Fault Tolerant Computation

Ismail San

Anadolu University

February 17, 2016
Outline

- Why fault tolerant computation is important?
- Causes of Errors
- Fault Tolerant State Machine Models
- Scalable Fault Tolerant GF Multiplication
- Architectural Considerations
- Concluding Remarks
Fault tolerant computing is really important for mission-critical systems, for instance, aerospace and defense sectors.
Motivation

- Fault tolerant computing is really important for mission-critical systems, for instance, aerospace and defense sectors.

- It also finds many application areas such as networking, automotive, embedded real-time systems, and medical equipment.
Fault tolerant computing is really important for mission-critical systems, for instance, aerospace and defense sectors.

It also finds many application areas such as networking, automotive, embedded real-time systems, and medical equipment.

The need for more reliable computing is increasing with
  + shrinking device geometries,
  + increasing chip size of CMOS technology,
  + higher transistor density,
  + decrease in switching voltage levels,
  + rise in switching speeds and
  + environmental effects.
Motivation

- Fault tolerant computing is really important for mission-critical systems, for instance, aerospace and defense sectors.
- It also finds many application areas such as networking, automotive, embedded real-time systems, and medical equipment.
- The need for more reliable computing is increasing with shrinking device geometries, increasing chip size of CMOS technology, higher transistor density, decrease in switching voltage levels, rise in switching speeds and environmental effects.
- Defective chip rates in computers and smartphones is %5. As the market proceed to more advanced process technologies, this percentage will increase.
Fault tolerant computing is really important for **mission-critical** systems, for instance, aerospace and defense sectors.

It also finds many application areas such as **networking, automotive, embedded real-time systems, and medical equipment.**

The need for more reliable computing is increasing with:
- shrinking device geometries,
- increasing chip size of CMOS technology,
- higher transistor density,
- decrease in switching voltage levels,
- rise in switching speeds and
- environmental effects.

Defective chip rates in computers and smartphones is %5. As the market proceeds to more advanced process technologies, this percentage will increase.

Faults, for instance, Radiation may create glitches, induced soft errors, became widely known in the 1970s with the introduction of dynamic RAM chips.
Fault tolerant computing

- It is the science of building systems so that they can continue to operate its intended function in the presence of faults.
It is the science of building systems so that they can continue to operate its intended function in the presence of faults.

Fault tolerant systems may be able to tolerate one or more fault types:

- transient or permanent hardware faults
- software and hardware design errors
- operator errors
- externally induced upsets or physical damage
It is the science of building systems so that they can continue to operate its intended function in the presence of faults.

Fault tolerant systems may be able to tolerate one or more fault types:
- transient or permanent hardware faults
- software and hardware design errors
- operator errors
- externally induced upsets or physical damage

It is challenging to build computers that automatically recover from random faults appeared in hardware components.
Dependability

Dependability is a measure of a system’s reliability, availability, and maintainability.

- **Reliability** refers to the ability of a system to perform its required function correctly under the stated conditions for a specified period of time.

- **Availability** refers to the ability of a system to accept requests.

- **Maintainability** refers to the ability of a system to undergo modifications and repairs.
There are many causes to create an error in the system;
There are many causes to create an error in the system;

- **Radiation**: "glitches", induced soft errors, alpha particles, cosmic rays and thermal neutrons.

**Figure**: Single event upset (SEU) caused by storing from an unwanted transient event due to the radiation [*Angela Sutton, Synopsys, Military & Aerospace Technical Bulletin Issue 1, 2013*].
There are many **causes** to create an error in the system;

- **Radiation**: "glitches", induced soft errors, alpha particles, cosmic rays and thermal neutrons.

*Figure:* Single event upset (SEU) caused by storing from an unwanted transient event due to the radiation [Angela Sutton, Synopsys, Military & Aerospace Technical Bulletin Issue 1, 2013].

- **Deliberate faults**: fault injection security attacks
There are many causes to create an error in the system;

- **Radiation**: "glitches", induced soft errors, alpha particles, cosmic rays and thermal neutrons.

**Figure**: Single event upset (SEU) caused by storing from an unwanted transient event due to the radiation [Angela Sutton, Synopsys, Military & Aerospace Technical Bulletin Issue 1, 2013].

