low-power, low-cost and high data throughput mass storage systems for consumer/industrial electronics and mission critical applications. Manufacturers are pushing flash technologies into smaller geometries to fr e the reduce the cost per unit of storage. This myles moving frointraditonal single-level cell (SLC) technologies. able to store a single bit ofioformation, to multi-level cell (MLC) te enologies, storing mgr than one bit per cell.


The strong transistor minnaruization and the adoption of an increasing number of levels per cell introduce serious issues related to yield, reliability, and endurance ? ? ? . Error correct) on codes (ECCl) must therefore be systeraticaty applied. DeCs are agost-efficient techarique to detect and ${ }^{13}$ correct multiple errors [?] ]. Fash memories support ECCl by providing ${ }_{14}$ spare storage cell dedicated to system maregempentand parity bit storage, ${ }^{15}$ while demanding the actual implementation to the application designer $\left[\begin{array}{ll}? & 16\end{array}\right.$ ? ]. Hosing the correction capability of an ECC is a trade-off between ${ }^{17}$ reliability and code complexity. It is therefore a strategic decision in the ${ }_{18}$ deIgn of a flash-based storage system. A wrong choice may either overestimate or underestimate the required redundancy, with the risk of missing the target failure rate. In fact, the reliability of a NAND flash memory continuously decreases over time, since program and erase operations are somehow destructive. At the early stage of their life-time, devices have a reduced error-rate compared to intensively used devices [?]. Therefore, designing an ECC system whose correction capability can be modified in-field is an attrac-

12
tive solution to adapt the correction schema to the reliability requirements the flash encounters during its life-time, thus maximizing performance and ${ }^{27}$ reliability.

This paper proposes the hardware implementation of an optimized adaptable Bose - Chaudhuri - Hocquenghem (BCH) codec core for NAND flash ${ }_{30}$ memories and a related framework for its autontatic gearation.

Even though there is a considerable literature about efficient $(\mathrm{BCH}){ }^{32}$ coder/decoder software implementations [? ? ? ], modern foem-based mem- ${ }^{33}$ ory systems (e.g., Solid State D令es (SSDS)) usually resort taspecific high ${ }_{34}$ speed hardware IP core [?? in orger to minimizethe memory latency. This ${ }_{35}$ is motivated by the foet that Contemporary hign-density MLC flash mem- ${ }^{36}$ ories require a vare powerful error correction capability, and, at the same ${ }_{37}$ time, they hatetomfet more dernaining requirements in therms of read/write ${ }_{38}$ latency.


Given this premise we will tackle a BCH hardevane implementation for encoding and Necoding tasks. In particular the main contribution of the

The paper is organized as follows: Section ?? shortly introduces basic

## 2. Background and related works

Several hard- and soft-decision error correction codes have been proposed ${ }_{52}$ in the literature, including Hamming based block codes [? ? ], Reed-Solomon codes [? ], Bose-Chaudhuri-Hocquenghem (BCH) codes [? ], Goppa codes 54 [? ], Golay codes [? ], etc.

Even though selected classes of codessuah as groppa codes hatobeeh 56 demonstrated to provide high correctiondefiency f? , when considering the ${ }_{57}$ specific application domain of flash remorids, the need topradeoff code effi- ${ }_{58}$ ciency, hardware complexity andperformances have neryd both the scientific 59 and industrial community towerd a set of codes that enabie very efficient and optimized hardware inppinmentations $\%$


Hamming based block codes. both softwase and hardware, byig they offer very hinited conrection capability ${ }_{64}$ [? ? ]. As the epror rate increased with sucessive generations of both SLC ${ }_{65}$ and MLC NATD flashimemories, design ers moved to more complex and pow- ${ }_{6}$ erful quas includirg Reed-Solomon (RS) codes [? ] and Bose-Chaudhuri- ${ }^{67}$ Hecquengher ( BCH ) codes [? ]. Both codes are similar and belong to the ${ }_{68}$ larber chass of cyclic codes which have efficient decoding algorithms due to better suited when errors are expected to occur in bursts, while BCH codes perform correction over single-bit symbols and better perform when bit errors are not correlated, or randomly distributed. In fact, several studies have reported that NAND flash memories manifest non-correlated or randomly
distributed bit errors over a page [? ] making BCH codes more suitable for ${ }_{76}$ their protection.

An exhaustive analysis of the mathematics governing BCH code is out ${ }_{78}$ of the scope of this paper. Only those concepts required to understand the ${ }^{79}$ proposed hardware implementation will be shortly discussed. It is worth to ${ }_{80}$ mention here that, since several publications prososed vesy efficient hardware si implementations of Galois fields polynomia manipulations, sueh ranipyla- 82 tion will be used in both encoding apat decodirg operations ? ?

Given a finite Galois field $G F\left(2^{m}\right)$ (with $m \geq 3$ ), a $t$-error-correcting BCH ${ }_{84}$ code, denoted as $B C H\left[n_{n} k, 4\right.$. enesdes a $k$-bit neqsage $b_{k-1} b_{k-2} \ldots b_{0}\left(b_{i} \in{ }_{85}\right.$ $G F(2)$ ) to a $n$-bit codemord $b_{b-1}, b_{k-2} \ldots b_{0} p_{r-1} p_{r-2}$.. $p_{0}\left(b_{i}, p_{i} \in G F(2)\right)$ by ${ }_{86}$ adding $r$ parity/fits to the original message. The number $r$ of parity bits ${ }_{87}$ required to dorrect errors iny the $\boldsymbol{p}^{2}$-bit codeword is cornputed by finding 88 the mininvim $m$ that sofves the inequality $k+{ }_{89}$ $m \cdot t$. Whenever po =k,tr<2m-1, the BCH eodde is called shortened 90 or polynomig!. In a shortened BCH code the eodeword includes less binary symbols than the ones the selected Gabis held would allow. The missing 92 information symbols are imagined to be at the beginning of the codeword ${ }_{93}$ An are ecinsidered to be 0 . Let $\alpha$ be a primitive element of $G F\left(2^{m}\right)$ and $\psi_{1}(x)$ a primitive polynomial with $\alpha$ as a root. Starting from $\psi_{1}(x)$ a set of $\quad 95$ minimal polynomials $\psi_{i}(x)$ having $\alpha^{i}$ as root can be always constructed [? ${ }_{9}$ ]. For the same $G F\left(2^{m}\right)$, different valid $\psi_{1}(x)$ may exist [?]. The generator ${ }_{9}$ polynomial $g(x)$ of a $t$-error-correcting BCH code is computed as the Least 9 Common Multiple (LCM) among $2 t$ minimal polynomials $\psi_{i}(x)(1 \leq i \leq 2 t) . \quad 99$ Given that $\psi_{i}(x)=\psi_{2 i}(x)(\forall i \in[1, t])[?]$, only $t$ minimal polynomials must ${ }_{100}$
be considered and $g(x)$ can therefore be computed as:

$$
\begin{equation*}
g(x)=L C M\left[\psi_{1}(x), \psi_{3}(x) \ldots, \psi_{2 t-1}(x)\right] \tag{1}
\end{equation*}
$$

When working with BCH codes, the message and the codeword can be 102 represented as two polynomials: (1) $b(x)$ of degree $k-1$ and (2) $c(x)$ of degpee ${ }^{103}$ $n-1$. Given this representation, both the encering andtrne decoding precess you can be defined by algebraic operations ariong polfnomials in $G \mathbb{A}\left(2^{2}\right)$. The 105 encoding process can be expressed

where $\operatorname{Rem}(m(x) \cdot x)$ denotes the remainder of the division between the ${ }_{107}$ message leftshifte○ait $r$ positicing and the generator polynomial $g(x)$. This 108 remainder rebresents the $r$ parity bits to append to the enginal message. 109

