BCH Code Based Multiple Bit Error Correction in GF Multiplier Circuits

P. Mahesh[1] & A. M . Jabir : Oxford Brookes University
J. Mathew[2] & D. K . Pradhan : Bristol University
S. P . Mohanty[3] : University of North Texas
E-mail: 09137484@brookes.ac.uk[1], jimson@cs.bristol.ac.uk[2], saraju.mohanty@unt.edu[3]

Presented by:
Oleg Garitselov : University of North Texas

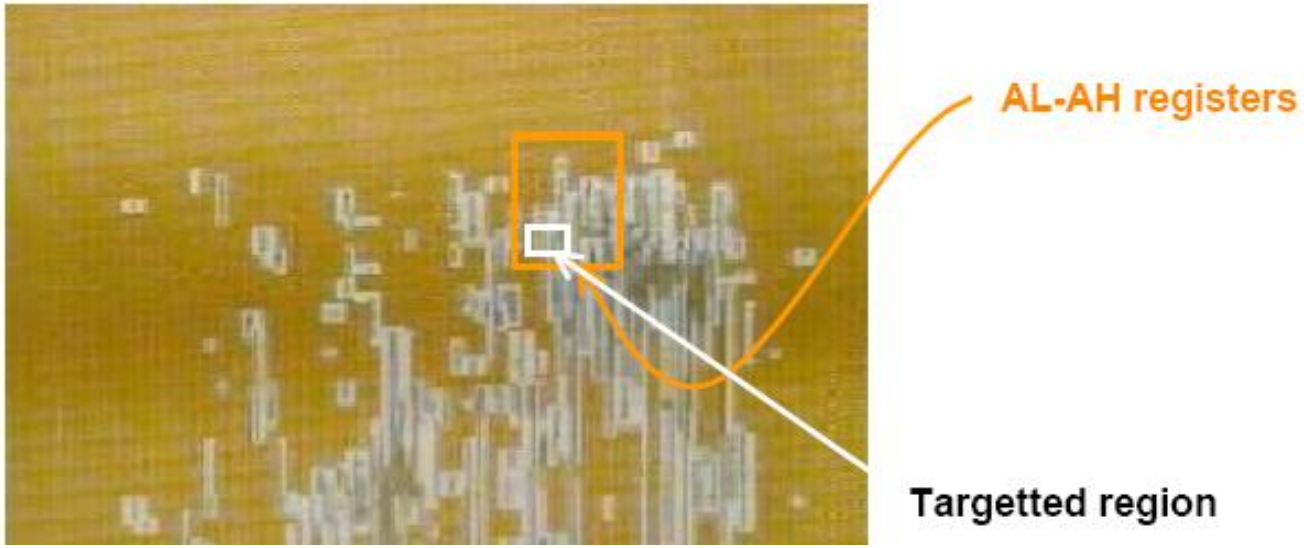
Overview

- Motivation
- Background
- Galois Arithmetic Circuits
- BCH Code Based Error Correction
- Design Steps
- Experimental Results
- Conclusion & Future Work