- **Deliberate faults**: fault injection security attacks
- **Other causes**: random noise or signal integrity problems.
There are many causes to create an error in the system;

- *Radiation*: "glitches", induced soft errors, alpha particles, cosmic rays and thermal neutrons.

**Figure**: Single event upset (SEU) caused by storing from an unwanted transient event due to the radiation [Angela Sutton, Synopsys, Military & Aerospace Technical Bulletin Issue 1, 2013].

- *Deliberate faults*: fault injection security attacks
- Other causes: random noise or signal integrity problems.

All these effects have an impact on the correct operation of the circuit elements in the device and it eventually degrades the reliability.
Component failure rate is the expected number of failures per a period of time.

It depends on:
- the current age of the component (Bathtub curve)
- any voltage or physical shocks
- the ambient temperature
- technology
- complexity factors; the number of gates, the number of pins
SEE occurs both in space and on earth
SEE occurs both in space and on earth

Protons, electrons, neutrons, or alpha particles may perturb the MOS or bipolar device operation in either a destructive or non-destructive fashion.
- SEE occurs both in space and on earth
- Protons, electrons, neutrons, or alpha particles may perturb the MOS or bipolar device operation in either a destructive or non-destructive fashion
SEE occurs both in space and on earth

Protons, electrons, neutrons, or alpha particles may perturb the MOS or bipolar device operation in either a destructive or non-destructive fashion.

- **Non-destructive events**
  - Single event transient
  - Single event upset
  - Multi bit upset

- **Destructive events**
  - Single Event Latchup (gone away with improved tech.)
  - Single Event Burnout
- SEE occurs both in space and on earth
- Protons, electrons, neutrons, or alpha particles may perturb the MOS or bipolar device operation in either a destructive or non-destructive fashion

**Non-destructive events**
- Single event transient
- Single event upset
- Multi bit upset

**Destructive events**
- Single Event Latchup (gone away with improved tech.)
- Single Event Burnout
Due to the manufacturing defect

Only one line is faulty

Faulty line is always set to 0 or 1

Faulty line can be an input or output of a gate
- Due to the manufacturing defect
- Only one line is faulty
- Faulty line is always set to 0 or 1
- Faulty line can be an input or output of a gate

**Stuck-at 0**
- One of the gate input was permanently connected to ground
- Fault: $a$ stuck-at 0, signal $a$ always be 0
Common Approach for Fault Tolerance

- All elements in the logic need to be tripled in this approach.

**Figure:** TMR helps mitigate SEUs induced by radiation effects by tripling the logic [Angela Sutton, Synopsys, Military & Aerospace Technical Bulletin Issue 1, 2013].
Errors are introduced in the noisy environment

Some bits are flipped in the codeword

Noisy environment can be in computing

Decoder locates the errors in the received codeword

Error detection

Error correction
Data (d) → Encoder → Codeword (c) → Noisy Environment → Codeword (\(\vec{c}\)) → Decoder → Data (d)

- Errors are introduced in the noisy environment
  - Some bits are flipped in the codeword
  - Noisy environment can be in computing
General model for fault tolerance with ECC

- Errors are introduced in the noisy environment
  - Some bits are flipped in the codeword
  - Noisy environment can be in computing
- Decoder locates the errors in the received codeword
  - Error detection
  - Error correction
State Machine Replication

- Two different failure models
  - **Byzantine**: requires \(2t + 1\) replicas to tolerate \(t\) many faults.
  - **Fail-stop**: stop the process and stop is detectable with \(t + 1\) replicas.

(a) FSM

(a) Duplicated FSM (Error detection)
• Criteria ensuring the reliability of SM
  1. All possible states are defined
  2. There is no possibility of a hang state
  3. No false state is entered
  4. An SEU exerts no effect on the SM
- \( \log_2 N \)-bits are used to represent \( N \) states and requires to use \( \log_2 N \) flip-flops

- Simplest state machine encoding

- Binary count of state machine number in sequence
  - Suppose you have 8 states
  - Binary encoded state 4 is represented with 100