The BCH decodire process searches for the position of erroneous bits ${ }_{110}$ in the codeword. This eperation requires (hree nating computational steps: 111 1) syndrome computation, 2) error locator polynomial computation, and 3) ${ }^{112}$ error position computation. first the computation of $2 t$ syndromes of the codeword $c(x)$, each associ- ${ }_{115}$ ated with one of the $2 t$ minimal polynomials $\psi_{i}(x)$ generating the code. ${ }^{116}$ Syndromes are calculated by first computing the remainders $R_{i}(x)$ of the ${ }_{117}$ division between $c(x)$ and each minimal polynomial $\psi_{i}(x)$. If all remainders ${ }_{118}$ are null, $c(x)$ does not contain any error and the decoding stops. Otherwise, 119 the $2 t$ syndromes are computed by evaluating each remainder $R_{i}(x)$ in $\alpha^{i}$ : ${ }^{120}$ $S_{i}=R_{i}\left(\alpha^{i}\right)$. Practically, according to (??), given that $\psi_{i}(x)=\psi_{2 i}(x)$, only ${ }_{121}$
$t$ remainders must be computed and evaluated in $2 t$ elements of $G F\left(2^{m}\right)$. ${ }^{122}$
The most used algebraic method to compute the coefficients of the error ${ }^{123}$ locator polynomial from the syndromes is the Berlekamp-Massey algorithm ${ }_{124}$ [? ]. Since the complexity of this algorithm grows linearly with the correction ${ }_{125}$ capability of the code, it enables efficient hardware implementations. The ${ }_{126}$ equations that link syndromes and error locator polyomal can be expressed try as:

The Berickamp=Massey aroninin itenatively solves the system of equa- 130 tions defined in (??) using consecutive approximations 131

Finally, the Chen Machine searches for the rops of the error locator 132 polynomial ( basically evaruates/the polynomial $\lambda(x)$ ineach element $\alpha^{i}$ of $G F\left(2^{m}\right)$. If $\alpha^{i}{ }^{134}$ satisfles the gquation $1+\lambda_{1} \alpha^{i}+\lambda_{2} \alpha^{2 i}+\ldots+\lambda_{t}\left(\alpha^{i}\right)^{t}=0, \alpha^{i}$ is a root of the ${ }_{135}$ er -1 racator polynomial $\lambda(x)$, and its reciprocal $2^{m}-1-i$ reveals the error ${ }^{136}$ position. In practice, this computation is performed exploiting the iterative ${ }^{137}$ relation:

$$
\begin{equation*}
\lambda\left(\alpha^{j+1}\right)=\lambda_{0}+\sum_{k=1}^{t-1}\left[\lambda_{k}\left(\alpha^{j}\right)^{k}\right] \alpha^{k} \tag{4}
\end{equation*}
$$

Several publications proposed optimized hardware implementations of 139 BCH codecs with fixed correction capability [? ? ? ? ? ? ]. However, ${ }_{140}$
to the best of our knowledge, only Chen et al. proposed a solution allowing ${ }^{14}$ limited adaptation by extending a standard BCH codec implementation [? ${ }^{142}$ ]. One of the main contributions of Chen et al. is a Programmable Parallel ${ }_{143}$ Linear Feedback Shift Register (PPLFSR), whose generic architecture is re- ${ }_{144}$ ported in Fig. ??. It enables to dynamically change the generator polynomial ${ }_{145}$ of the LFSR. This is a key feature in the implementation of an adaptable tyo BCH encoder.

Fisyre 1: Architecture of a $r$-bit PPLFSR with $s$-bit parallelism.
The gray box of Fig. ?? highlights the basic adaptable block of this ${ }^{148}$ circuit. It exploits a multiplexer, controlled by one of the coefficients of the ${ }_{14}$ desired divisor polynomial, to dynamically insert an XOR gate at the output ${ }_{150}$ of one of the related D-type flip-flops composing the register. The $s$ vertical ${ }_{151}$ stages of the circuit implement the parallelism of the PPLFSR computing ${ }_{152}$ the state at clock cycle $i+s$, based on the state at cycle $i$. However, this ${ }_{153}$
solution has high overhead. In fact such PPLFSR is able to divide by all possible $r$-bit polynomials, while just well selected divisor polynomials are required.

Although Chen at al. deeply analyze the encoding process and the issues related to the storage of parity bits, the deceading process is scarcely ${ }^{158}$ analyzed, without providing details on how adantability is achieved. Four tisg different correction modes, namely $t=(9,14,19,24)$ are considered in) ? 160 for a BCH code defined on $G F\left(2^{13}\right)$ with a block size of 51 (2B) every $2 \mathrm{~KB}{ }_{161}$ page of the flash is split in four bitacks. The selection of the 4 nodes is based 162 on considerations about the nomber of parity bits \&ostore. However, there 163 is no provision to understand hether additionar modes can be easily imple- ${ }_{164}$ mented. As an whinge when selgcting correction modes in which the size 165 of the codendadis I1st a multios of fine parallelism of the decoder, alignment 166 problems arise, which are completely neglected in thepoaper. 167

## 3. Optimized Ancwitectures of Progranmable Parallel LFSRs

In this section, 少e will introduce an optimized block to perform an adapt- 169 able remainder computation. In fact, one of the most recurring operations 170 in 5 H H encoding/decoding is the remainder computation between a poly- ${ }_{171}$ nomhial representing a message to encode/decode and a generator/minimal 172 polynomial of the code, that depends on the selected correction capability. ${ }^{173}$ The PPLFSR of Fig. ?? can perform this operation [? ].

A $r$-bit PPLFSR can potentially divide by any $r$-bit polynomial by prop- ${ }^{175}$ erly controlling its configuration signals $\left(g_{0} \ldots g_{r-1}\right)$. However, in BCH en- ${ }_{176}$ coding/decoding, even considering an adaptable codec, just well selected divi- 177
sor polynomials are required (e.g., the generators polynomials $g_{9}(x), g_{14}(x), \quad{ }_{178}$ $g_{19}(x), g_{24}(x)$ of the four implemented correction modes of [?]). This com- ${ }_{179}$ putational block is therefore highly inefficient. Moreover, the set of divisor ${ }_{180}$ polynomials required in a BCH codec usually share common terms among ${ }^{181}$ each other. Such terms can be exploited to generatean optimized PPLFSR ${ }_{182}$ (OPPLFSR) architecture.

Let us consider, as an example, the design of-a $r=15$-bit programable 184 LFSR able to divide by two polynomidis $p_{1}(x)=x^{15}+x^{13}+x^{10}\left(+x^{5}+x^{3}+x+1 \quad 185\right.$ and $\left.p_{2}(x)=x^{13}+x^{12}+x^{10}+x^{5}+x^{4}+x^{2}+x\right\rangle+x+1$ using as $s=8$-bit parallelism. $\quad 186$

A traditional PPFLSR implementation would Feciires $15 \times 8=120$ gray ${ }^{187}$
 LFSR could divide by any $2^{15}=32,768$ possible 15 -bit polynomials, even if 189 just 2 pelynmiais (i.e., the 0.00 gerof its fall potential) are required. 190

An anatysis of the target divisor polynomials canbe exploited to optimize 191 the PPLFSR architecture. Table ?? reports the binary representation of the 192 two polynomizis.

Looking at Table? ? three categories of polynomial terms can be identi- 199 195
fied

1. ©anmmon terms (represented in bold), i.e., terms defined in all considered196 polynomials ( $x^{13}, x^{10}, x^{5}, x^{3}, x$, and 1 in Table ??). For these terms, ${ }^{197}$ an XOR will be always required in the PPLFSR, thus saving the area ${ }^{198}$ dedicated to the MUX and the related control logic.
2. Missing terms (represented in underlined italic zeros), i.e., terms not 200 defined in any of the considered polynomials, $\left(x^{14}, x^{11}, x^{9}, x^{8}, x^{7}\right.$ and ${ }_{201}$ $x^{6}$ in Table ??). For these terms both the XOR and the related MUX ${ }^{202}$

can be avoided.
3. Specific terms, i.e., terms that are specific of a subset of the considered 204 polynomials $\left(x^{15}, x^{12}, x^{4}, x^{2}\right.$ in Table ??). These terms are the only 205 ones actually required.