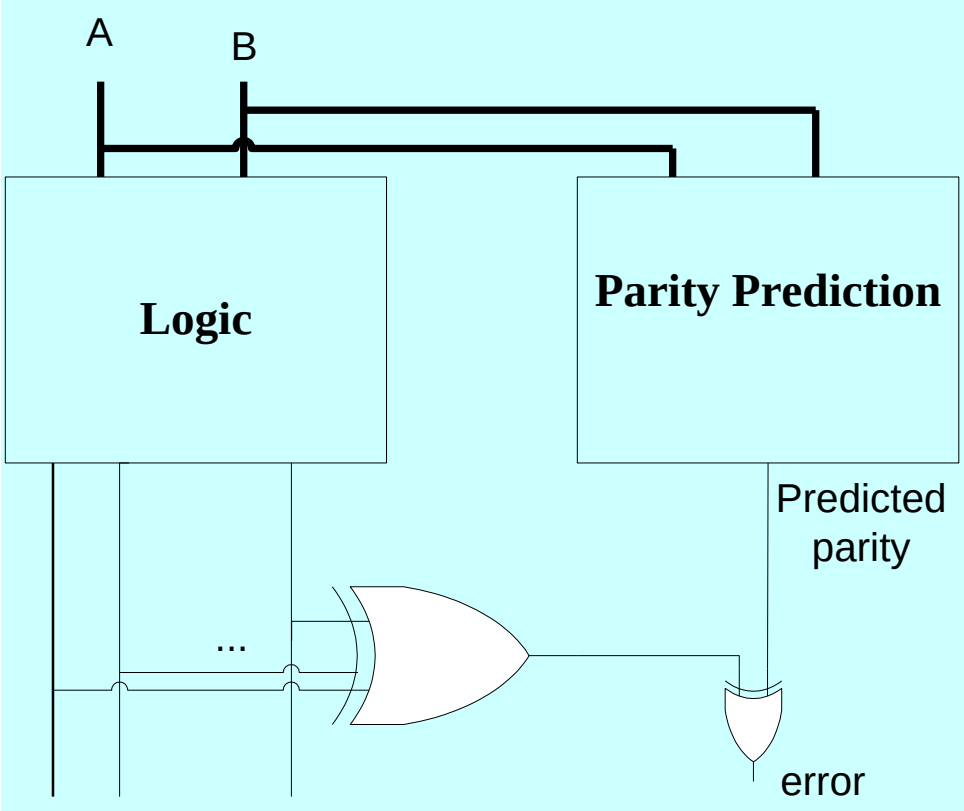
Motivation

- Fault attacks on Crypto. Hardware

Laser on the AL-AH registers

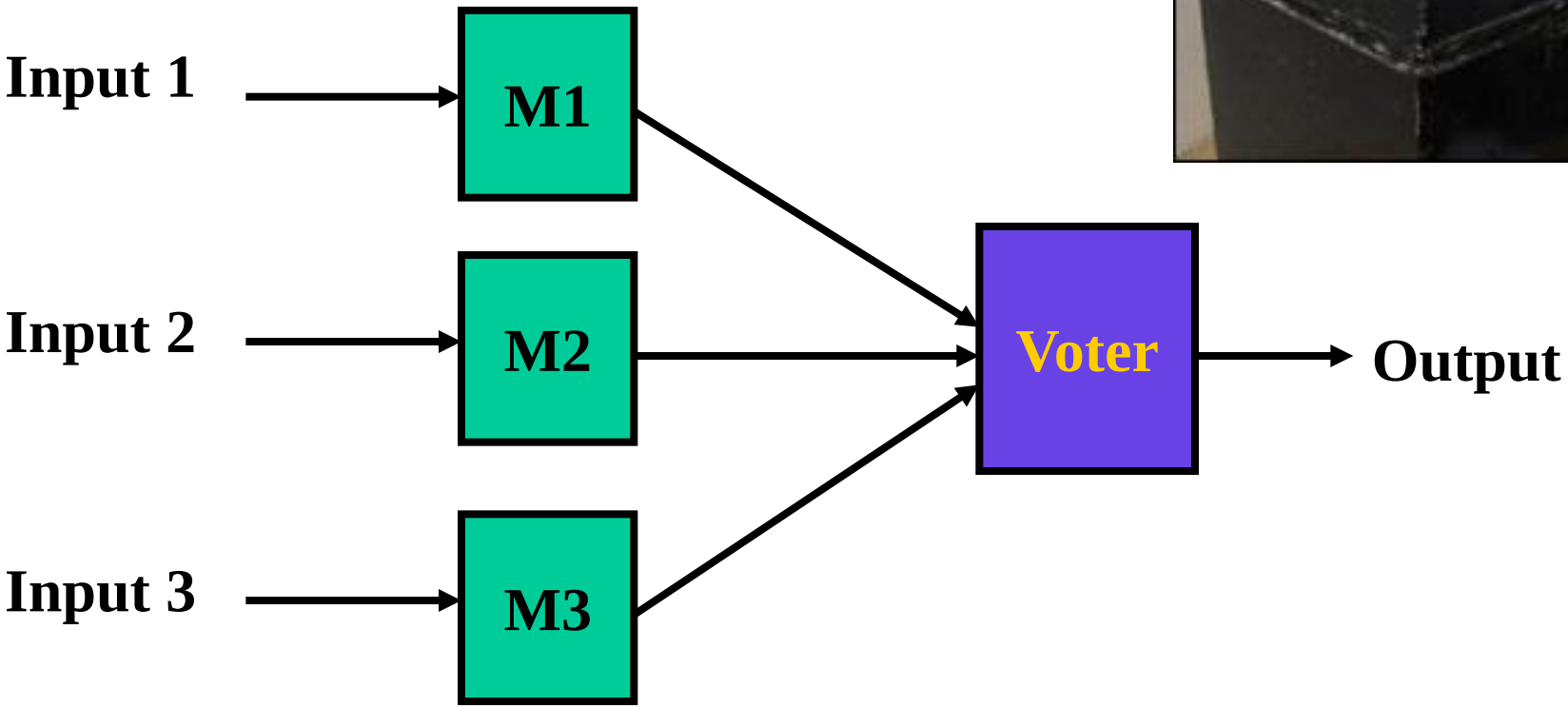


Previous Work



Ref: M. Nicoliadis , "**Carry checking/parity prediction adders and ALUs** ", IEEE Trans. VLSI Systems, vol. 11, Oct. 2003