- Satisfies 1 and 2 criterias

- For a SM with 16 states,
  - Binary FSM requires 4 flip-flops

<table>
<thead>
<tr>
<th>State</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>000</td>
</tr>
<tr>
<td>S1</td>
<td>001</td>
</tr>
<tr>
<td>S2</td>
<td>010</td>
</tr>
<tr>
<td>S3</td>
<td>011</td>
</tr>
<tr>
<td>S4</td>
<td>100</td>
</tr>
<tr>
<td>S5</td>
<td>101</td>
</tr>
<tr>
<td>S6</td>
<td>110</td>
</tr>
<tr>
<td>S7</td>
<td>111</td>
</tr>
</tbody>
</table>
One-hot encoding

- \( N \)-bits are used to represent \( N \) states and requires to use \( N \) flip-flops
- Each state is distinguishable by its own flip-flop
- All bits except one bit are 0 in a string
- The position of 1 in the string represents the state
  - Suppose you have 8 states
  - One-hot encoded state 4 is represented with 00010000
- All states are equally different from each other
  - Satisfies 1 through 3 criteria
  - Simplicity
    - Inefficient due to high number of bits to represent a state
- For a SM with 16 states,
  - Binary FSM requires 4 flip-flops
  - One-hot FSM requires 16 flip-flops

<table>
<thead>
<tr>
<th>State</th>
<th>One-hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00000001</td>
</tr>
<tr>
<td>S1</td>
<td>00000010</td>
</tr>
<tr>
<td>S2</td>
<td>00000100</td>
</tr>
<tr>
<td>S3</td>
<td>00010000</td>
</tr>
<tr>
<td>S4</td>
<td>00100000</td>
</tr>
<tr>
<td>S5</td>
<td>01000000</td>
</tr>
<tr>
<td>S6</td>
<td>10000000</td>
</tr>
</tbody>
</table>
Hamming 2 encoding

- Hamming distance of 2 encoding requires fewer than $N$-bits to represent the whole $N$ states.

- Between any two adjacent defined legal states, Hamming distance is 2.

- If an SEU appears, it changes the content of the flip-flops to a defined illegal state representation.

- Since illegal states are known, it can be detected automatically.

- It prevents the SM from entering into a illegal state.

- For illegal states, some recovery process can be defined:
  - Suppose you have 8 states with $S_4 = 1001$ and $S_5 = 1010$
  - These two states differ in 2 places

- All states are not equally different from each other
  + Satisfies 1 through 3 criteria
  + Efficient memory utilization

<table>
<thead>
<tr>
<th>State</th>
<th>Hamming 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0000</td>
</tr>
<tr>
<td>S1</td>
<td>0011</td>
</tr>
<tr>
<td>S2</td>
<td>0101</td>
</tr>
<tr>
<td>S3</td>
<td>0110</td>
</tr>
<tr>
<td>S4</td>
<td>1001</td>
</tr>
<tr>
<td>S5</td>
<td>1010</td>
</tr>
<tr>
<td>S6</td>
<td>1100</td>
</tr>
<tr>
<td>S7</td>
<td>1111</td>
</tr>
</tbody>
</table>
Between two adjacent defined legal states, Hamming distance is 3.

If an SEU appears, it changes the content of the flip-flops to a defined illegal state representation.

Illegal states are unique to the legal state representation.

Since the illegal states are associated with only one state, SM continues normal operation even if SEU appears.

SEU is corrected thanks to having Hamming distance 3 between states.

Suppose you have 8 states with $S_4 = 101010$ and $S_5 = 101101$.

These two states differ in 3 places.

Satisfies all four criterias.
Fault tolerant state machines

- Hamming 3 presents the best fault tolerance (0 errors in test)
- However, it consumes the most resources and slower than Hamming 2.
- Hamming 2 had fewer errors than binary encoding
- One-hot have the most errors due to its largest amount of target ffs.
- One-hot encoding is the slowest and shows poor use of resources.