We can therefore implement an optimized programmable LFSR (QR ${ }_{207}^{207}$ PLFSR) with three main building blocks: $\diamond$, ?

1. each common present term (i.e. pohmons of all " 1 " of (able ?) needs 209 an XOR, only;
2. each common absentaterm (i.e, columns of all (8) Of Table ??) needs 211 neither XOR nor mux:
3. each specife termbas a gray box, as Fig. ??;

(a) PPLFSR

(b) OPPLFSR

Fiqure 2: Example of the resulting PPLFSR (a) and OPPLFSR (b) with 8-bit parallelism for $x^{15}, x^{14}$ and $x^{13}$ of $p_{1}(x)$ and $p_{2}(x)$

This optimization also applies on polynomials with very different lengths. 215 As an example, an OPPLFSR with single bit parallelism and able to divide ${ }_{216}$ by $p_{1}(x)=x^{225}+x+1$ and $p_{2}(x)=x+1$, would only require a single ${ }_{217}$ adaptable block, compared to the 226 blocks required by a normal PPLFSR. ${ }^{218}$

Furthermore, the advantage of the OPPLFSR increases with the parallelism 219 of the block. In fact, with the same 2 polynomials, a 8-bit OPPLFSR would 220 require 8 adaptable blocks compared to $226 \times 8=1,808$ adaptable blocks of ${ }_{221}$ a traditional PPLFSR.

For sake of generality, Fig. ?? shows the high level architecture of a ${ }^{223}$ generic OPPLFSR. Such a block is able to divide by as et $p_{1}(x), \ldots, p_{M} \lambda(x)$ 2g4 of polynomials. We denote with $q$ the number of required grayhoxes. ${ }_{225}$


The OPRLFSR interface includes: as ion input port (b) used to feed the data, a $\left\lceil\log _{2}(\mathrm{M})\right\rceil$-bit input port (eel) used to select the polynomial of ${ }^{227}$ th a division, and a $s$-bit port (o) providing the result of the division. Two ${ }^{228}$ blpeks compose the OPPLFSR: OPPLFSR $R_{n e t}$ and $R O M$. The OPPLFSR ${ }_{n e t}{ }^{229}$ represents the complete network, partially shown in the example of Fig. ??. ${ }^{230}$ Given the output of the ROM, the q-bit signal $g$ controls the MUXs of the ${ }^{23}$ q gray boxes (Fig. ??) according to the selected polynomial. The ROM is ${ }_{232}$ optimized accordingly with the design of the OPPLFSR, which leads to a ${ }^{23}$ reduced ROM and to a lower area overhead w.r.t. a full PPLFSR.

## 4. BCH Code Design Optimization

In this section, we address first the issue of choosing the most suitable set of polynomials for an optimized adaptable BCH code. Then, we propose a novel block, shared between the adaptable BCH encoder and the decoder, which reduces the area overhead of the resulting codec core.

### 4.1. The choice of the set of polynomials

The optimization offered by the QPPLFSR introduced in Cention??, may 241 become ineffective if not proper exploited. It depends on the number and ${ }_{242}$ on the terms of the sharedinisor ioslynomials implementy in the block. As 243 an example, an excessipe number of shared polynonlals may make it difficult 244 to find compon terms, leading to an unwilled increase of the area overhead. 245 Therefore the chole of the polviomials to share is critical and must be properly tailored to the oralldesigi. ${ }_{247}^{247}$

Let us denote bys the set of $t$ generators grxp ant minimal polynomials ${ }_{24}^{248}$ $\psi_{i}$ which fully haracterize an adaptableBCH code (see Section ??). Since 249 for $G F\left(2^{2 n}\right)$ several primitive polynomiats $\psi_{i}(x)$ can be used to define the ${ }_{250}$ code, several $\Omega$ et $\Omega_{i}$ can be constructed. Choosing the most suitable set $\Omega_{i}$ is ${ }_{251}$ dr $\operatorname{ical}$ to obtain an effective design of the OPPLFSR. On the one hand, it ${ }^{252}$ can be shown that the complexity of $\Omega_{i}$ increases with $m$ [? ? ? ]. On the ${ }^{253}$ other hand, the current trend is to adopt BCH codes with high values of $m{ }^{254}$ (e.g., $G F\left(2^{15}\right)$ ) because current flash devices features a worse bit error rate $\left[?{ }^{255}\right.$ ]. Therefore, a simple visual inspection of each set $\Omega_{i}$ is not feasible to find ${ }^{256}$ the most suitable set of polynomials. An algorithmic approach is therefore ${ }^{257}$ mandatory.

Each set $\Omega_{i}$ can be classified resorting to a Maximum Correlation Index ${ }^{259}$ (MCI). We define as $\operatorname{MCI}\left(p_{1}, p_{2}, \ldots, p_{N}\right)$ the maximum number of common ${ }_{260}$ terms shared by a generic set of polynomials $p_{1}, p_{2}, \ldots, p_{N}$. As an example, ${ }^{261}$ the polynomials of Table ?? have $M C I\left(p_{1}, p_{2}\right)=12$. ${ }_{262}$

In the sequel, we introduce an algorithm to assess each set $\Omega_{i}$ according ${ }_{263}$ to its MCI. Given $i=\{1, \ldots, Y\}$, for each set $\Omega$

1. consider $\Omega_{i}=\left\{p_{1}, \ldots, p_{N}\right\}$ and $v_{0}=p_{i}$
2. determine the polynomial $p_{h}$ sucin that the partition $S$ ( $5 v_{0}>p_{h}$ ) has ${ }_{266}$ the maximum $M C I\left(v_{0}, p_{h}\right.$ cwhere $\left\{=\{1, \ldots, N\rangle\right.$ and $p_{h} \neq v_{0} ; \quad{ }_{267}$
3. determine the polynemial $p_{k}$ such that the partition $\delta_{i, 1}=\left(\left(v_{0}, p_{h}\right), p_{k}\right){ }^{268}$ has the maximung $\left.\cap \mathcal{L} \circ_{0}, p_{h}, p_{k}\right)$ where $k=\{1, \ldots, N\}$ and $p_{k} \neq p_{h} \neq{ }_{269}$ $v_{0}$
4. repeat step 3 until all poiynonials hare been considefed in the partition 271 $S_{i, 1} ;$
5. change the stapting oslynomial to the ne wore, eng., $v_{0}=p_{2}$, considering

6. wherin $v_{0}=p_{N}$ Consider the next set $S_{L_{i+1}}$;

The algorithm ends when all sets $\Omega_{i}$ have been analyzed. For each $\Omega_{i},{ }^{276}$ the sutput is a set of partitions:

$$
\begin{equation*}
S_{i, j}=\left\{S_{i, 1}, S_{i, 2}, \ldots, S_{i, N}\right\} \tag{5}
\end{equation*}
$$

Fig. ?? graphically shows the MCI of two partitions generated from two ${ }^{278}$ different starting points, for an hypothetical set $\Omega_{i}$.