Hardware Redundancy (TMR)



Finite Field

- Also called *Galois Field,* denoted by GF(*N*)
- $N = p^k$, p is a prime number, and k *is* an integer
- Each element is a k-tuple
- There are exactly N elements in the field (0, 1,…, N-1) or (0, 1, α, α^2 , α^3 ,... , α^{N-2}) where , α is a primitive element

Finite Field

- Two operators
  - ‘+’ called addition operation forming Abelian Group
  - ‘.’ called multiplication operation forming Abelian group.
- Additive and multiplicative identities 0 and 1 respectively
- Additive and multiplicative inverse exists for each element

GF(4) Elements

$[0, 1, \alpha, \alpha^2] = [0, 1, \alpha, \beta]$

Elements can be represented in $GF(2^n)$ as 2-tuples over GF(2).

Example GF(4)

Let $\alpha^2 = \beta$

Thus we have 4 elements 0, 1, α and β

GF(4)	2-tuple of GF(2) a_1 a_0	Polynomial Representation $a_1 x + a_0$
0	0 0	0
1	0 1	1
α	1 0	x
β	1 1	$(x+1)$

Addition and Multiplication

- Addition and Multiplication in GF(4)

+	0	1	α	β
0	0	1	α	β
1	1	0	β	α
α	α	β	0	1
β	β	α	1	0

*	0	1	α	β
0	0	0	0	0
1	0	1	α	β
α	0	α	β	1
β	0	β	1	α

- β is multiplicative inverse of α, and vice versa.

Generation of GF(2^m)

- Let us generate GF(8) with PP $p(x) = x^3 + x + 1$.
- Let α be a root of $p(x)$, i.e. $p(\alpha) = 0$.
- Then $\alpha^3 + \alpha + 1 = 0$, i.e. $\alpha^3 = \alpha + 1$.

α^0	$=$	1
α^1	$=$	α
α^2	$=$	α^2
α^3	$=$	$\alpha+1$
α^4	$=$	$\alpha^2+\alpha$
α^5	$=$	$\alpha^3+\alpha^2=\alpha^2+\alpha+1$
α^6	$=$	α^2+1
0	$=$	0

0	$\leftrightarrow$	$[0,0,0]$
α	$\leftrightarrow$	$[0,1,0]$
α^2	$\leftrightarrow$	$[1,0,0]$
$\alpha^3=\alpha+1$	$\leftrightarrow$	$[0,1,1]$
$\alpha^4=\alpha^2+\alpha$	$\leftrightarrow$	$[1,1,0]$
$\alpha^5=\alpha^2+\alpha+1$	$\leftrightarrow$	$[1,1,1]$
$\alpha^6=\alpha^2+1$	$\leftrightarrow$	$[1,0,1]$
$\alpha^7=1$	$\leftrightarrow$	$[0,0,1]$

Multiplication

Multiplication is done by using polynomial mod a primitive polynomial $P(x)$. i.e. $\alpha(x) \bullet \beta(x) \bmod P(x)$

Let $P(x) = x^2 + x + 1$

Thus $\alpha \bullet \beta = x(x+1) = x^2 + x$

So $(x^2 + x) \bmod x^2 + x + 1 = 1$

$\alpha \bullet \beta = 1$

$\alpha^{-1} = \beta$, $\beta^{-1} = \alpha$

α and β are inverse of each other

Example

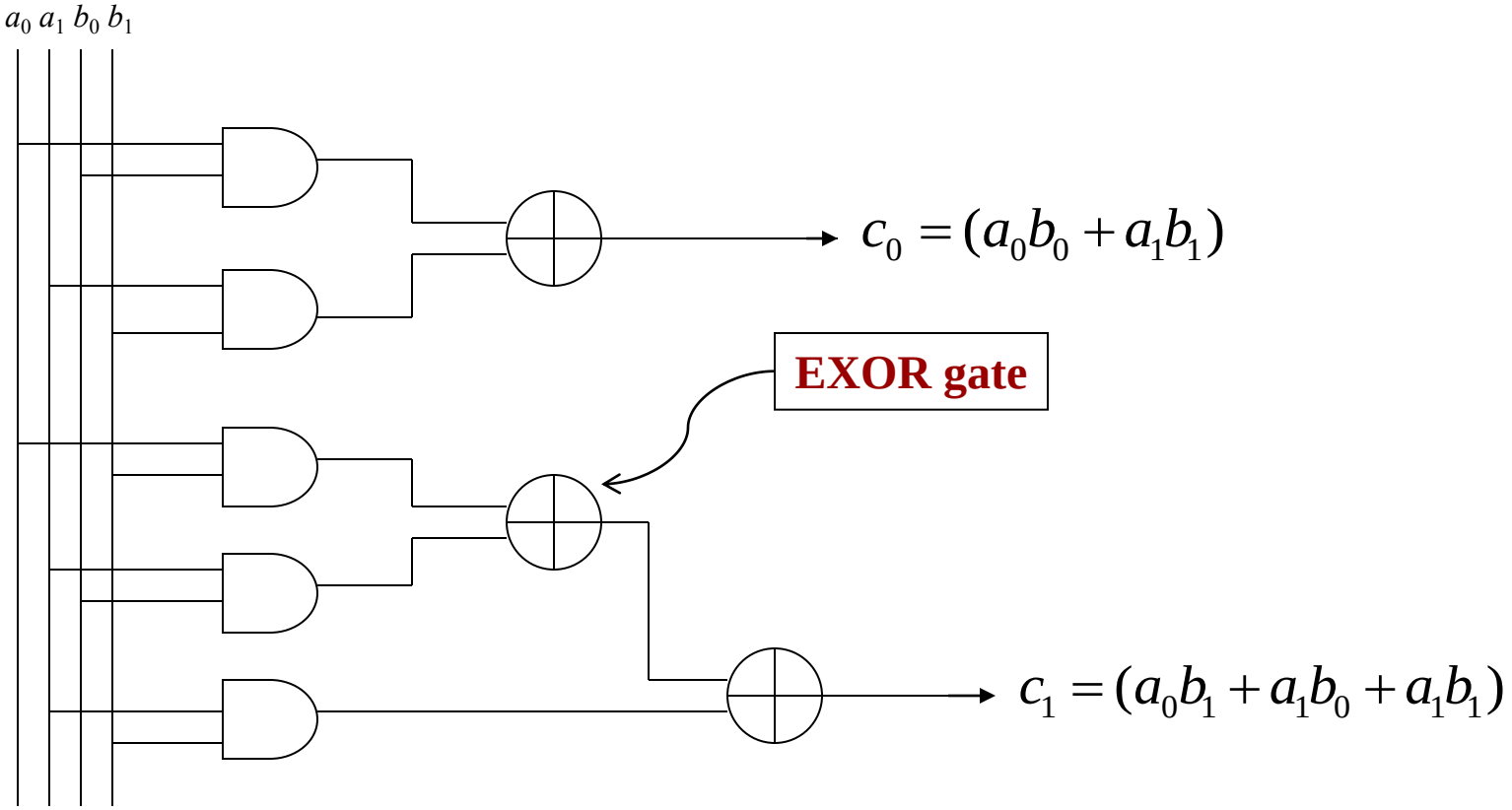
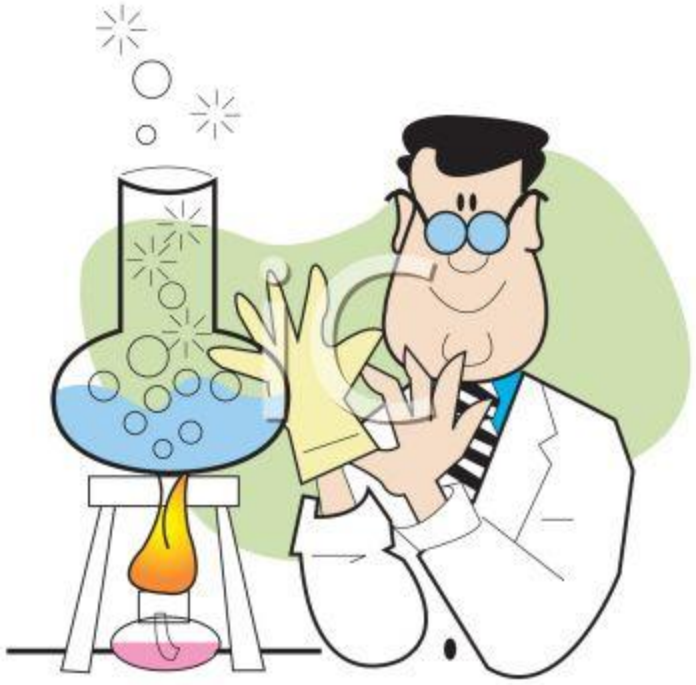
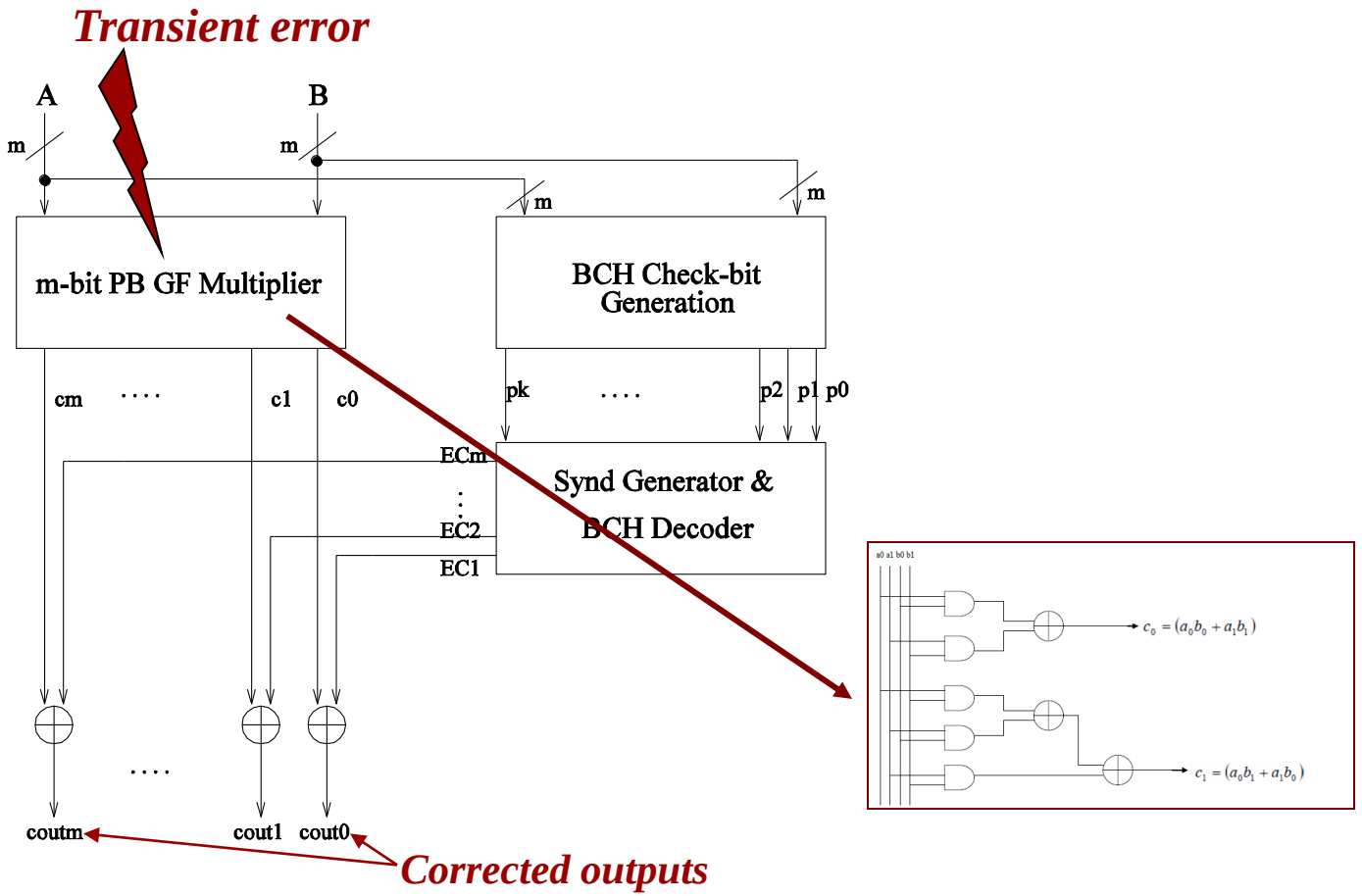