Fig. ?? shows that MCI always has a decreasing trend with the size of 280 the partition $S$. This is straightforward since adding a polynomial may only ${ }_{281}$


Fiqure 4: Mcrexamples of two hypothetical partitions $S_{i, 1}$ and $S_{i, 2}$
decrease or keep constant the current value of MCI. The curves, reported in ? ar critical in the choice of the most suitable set of polynomials for an optimized BCH code. For each partition $S_{i, j}$ with $j=\{1 \ldots N\}$, we can ${ }^{284}$ compute the average MCI ( $M C I_{\text {avg }}$ ) as:

$$
\begin{equation*}
M C I_{a v g}\left(S_{i, j}\right)=\frac{1}{N} \sum_{l=1}^{N-1} M C I_{l} \tag{6}
\end{equation*}
$$

Eq. ?? applies to each set $\Omega_{i}$ where $i=\{1 \ldots Y\}$.
The best partition of the set $\Omega_{i}$ is then computed selecting the one with ${ }_{287}$
maximum $M C I_{\text {avg }}$ :

$$
\begin{equation*}
S_{\text {best }_{i}}=\underset{j}{\operatorname{argmax}}\left[M C I_{\text {avg }}\left(S_{i, j}\right)\right] \tag{7}
\end{equation*}
$$

Finally, Eq. ?? compares the best partition of each set $\Omega_{i}$ to find the best ${ }_{289}$ set of polynomials:


Eq. ?? defines the family of pqinnomials $S_{\text {best } B C H}$, rwith tine maximum ${ }_{291}$


Let us provide an example to support the understanding of the algorithm. ${ }^{293}$ Suppose to consider a single set $\Omega_{i}$ composed of the polynomials of Table ??. 294 The steps of the algorithm are:

1. Let us start with $v_{0}=p_{1}$
2. We first evaluates $\operatorname{MCI}\left(p_{1}, p_{2}\right)=3, M C I\left(p_{1}, p_{3}\right)=4, M C I\left(p_{1}, p_{4}\right)={ }^{297}$ 3. Since $M C I\left(p_{1}, p_{3}\right)=4$ is the maximum, the resulting partition is 298 $S_{i, 1}=\left\{p_{1}, p_{3}\right\} \quad{ }_{299}$
3. The next step considers $\operatorname{MCI}\left(\left(p_{1}, p_{3}\right), p_{2}\right)=3$ and $\operatorname{MCI}\left(\left(p_{1}, p_{3}\right), p_{4}\right)=300$ 3. It is straightforward that the choice of either $p_{2}$ or $p_{4}$ does not affect ${ }_{301}$ the final value of the $M C I_{\text {avg }}$.


Given $\Omega_{i}$ with starting point $p_{1}$, it cann be shounn that the finalpartition ${ }_{303}$ is $S_{i, 1}=\left\{\left(\left(p_{1}, p_{3}\right), p_{4}\right), p_{2}\right\}$ with a $r\left(C I_{a y g} \Rightarrow(4+3+3) / 4=2 f\right.$ fiom $\mathrm{Eq} . ? ? . \quad 304$

The complete algorithm iteratesthis computationtry ak possible starting 305 points. Fig. ?? graphicallyshons the output ofthe MCI associated with each 306 partition $S_{i, j}$ calculated forl the followingstarting point $j=\{1,2,3,4\}$. $\quad 307$


Figure 5: The MCI Trend of Table ??

According to Eq. ??, $S_{i, 2}$ (the bold line) is the $S_{\text {best }_{i}}$ of the example of 308

Table ??, with a $M C I_{\text {avg }}\left(S_{i, j}\right)=4$.

### 4.2. Shared Optimized Programmable Parallel LFSRs

Let us assume to design an adaptable BCH code with correction capability ${ }_{311}$ from 1 up to $t_{M}$. Such a code needs to compute remainders of the division ${ }_{312}$ of:


- the message $m(x)$ by (potentially) ali genemator polynomials from $g_{1} \quad 314$ up to $g_{t_{M}}$, for the encoding (???;
- the codeword $c(x)$ by (pptenitially) all minimar nomials from $\psi_{1}(x){ }_{316}$ up to $\psi_{2 t_{M}-1}(x)$ to comprete the set of sympromes required during the ${ }_{317}$ decoding phase. ${ }_{318}$

In traditiphai implemeritation, these computations are performed by 319 two separate set of LESRAs. In this paper, we oropose to devise a shared 320 set of LFSRs ableto: (i) iperform all these compytations, and (ii) reduce the ${ }_{321}$ overall cost in terms of persources overhead. Therefore, we can adopt the same ${ }_{32}$ shared set of LFSW both in the encoding and decoding processes. This is ${ }^{323}$ possible since in a flash memory these operations are, in general, not required ${ }^{324}$ at $\boldsymbol{y}$ 少 some time.

The OPPLFSR, introduced in Section ??, is the main building block of ${ }_{326}$ the set of shared LFSRs. Therefore, we will refer hereafter to such set of ${ }^{327}$ LFSRs as shared OPPLFSR (shOPPLFSR). Fig. ?? shows the high-level ${ }^{328}$ architecture of the shOPPLFSR. Its interface includes: a $s$-bit input port ${ }_{329}$ (IN) used to input the data to be divided, a $\left\lceil\log _{2}(\mathrm{~N})\right\rceil$-bit input port (en) ${ }_{330}$ used to enable each OPPLFSR, an input port (sel) used to select the proper ${ }_{331}$
polynomial by which each OPPLFSR has to divide, and a $\mathrm{N} \times$ s-bit port (p) ${ }_{332}$ providing the result of the division.


Figure 6: The shOPPLFSR aschitecture is composed by maitiple OPPLFSRs

Given $\mathrm{MO} P \mathrm{PFSB}$ and a zraximum correction capability $t_{M}$, each ${ }_{33}$ OPPLESR i periotms the divispm by a setot generator pofynomials $g(x)$ and 335 minimal polyromials $\psi$ (r). Sưch shoPPLFSR candoe seen as an optimized ${ }_{336}$ programmable LFSR abvéto:

- divide oy all generator polynomialsfremn $g_{1}(x)$ to $g_{t_{M}}(x)$;
- divide by specific subsets of minimal polynomials from Eq. ??, as well. ${ }_{339}$

An improper choice of the shared polynomials $g(x)$ and $\psi(x)$ can dramat- ${ }^{340}$ ically reduce the performance of the overall BCH codec. Also the partitioning ${ }^{341}$ strategy adopted is critical to maximize the optimization in terms of area, ${ }^{342}$ minimizing the impact on the latency of encoding/decoding operations. ${ }^{343}$

The algorithm presented in Section ?? provides a valuable support for the ${ }_{344}$ exploration of this huge design space. In fact, the proposed method can be ${ }_{345}$ exploited to properly partition polynomials into the different OPPFLSRs of ${ }^{346}$

Fig. ??, in order to maximize the optimization of the resulting shOPPFLSR. ${ }^{347}$ Such optimization should not be obtained following blindly the outcomes of 348 the algorithm, but always tailoring them to the specific design. Regarding ${ }_{349}$ this topic, Section ?? provides more details about our experimental setup ${ }_{350}$ and the related experimental results.

## 5. Adaptable BCH Encoder

In this section, we propose an adatable BCH encoder winien expoits the

[^0] shOPPLFSR of Section ??. According to the BCH theory theshOPPLFSR 354 of Fig. ?? is a very efficient circuit to perform the compration expressed in 355 Eq. ??. However, intthe encoding phase, the message $m(x)$ must be multi- ${ }_{356}$ plied by $x^{r}$ ieforecgalccuating the yeminder of the division by $g(x)$ (see Eq. ${ }^{357}$ ??). This can be outained without significant modificati申ns of the architec- ${ }_{358}$ ture of sh $Q P$ PFLSR. It is enough to input the bit 5 Of the message directly ${ }_{359}$ in the most signiffant 1 it of the LFSR, instead than starting from least 360 significant bif. Fig. ? ? shows the highevel architecture of the adaptable ${ }_{361}$ encodery 362

The encoder's interface includes: a $s$-bit input port (IN) used to input the ${ }^{363}$ A- message to encode starting from the most significant bits, a $\left\lceil\log _{2}\left(t_{M}\right)\right\rceil-\quad 364$ bit input port ( t ) selecting the requested correction capability in a range 365 between 1 and $t_{M}$, a start input signal used to start the encoding process ${ }_{366}$ and a $s$-bit output port (OUT) providing the $r$ parity bits. Three blocks ${ }_{367}$ compose the encoder: a shOPPLFSR, a flush logic and a controller. 368

The shOPPLFSR performs the actual parity bits computation. Accord- 369 ing to the BCH theory, adaptation is achieved by supporting the computation ${ }_{370}$
 building blocks and their main conneetions.
of remainders with generator polynomials, one for each value $t$ may as- ${ }^{371}$ sume. The contwor achieves thisyask in two steps: (i) enabling the proper ${ }_{372}$ OPPLFSR through the len Sisnl, and (ii) selecting the proper polynomial ${ }^{373}$ through the isel signat accerding to the desisdogrection capability $t$. 374 Then, it manages the dye eall encoding procespasen two internal param- ${ }^{375}$ eters: 1) the number of s-bit words cornoosirrs the message (fixed at design ${ }_{376}$ time) ard 2) the nymber of produced $s$-bit parity words, that depends on 377 the selected qorrection capability. The flush logic splits the $r$ parity bits into ${ }_{378}$ $s-10$ mords, providing them in output, one per clock cycle. ${ }^{379}$

To further optimize the encoding and the decoding process, since in a flash 380 memory these operations are not required at the same time, the encoder's ${ }_{381}$ shOPPLFSR can be merged with the shOPPLFSRs that will be employed ${ }_{382}$ in the syndrome computation (see Section ??), thus allowing additional area ${ }_{383}$ saving. 384

## 6. Adaptable BCH Decoder

Fig. ?? presents the high-level architecture of the proposed adaptable ${ }_{386}$ decoder. The decoder's interface includes: a $s$-bit input port (IN) used to 387 input the $n$-bit codeword to decode (starting from the most significant bits), 388 a $\left\lceil\log _{2}\left(t_{M}\right)\right\rceil$-bit input port ( t ) to select the desired correction capability a 389 start input signal to start the decoding a, et of output ports providing 390 information about detected errors. In partoricular.
 have been detected in codeword. In case pf decoding failure it is set ${ }_{393}$ to 0 ;


394

- erraca an armask provide information about the detected error positipns. Assuming the codeword split into $h$-bitwords, erred is used as a word address in the codeword and errmask is a f bit mask whose ${ }^{397}$ asserted bits indicate detected erronequspits in the addressed word. 398 The parallelism haaf the error mass demerits on the parallelism of the
- fail is asserted whenever an error occurred during the decoding process (e.g., the number of errors is greater than the selected correction 404 capability); 405
- end is asserted when the decoding process is completed.


The full decoder therefore includes four main blocks: (1) the Adapt- ${ }^{407}$ able Syndrome Machine, computing the syndromes of the codeword, (2) 408 the Adaptable inversion-less Berlekamp Massey (iBM) Machine, that elabo- 409 rates the syndromes to produce the error locator polynomial, (3) the Adapt- ${ }_{410}$ able Chien Search Machine in charge of searching far the error positions, and ${ }_{411}$ (4) the Controller coordinating the overall decoding pages.

### 6.1. Adaptable Syndrome Machine



Fig. ?? shows the high-level architecture of the proposed adaptable syn- ${ }_{414}$ drome machine with correction capability $1 \leqslant t \leqslant t \backsim{ }_{415}$


Figure 9: Architecture of the adaptable Syndrome Machine

According to Section ??, remainders can be calculated by a set of Parallel ${ }_{416}$ LFSRs (PLFSRs) whose architecture is similar to the one of the PPLFSR ${ }_{417}$ of Fig. ??, with the only difference that the characteristic polynomial is 418 fixed (XOR gates are inserted only where needed, without multiplexers). ${ }_{419}$

Each PLFSR computes the remainder of the division of the codeword by a different minimal polynomial $\psi_{i}(x)$. Given two correction capabilities $t_{1}$ and ${ }_{421}$ $t_{2}$ with $t_{1}<t_{2} \leq t_{M}$, the set of $2 t_{1}$ minimal polynomials generating the code ${ }_{422}$ for $t_{1}$ is a subset of those generating the code for $t_{2}$. To obtain adaptability ${ }_{423}$ of the correction capability in a range between ${ }^{424}$ machine can therefore be designed to compute the noximum number to $M=5$ of remainders required to obtain $2 t_{M}$ syndromes. Based on the selected ${ }_{426}$ correction capability $t$, only the first 1 PLFSRS out of the $t_{M}$ available in the 427 circuit are actually enabled thrø名g the enable div. network of tig. ??. 428

A full parallel syndrome calctiator, including th PSFSRs, requires a 429 considerable amount 9 resoutbes that are undentilized in the early stages ${ }_{430}$ of the flash lifetingwhen reduced grrection capability is required. To opti- ${ }^{431}$ mize the adaptable syndromy machine arl to trade-off between complexity ${ }_{432}$ and performance, we exploit the shoPPLFSR introduged n Section ??. The ${ }^{43}$ architecture proposed in Fig. ?? includes tyosen of LFSRs for remainder ${ }_{434}$ computation: (i) ensentional PLFSRs. and (ii) shOPPLFSR. Conventional ${ }^{435}$ PLFSRsare exploited for parallel fast computation of low order syndromes ${ }^{436}$ required when the requested correction capability is below a given threshold. ${ }^{437}$ ShPPRLFSR is designed to divide for selected groups of minimal polynomials ${ }^{438}$ nqt covered by the fixed PPLFSRs. It represents a shared resource utilized ${ }_{439}$ when the requested correction capability increases. It enables area reduction ${ }_{440}$ at the cost of a certain time overhead. The architectural design, chosen for ${ }_{441}$ the fixed PLFSRs and the OPPLFSR, enables to trade-off hardware com- ${ }_{442}$ plexity and decoding time, as it will be discussed in Section ??. ${ }_{443}$

It is worth to mention here that the parallel architecture of the PLFSR, 444

coupled with the adaptability the code, introduces a set of additional 445 word alignment problemg that nirist be addressed to edrrectly adapt the ${ }_{446}$ syndrome calculationtodifferent values of $t$. The syndrome machine receives 447 the codeword in yodrds of $s$ bits, starting from the most significant word. 448 When tire numberof parity bits does notallow to align the codeword to the 449 parallelisins, the unused oits of the hast word aref fited with 0 . To correctly 450 compute each synyome, the parity bit $r_{0}$ ofthe codeword must enter the ${ }_{451}$ least significat bit of pach LFSR. The aligner block of Fig. ?? assures ${ }_{452}$ this condrtibn by pyoperly right-shifting the codeword while it is input into ${ }_{453}$ the syndromg machine. Let us consider the following example: $k=2 \mathrm{~KB}$, ${ }^{454}$ $m-15 t=2, s=8$ and therefore $r=m \cdot t=30$. Since 30 is not multiple of ${ }_{455}$ $s=8$, the codeword is filled with two zeros and $\mathrm{p}_{0}$ is saved in position 2 of ${ }^{456}$ the last byte of the codeword $\left(m_{2047} m_{2046} \ldots m_{1} m_{0} p_{29} p_{28} \ldots p_{1} p_{0} 00\right)$. In this case ${ }_{457}$ the PLFSRs require a 2-bit alignment, implemented by the network of Fig. ${ }^{458}$ ??. It simply delays the last 2 input bits resorting to two flip-flops, whose ${ }_{459}$ initial state has to be zero, and properly rotates the remaining input bits. 460 Changing the correction capability of the decoder changes the number of ${ }_{461}$
parity bits of the codeword, and therefore the required alignment. Given the ${ }_{462}$ parallelism $s$ of the decoder, a maximum of $s$ alignments must be provided ${ }_{463}$ and implemented in the Aligner block of Fig. ??.