Figure:2 GF(4) multiplier

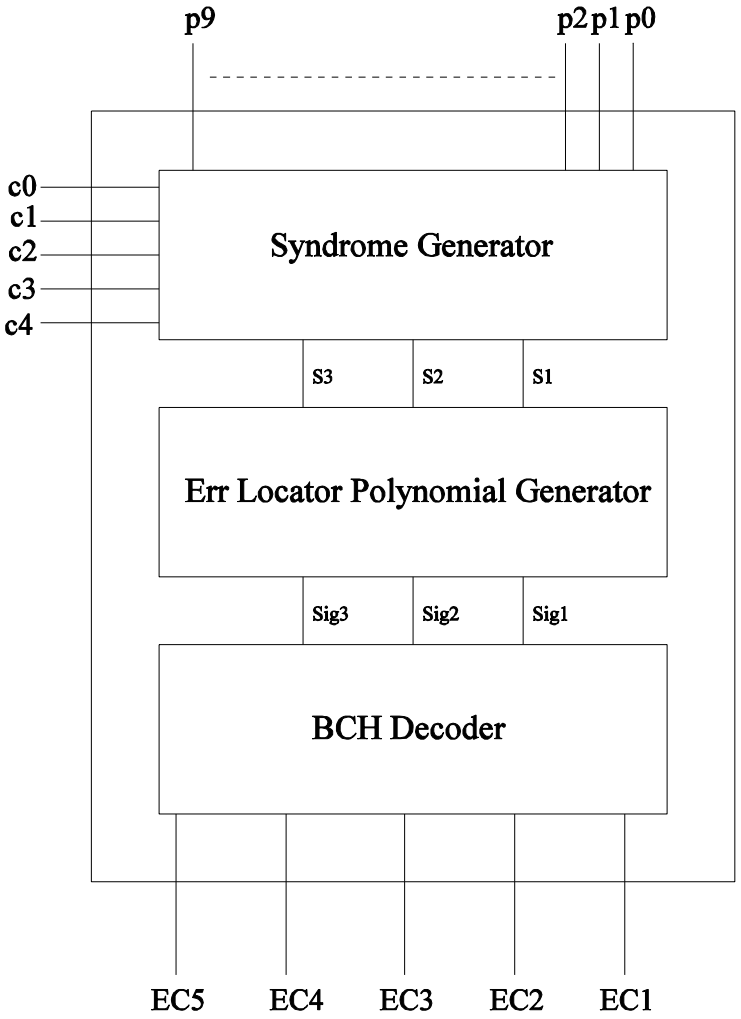
Multiple Bit Error Correction Using BCH Codes