With the proper alignment, the PLFSRs can perform the correct division and the evaluators can provide the required syndrames. The evaluators are ${ }^{466}$ simple combinational networks involving XOR onerations, according to the thy Galois Fields theory (the reader may refer t 0 ? ${ }^{2}$ for specific implernentation 468 details).

### 6.2. Adaptable Berlekamp Masseytmachine

In our adaptable codec we implemented the inversion-less Berlekamp- ${ }^{471}$ Massey (iBM) algarithm proposed in ?Thich is able to compute the error locator polynamiáa 0 a in $t$ itarations 473

The nian steps of the computation are reported in Ald. ??. At iteration 474 $i$ (rows 2 to 12), the alopithm finds an error locator D dynomial $\lambda(x)$ whose 475 coefficients solye the first $i$ equations of ? ? (1) (4). It then tests if the 476 same polynomial solves also $i+1$ equatrons)(row 5). If not, it computes a 477 discrepancy terill so that $\lambda(x)+\delta$ solves the first $i+1$ equations (row 9). ${ }^{478}$ This iteratife process is repeated until all equations are solved. If, at the encof the iterations, the computed polynomial has a degree lower than $t$, it correctly represents the error locator polynomial and its degree represents the number of detected errors; otherwise, the code is unable to correct the given codeword.

The architecture of the iBM machine is intrinsically adaptive as long as ${ }_{48}$ one guarantees that the internal buffers and the hardware structures are sized ${ }_{485}$ to deal with the worst case design (i.e., $t=t_{M}$ ). The coefficients of $\lambda(x)$ are ${ }_{486}$
Algorithm 1 Inversion-less Berlekamp-Massey alg.
$\lambda(x)=1, k(x)=1, \delta=1$
for $\mathrm{i}=0$ to $\mathrm{t}-1$ do
$d=\sum_{j=1}^{t}\left(\lambda_{j} \cdot S_{2 i-j}\right)$
$\lambda(x)=\delta \lambda(x)+d \cdot x \cdot k(x)$
if $d=0$ OR $\operatorname{Deg}(\lambda(x))>i$ then
$k(x)=x^{2} \cdot k(x)$
else
$k(x)=x \cdot k(x)$
$\delta=d$
end if
$\mathrm{i}=\mathrm{i}+1$
end for
if $\operatorname{Deg}(\lambda(x))<t$ then
output $\lambda(x), B \cap(x, x)$
else output NAILURE
end if
-bit registers whose rumber depends on the correction capability. In the worst qase, upta cor coefficients must be stored for each polynomial. 488 The adart able iBM machine therefore includes two $m$-bit register files wiry $\hat{N}_{\mathrm{N}}$ registers to store these coefficients. Whenever the requested correction capability is lower than $t_{M}$ some of the registers will remain unused. The 491 number of multiplications performed during the computations also depends on $t$. Row 3 requires $t$ multiplications, while row 4 requires $t$ multiplications 49 to compute $\delta \lambda_{i}(x)$ and $t$ multiplications to compute $d \cdot x \cdot k(x)$
We implemented a serial iBM Machine including 3 multipliers for $\operatorname{GF}\left(2^{m}\right){ }_{495}$ to perform multiplications of rows 3 and 4 . It can perform each iteration of 496
the iBM algorithm in $2 t$ clock cycles ( $t$ cycles for row 3 and $t$ cycles for ${ }_{497}$ row 4) achieving a time complexity of $2 t^{2}$ clock cycles. This implementation ${ }_{498}$ is a good compromise between performance and hardware complexity. An 499 input $t$ dynamically sets the number of iterations of the algorithm, thus 500 implementing the adaptation.

### 6.3. Adaptable Chien Machine

The overall architecture of the proposed adaystable Chien Machmine is ${ }^{503}$ shown in the Fig. ??. The machin efirst loads into $t_{M}$ ri-bitpgisters the 504 coefficients from $\lambda_{1}$ to $\lambda_{t_{M}}$ ofthe-ery $g$ locator polynomial $\lambda(x)$ computed by ${ }_{505}$ the iBM machine ( 1 d - 0 . The actual search is then started $\left(1 \mathrm{~d}=1\right.$ ). At ${ }_{506}$ each clock cycle, the biockperforms paralizl evaluations of $\lambda(x)$ in $\mathrm{GF}\left(2^{m}\right)$ 507
 sponds to onte of the $h$ carlinate envoy locations thatinaty been evaluated. 509 Asserted bits denote detected entors. This mask can then be XORed (outside 510 the Chien Machine with the related bits of the ogdeword in order to correct 511 the detected erronedus bits.

The architectine of Fig. ?? provides an adaptable Chien machine with ${ }^{513}$ Omer area Consumption than other designs [? ], having, at the same time, ${ }_{514}$ a marginal impact on performance. Four interesting features contribute to 515 such optimization: (i) constant multipliers substructure sharing, (ii) adapt- 516 ability to the correction capability, (iii) improved fast skipping to reduce the ${ }_{517}$ decoding time, and (iv) reduced full GF multipliers area. In the sequel, we 518 briefly address each feature.

The first feature is represented by the optimized GF Constant Multipliers ${ }_{520}$ (optGFCM) networks of Fig. ??. The $h$ parallel evaluations are based on ${ }^{521}$


Figure 11: Architecture of the proposed parallel adaptable ahien Machme with parallelism equal to $h$

equation (??), Phenorst case $\left.\left(t^{2}\right)=A x\right)$, the parallel evaluation of equation ${ }_{522}$ (??) requires a matrix of $t \times x$ constant Galois multipliers. They multiply the content of the ing registers by $\alpha, \alpha^{2}, \ldots, a^{2}$, fespectively. However, 524 we can note that/each corumn of constant aF nuthpliers shares the same ${ }_{525}$ multiplicand. Thenefore. we can iteratively group their best-matching com- ${ }^{526}$ binations [? ints the $t_{M}$ optGFCM networks of Fig. ??. Such optGFCMs ${ }_{527}$ provide up to $60 \%$ reduction of the hardware complexity of the machine with ${ }_{528}$ nd $\rightarrow$ nest on performance.
The second feature is the adaptability of the Chien machine. The rows of ${ }_{530}$ the matrix define the parallelism of the block (i.e., the number of evaluations ${ }_{531}$ per clock cycles), while the columns define the maximum correction capability ${ }_{532}$ of the block. Whenever the selected correction capability $t$ is lower than $t_{M},{ }_{533}$ the coefficients of the error locator polynomial of degree greater than $t$ are ${ }^{534}$ equal to zero and do not contribute to equation (??), thus allowing us to ${ }_{535}$
adapt the computation to the different correction capabilities.
The third feature stems from a simple observation. Depending on the selected correction capability $t$, not all the elements of $\mathrm{GF}\left(2^{m}\right)$ represent ${ }_{538}$ realistic error locations. In fact, considering a codeword composed of $k$ bits ${ }_{539}$ of the original message and $r=m \cdot t$ parity bits, only $k+m \cdot t$ out of $2^{m}{ }^{540}$ elements of the Galois field represent realistic eprorlacations. Given that an error location $L$ is the inverse of the related $C F$ element $(L=2$ ) $-i$ the elements of $\operatorname{GF}\left(2^{m}\right)$ in which the erroryocator polynomial mote be evaluated are in the following range:


All elements between $\alpha^{0}$ and $\alpha z^{m-A-m t}$ can be skipped to reduce the computation time. Differently from liked correctioncapability fast skipping ${ }_{546}$ Chian machines this interval is not constant here but depends on the se- ${ }^{547}$ lected $t$. The archive texture of Fig. ?? implements an adaptable fast skipping 548 by initializing the internal registers to the Coefficients of the error corrector ${ }_{549}$ polynomial multiplied by a proper value $\beta_{i n i}^{t}=\alpha^{2^{m}-k-m \cdot t-1}$. For each value ${ }_{550}$ Of t. the mit constant values corresponding to $\beta_{i n i}^{t},\left(\beta_{i n i}^{t}\right)^{2}, \ldots,\left(\beta_{\text {init }}^{t}\right)^{t_{M}}{ }^{551}$ must be stored in an internal ROM (not shown in Fig. ??) and multiplied ${ }_{552}$ by the coefficients $\lambda_{i}$ using a full GF multiplier. ${ }_{553}$

This is connected with the last feature, the reduced GF Full Multipliers ${ }^{554}$ (redGFFM) network of Fig. ??. Each full GF multiplier has a high cost in ${ }_{555}$ terms of area. Since they are used only during initialization of the Chen, the ${ }_{556}$ redGFFM adopts only $z \leqslant t_{M}$ full GF multipliers. It also includes a $(\lambda)$ input ${ }_{557}$
port to input $z$ coefficients, per clock cycles, of the error locator polynomial. ${ }_{558}$ This network enables to reduce area consumption, at a reasonable cost in ${ }_{559}$ terms of latency. 560

For the sake of brevity, a detailed description of the controller required ${ }_{561}$ to fully coordinate the decoder's modules interaction is out of the scope of 562 this paper.

## 7. Experimental Results

This section provides expermental प̌ata from the impiementation of the ${ }_{565}$ adaptable BCH codec proposed ora selected caselstudy.)

To cope with (tO , eomplexit of mannen design of these blocks, a semi- 568 automatic generation tool hamed ADAGE (ADaptive ECC Automatic GEn- ${ }_{569}$ erator) [? Nable to generate a fully synthesizable adaptable BCH codec core 570 following the propesed ardhitecture has been desibned and exploited in this 571 experimentation exterling a preliminact frymework previously introduced 572 in [? ]. The dveran architecture of the framework is in Fig. ??. ${ }_{573}$ The code analyzer block represents the first computational step required 574 to select the desired code correction capability based on the Bit Error Rate ${ }_{575}$ (BER) of a page of the selected flash [? ]. The BER is the fraction of er- ${ }_{576}$ roneous bits of the flash. It is the key factor used to select the correction 577 capability. Two values of BER must be considered. The former is the raw ${ }_{578}$ bit error rate (RBER), i.e., the BER before applying the error correction. ${ }^{579}$ It is technology/environment dependent and increases with the aging of the 580 page [? ? ]. The latter is the uncorrectable bit error rate (UBER), i.e., ${ }_{581}$


Figure 12: BCH codec automatic generation framework.
the BER after the application of the ECC, which is application dependent. ${ }_{582}$ It is computed as the probability of having more than $t$ errors in the code- ${ }_{583}$ word (calculated as a binomial distribution of randomly occurred bit errors) ${ }_{584}$ divided by the length of the codeword [?]: 585
 exploited to compute the maximum required correctioncapabilicy of the 587 code and consequently the value of that defines thetarget GF. Given these ${ }_{588}$ two parameters, the Galonis Field manager exploits diristernal polynomials 589 database to generaternet of minimal polynomialsand the related generator 590 polynomialsforthaselected code. 59

Finaly, the RHL VHDI code generator combines theste parameters and 592 generates a RTL descrintion of the BCH encoderand decoder implementing ${ }^{593}$ the architecture iristrated in this paper. $\bigcirc 59$


Experiments have been performed, using as a case study a 2-bit per cell 599 MLC NAND Flash Memory featuring a 45nm manufacturing process de- 600 signed for low-power applications, with page size of 2 KB plus 64 B of spare ${ }_{601}$ cells. The memory has an 8-bit I/O interface. Considering the design of 602 the BCH code, the current trend is to enlarge the block size $k$ over which ${ }_{603}$

ECC operations are performed. In fact, longer blocks better handle higher 604 concentrations of errors, providing more protection while using fewer parity 605 bits [? ]. For this reason, we adopted a block size $k=2 \mathrm{~KB}$, equal to the 606 page size of the selected memory.

Experiments performed on the flash provided that, in a range between 608 10 and 100,000 program/erase (P/E) cycles on a Dage, the estimated RBER कog changes in a range $\left[9 \times 10^{-6} \div 3.5 \times 10^{-4} ? ?\right.$ Nith a target BER of 610 $10^{-13}$, which is typical for commercise applications [? ? according to 611 equation (??) we need to desigs a coded with correction caparosility in the 612 range $t_{\text {min }}=5$ up to $t_{\text {M }} 今 24$. Ssince $k=2^{14}$ anid $t_{\text {人 }}=24$, from the ${ }_{613}$ expression $k+m \cdot t_{M} \Omega^{m}$-1ve deduce $m=15$, thlus obtaining a maximum ${ }_{614}$ of $r=m \cdot t^{2} \simeq 45 B$ of parity information. Given the 8-bit I/O interface of ${ }_{615}$ the memory, bothe whe encodery and fine deroder have been designed with an ${ }_{616}$ input paravielism of $s=s$ bits. The values of $h$ artr 2 or the Chien Machine 617 are a trade-off between the complexity of the decoder and the decoding time. ${ }_{618}$ Given the I/Cparareismof the flash ant thearea optimizations of Fig. ??, ${ }_{619}$ we opted for a Chien machine with paralieism $h=8$ and $z=1$ full GF ${ }_{620}$ multip iers.

621 Ninthis experimentation we analyzed the three architectures summarized ${ }^{622}$ in fable ??. ${ }_{623}^{623}$

Arch. 1 is classic BCH architecture with fixed correction capability of ${ }_{624}$ 24 errors per page. It represents the reference to compare our adaptable ${ }_{625}$ architectures. ${ }_{626}$

Arch. 2 is an adaptable architecture with $t_{\min }=5<t \leq 24$ using ${ }^{627}$ a traditional PPLFSR for the encoder and 24 PLFSRs for the syndrome ${ }_{628}$
calculation. It is worth mentioning here that, differently from what reported ${ }_{62}$ in the previous sections, the minimum required correction capability of the ${ }_{630}$ codec is higher than 1 . This allows us to save space in the encoder PPLFSR ${ }_{631}$ since less polynomials must be stored, and in the Chien Machine's ROM ${ }_{632}$ since less $\beta_{\text {ini }}$ terms must be stored.

Arch. 3 is an optimized version of Arch. 2 exproiting the use of a shOPPLFSR shared between the encoder and the decoder, to traberedf design 635 complexity and decoding time. In graek to optimize the use of thas shOP- ${ }_{636}$ PLFSR, we exploited the algoritimpropsed in Section ? ? Giren our adapt- ${ }^{637}$ able BCH code, a set of ad-hoc Matlab simulation ereriyts implement this ${ }^{638}$ preliminary analysis 1,800 set $\Omega_{i}$ of polynomials. Each set $\Omega_{i}$ contains ${ }_{639}$ $t_{M}-t_{\text {min }}-1=2 \pi$ generator polyngmials required in the encoder and $t_{M}=24 \quad{ }_{640}$ minimad polymomials required in thedeceder. This analypis aimed at finding ${ }_{641}$ the most suitable set of shared gencrator and minimarpolyhomials to trade- ${ }^{642}$ off between decoder's area and latency. A reasonnable trade-off has been ${ }^{643}$ found using 2 shOPRLFSK composed of $=-5$ PPLFSRs, each of which 644 dividing by the following set of polynomians: $\left\{g_{5}, \psi_{29}, \psi_{39}\right\},\left\{g_{6}, \psi_{31}, \psi_{41}\right\},{ }_{645}$ $\left\{g_{7}, \psi_{3}\left\{, \psi_{43}\right\}^{\{ }\left\{g_{8}, \psi_{35}, \psi_{45}\right\}\right.$, and $\left\{g_{9}, \ldots, g_{24}, \psi_{37}, \psi_{47}\right\}$. The reader may refer ${ }_{646}$ to the appendix of this paper for the full list of employed polynomials. All ${ }_{647}$ other structures remain almost unchanged. The comparison between Arch. $1{ }_{648}$ and Arch. 2 enables to highlight the benefits of using an adaptable codec, ${ }_{649}$ while the comparison between Arch. 2 and Arch. 3 shows the advantages of 650 adding optimized shared blocks.