Multiple Bit Error Correction Using BCH Codes: The Design Architecture

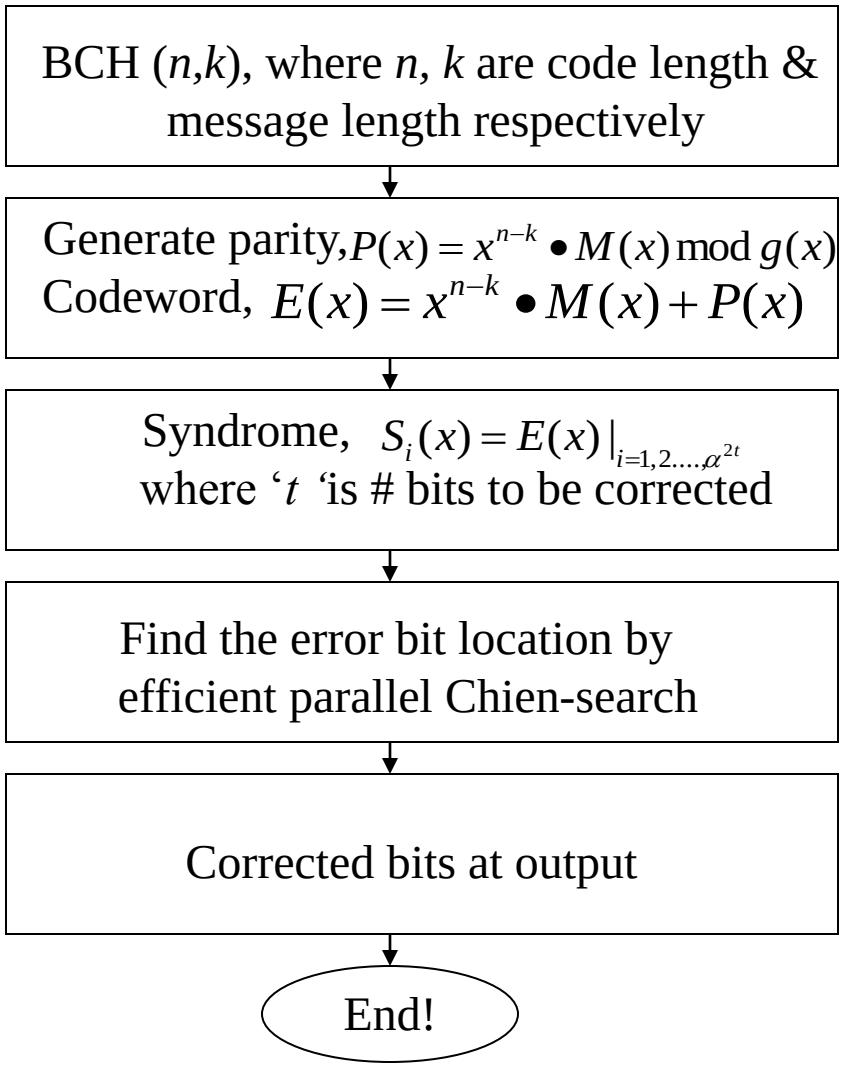


Multiple Bit Error Correction Using BCH Codes: Synd. Generator and BCH Decoder

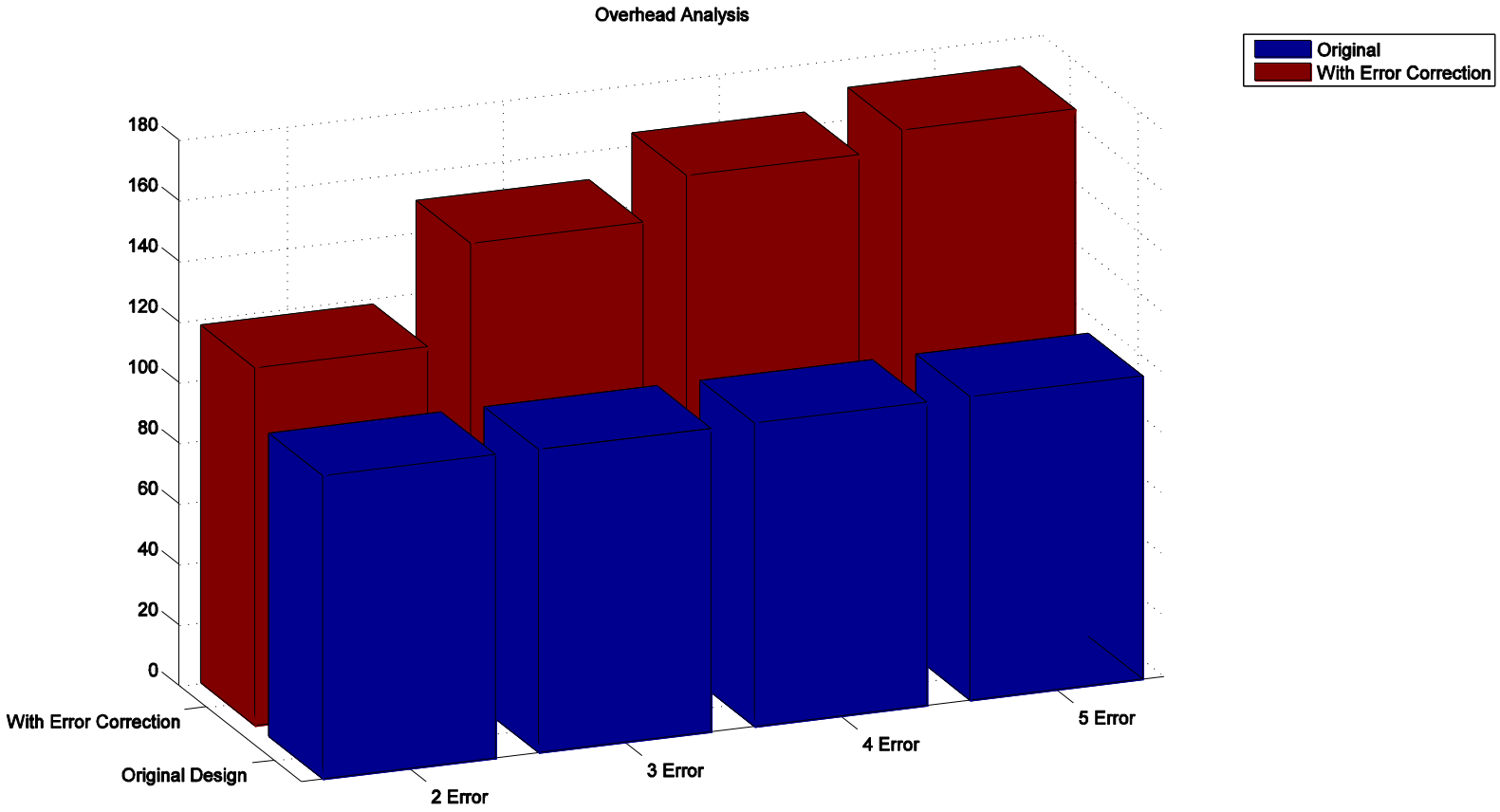


Multiple Bit Error Correction Using BCH Codes

BCH Algo:



Experiments & Results

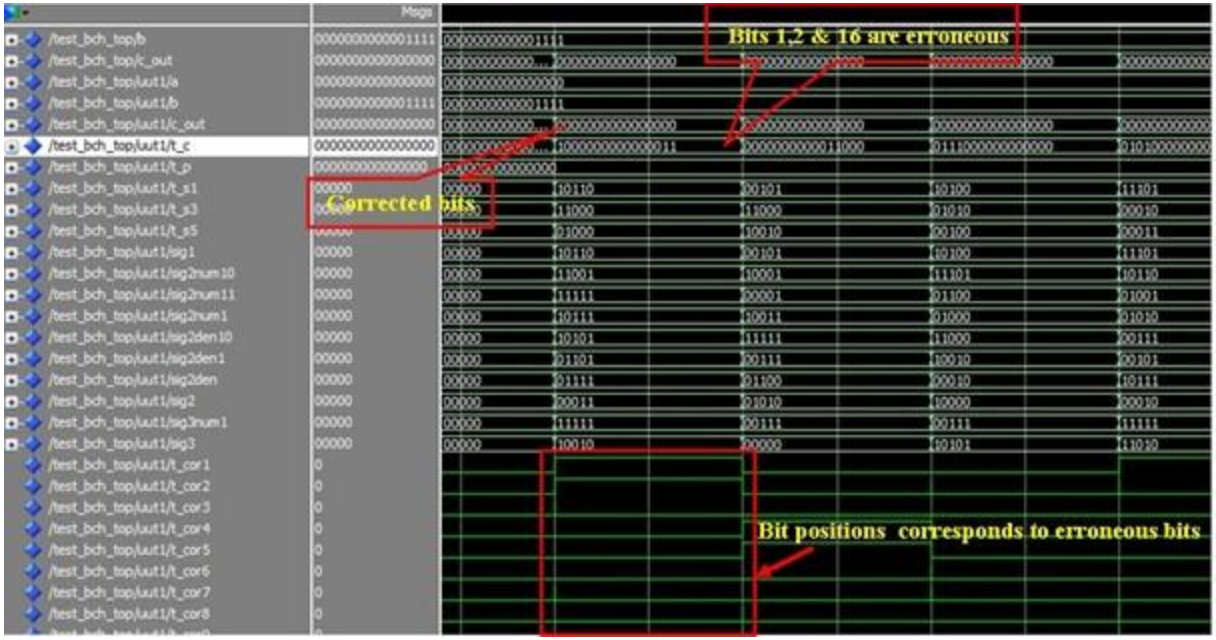


Experiments & Results

Property	[12]	[10]	Proposed	Proposed	Proposed
#errors correction	single	single	3 Errors	4 Errors	5 Errors
Coding technique	Hamming	LDPC	BCH	BCH	BCH
Overhead	>100%	100%	150.4%	164.04%	170.4%

Area comparison with other techniques

Modelsim simulation results



[10] Fault Tolerant Bit Parallel Finite Field Multipliers Using LDPC Codes, J. Mathew, J. Singh, A. M .Jabir, M. Hosseinabady, D. K .Pradhan, IEEE 2008.
[12] Single Error Correctable Bit Parallel Multipliers Over GF(2^m), J. Mathew, A. M . Jabir, H. Rahaman, D. K .Pradhan, IET Computer Digital Tech., Vol. 3, Iss. 3, pp. 281-288, 2009.

Conclusion & Future Work

- Multiple bit error correction with compromise over slightly higher area
- For a fixed number of bits to be corrected, percent area overhead reduces with larger and more practical multipliers
- Highly parallel and efficient Chien-search block
- This scheme can be easily extendable to GF multiplier of any size


Thank
YOU!

Questions?