[^1]Table 3: Characteristics of the analyzed architectures


Table ?? summarizes the neain impomentation detain of the three se- ${ }_{653}$ lected architectures in terms of required parity bitt and worst case encod- 654 ing/decoding latency, expressed in terms of clock cycles. ${ }^{655}$

Let us start writh theyevaluation of the amount of redundancy introduced ${ }_{656}$ by the twe architectures. Arg. I which has a fixed correction capability of 24 errons per page, requires to store $m \cdot t_{M} \sqrt{24} 15=360$ parity bits ${ }^{658}$ (about 45B) for eas 2kis page of the flash. Hisp accernts for about $70 \%$ of 659 the full spare area availabie for each pare. Sunce the spare area cannot be ${ }_{660}$ fully resped for storing ECC information (high-level functions, such as file ${ }_{661}$ systern management and wear-leveling need to save considerable amount of 662 in $V$ mation in this area), this percentage represents a considerable overhead ${ }_{663}$ for the selected device. Based on the results of Table ??, Fig. ?? shows how, ${ }_{664}$ for the adaptable codecs of both Arch. 2 and Arch. 3, the percentage of spare ${ }_{665}$ area dedicated for storing parity bits changes with the selected correction ${ }_{666}$ capability. The total occupation ranges in this case from $15 \%$ to $70 \%$ of the ${ }_{667}$ total spare area. This mitigates the overhead for storing parity bits whenever ${ }_{668}$ the error rate enables to select low correction capabilities (e.g., for devices in 669

the early stages of their life).


For all implementations, the encoding latenen devends on the size of the 671 incoming mespage and istherefore constant regardless the adaptability of the 672 encoder (see Table??). The decoding latency is instead influenced by the ${ }^{673}$ correction cappability, as reported in Table ??. Fig. ?? compares the decoding 674 lat acy $n$ the three architectures for each considered correction capability. Results are provided in number of clock cycles. It is worth mentioning here 676 that timing estimations of Table ?? and Fig. ?? depict the worst-case sce- 677 nario in which the Chien Machine must search all possible positions prior to 678 find the detected number of errors. Fig. ?? highlights that, for the lowest 679 correction capability, both Arch. 2 and Arch. 3 enable $22 \%$ of decoding time ${ }_{680}$ reduction when compared to the fixed decoding time of Arch. 1. The decod- ${ }_{681}$
ing time increases with the correction capability. For Arch. 2, it reaches the same level of the fixed architecture when the correction capability reaches ${ }_{68}$ $t=24$. Arch. 3 deviates from this behavior for $t \geqslant 20$. This penalty is intro- ${ }^{684}$ duced by the use of the shOPPLFSR in the Syndrome Machine. In this case, 685 the codec includes 5 blocks to perform remainder camputation with 10 min- ${ }^{686}$ imal polynomials $\left\{\psi_{29}, \psi_{39}, \psi_{31}, \psi_{41}, \psi_{33}, \psi_{43}, \psi_{35}, \psi_{45}, \psi_{47}\right\}$. This implies ${ }^{697}$ doubling the syndrome computation time everytime the required correction 688 capability reaches a level in which allyhese porynomials must be used. Nev- 689 ertheless, we will show that this reduced performance is countroalanced by 690


Figure 14: Worst case decoding latency for the three architectures considered.

### 7.4. Synthesis Results

Synopsys Design Vision and a CORE 45nm technology cell library have ${ }_{69}$ been exploited to synthesize the designs. Table ?? shows the results of the 694
synthesis of the three architectures. The hardware structures required to ${ }_{695}$ obtain the adaptability of the code introduce a certain area overhead. Con- 696 sidering Arch. 2, the area of the encoder increases since 19 generator poly- ${ }^{697}$ nomials must be stored in its ROM, while the area of the decoder increases 698 due both to the aligners in the syndrome machine and to the ROM in the 699 Chien machine to adapt the fast skipping process. Newertheless, the intro- too duced overhead is about $14 \%$ which is still accentarle. Considering Arch. 3, 701 the introduced overhead is halved wry. Arch. 2. Yhe area fethe encoder is 702 almost comparable with Arch. $2>$ Howerer, it now includes thershOPPLFSR 703 and a smaller ROMs which contribete, with the LAASR Bharing, at decreasing 704 the area of the decodef Forbor architectures weobtained a maximum clock 705 frequency of $100 /{ }^{\prime} \mathrm{Hz}$, which confirns that the adaptability does not impact 706 the maximunt speed of the cireqit. This are result is interesting if compared 707 with an estrination of the area for the adaptable grenitecture proposed in $\left[\begin{array}{ll}? & 708\end{array}\right.$ ]. [? ] designed afoded working on blocks of data of 512B, smaller than 709 the 2 KB used in this paner. Given the somenasimum correction capability 710 $\left(t_{M}=2\right.$ 章, $\because\left[\right.$ uses a code defined on CF $\left(2^{13}\right)$ instead of the code defined ${ }_{711}$ on $G F\left(2^{15}\right)$ used in this paper. However, even if the code is simpler and the 712 ny mer of correction modes is smaller (only 4 correction modes), the area of ${ }_{713}$ the codec accounts about 158.9 K equivalent gates $^{2}$, which is higher than the ${ }_{714}$ 111.4 K and the 105.2 K equivalent gates of the Arch. 2 and Arch. 3 proposed. 715

Fig. ?? compares the decoder's dynamic power dissipation of the three ${ }_{716}$ architectures computed using Synopsys PrimeTime. As for the decoding ${ }_{717}$

[^2]Table 5: Synthesis Results

latency the analysis has been perforned for a worstease sinulation in which 718 $t$ errors are injectefy at ond end the codeyong that the Chien Machine 719 must search aisossible positions prior todetectall errors. Considering Arch. 720 2, results how thaty the introduction of the adaptability enables up to $15 \%$ of ${ }_{721}$ dynanaic powgr saving when the lowest correction capability can be selected. 722 $T V$ is diue to the fact that the portions of the circuits not required for low ${ }_{723}$ corfection capabilities are disabled. The introduction of the optimizations ${ }_{724}$ proposed in Arch. 3 has no significant impact on the dynamic power that ${ }_{725}$ remains almost equal to the one of Arch. 2.


Figure 15: Worst case dynanire poher consumption of the threo decoders for the three considered architectures Pawer is expressed in mW .

## 8. Conclusions $\bigcirc$

This Raper proposed BCH codec architectryes and its related auto- ${ }^{728}$ matic generation frymework which enables itscode eorrection capability to ${ }_{729}$ be selected in Pprederined range of valuer. Desiging an ECC system whose ${ }_{730}$ correctiral qapability can be modified inned has the potentiality to adapt ${ }_{731}$ the cprrection schema to the reliability requirements the flash encounters 732 Exing its ife-time, thus maximizing performance and reliability. proved that the proposed solution reduces spare area usage, decoding time, ${ }^{735}$ and power dissipation whenever small correction capability can be selected. ${ }^{736}$

Table 7: Generator polynomial expressed with the corresponding hexadecimal string of coefficients




[^0]:    

[^1]:    ${ }^{1}$ our BCH code has 1,800 primitive polynomials $\psi_{1}(x)$

[^2]:    ${ }^{2}$ Equivalent gates for state-of-the-art architectures have been estimated from the information provided in the papers