<table>
<thead>
<tr>
<th>State</th>
<th>Binary</th>
<th>One-hot</th>
<th>Hamming 2</th>
<th>Hamming 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>000</td>
<td>00000001</td>
<td>0000</td>
<td>000000</td>
</tr>
<tr>
<td>S1</td>
<td>001</td>
<td>00000010</td>
<td>0011</td>
<td>000111</td>
</tr>
<tr>
<td>S2</td>
<td>010</td>
<td>00000100</td>
<td>0101</td>
<td>011001</td>
</tr>
<tr>
<td>S3</td>
<td>011</td>
<td>00001000</td>
<td>0110</td>
<td>011110</td>
</tr>
<tr>
<td>S4</td>
<td>100</td>
<td>00010000</td>
<td>1001</td>
<td>101010</td>
</tr>
<tr>
<td>S5</td>
<td>101</td>
<td>00100000</td>
<td>1010</td>
<td>101101</td>
</tr>
<tr>
<td>S6</td>
<td>110</td>
<td>01000000</td>
<td>1100</td>
<td>110011</td>
</tr>
<tr>
<td>S7</td>
<td>111</td>
<td>10000000</td>
<td>1111</td>
<td>110100</td>
</tr>
</tbody>
</table>

**Challenge**: to incorporate an error-correction codes in an operation in order to have a fault tolerant computation of selected operation.
**Challenge**: to incorporate an error-correction codes in an operation in order to have a fault tolerant computation of selected operation.

Finite field arithmetic is used in many different areas:
- especially for Cryptography
- Coding theory

Scalability refers to the ability of a system which is capable to handle a growing amount of work.
Challenge: to incorporate an error-correction codes in an operation in order to have a fault tolerant computation of selected operation.

Finite field arithmetic is used in many different areas:
+ especially for Cryptography
+ Coding theory

Two important issues:
+ Preventing from error propagation,
+ Predicting new redundancy by using previous input information for all components of the GF multiplier
**Challenge**: to incorporate an error-correction codes in an operation in order to have a fault tolerant computation of selected operation.

Finite field arithmetic is used in many different areas:
- especially for Cryptography
- Coding theory

**Two important issues:**
- Preventing from error propagation,
- Predicting new redundancy by using previous input information for all components of the GF multiplier

**Focus on a scalable generic architecture**
**Challenge**: to incorporate an error-correction codes in an operation in order to have a fault tolerant computation of selected operation.

- Finite field arithmetic is used in many different areas:
  - especially for Cryptography
  - Coding theory

**Two important issues:**
- Preventing from error propagation,
- Predicting new redundancy by using previous input information for all components of the GF multiplier

**Focus on a scalable generic architecture**

- **Scalability** refers to the ability of a system which is capable to handle a growing amount of work.
Let the binary field \( GF(2^m) \) be generated with a monic irreducible polynomial of degree \( m \)

\[
F(x) = x^m + \sum_{i=1}^{m-1} f_i x^i + 1
\]  

(1)
Let the binary field \( \text{GF}(2^m) \) be generated with a monic irreducible polynomial of degree \( m \)

\[
F(x) = x^m + \sum_{i=1}^{m-1} f_i x^i + 1
\]  

(1)

The polynomial basis of \( \text{GF}(2^m) \) is then the set \( \{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\} \)
Let the binary field $\text{GF}(2^m)$ be generated with a monic irreducible polynomial of degree $m$

$$F(x) = x^m + \sum_{i=1}^{m-1} f_i x^i + 1$$  \hspace{1cm} (1)

The polynomial basis of $\text{GF}(2^m)$ is then the set $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$

The product $C$ is obtained with respect to the polynomial basis:

$$S = A \cdot B$$
Modular reduction:

\[ C = S \mod F(\alpha) = \sum_{i=0}^{m-1} b_i \cdot X^{(i)}, \]  

(2)

where

\[ X^{(i)} = \alpha \cdot X^{(i-1)} \mod F(\alpha), \quad 1 \leq i \leq m - 1, \]  

(3)

and \( X^{(0)} = A \).
Bit-parallel GF multiplication

- Modular reduction:

\[
C = S \mod F(\alpha) = \sum_{i=0}^{m-1} b_i \cdot X^{(i)},
\]

(2)

where

\[
X^{(i)} = \alpha \cdot X^{(i-1)} \mod F(\alpha), \quad 1 \leq i \leq m - 1,
\]

(3)

and \(X^{(0)} = A\).

**Figure:** Bit-parallel polynomial basis GF\((2^m)\) multiplication (Figure adopted from [Reyhani-Masoleh and Hasan, 2006].)
Error Propagation: \( F(X) = X^8 + X^4 + X^3 + X + 1 \)
Error Propagation: 

\[ F(X) = X^8 + X^4 + X^3 + X + 1 \]
Error Propagation: \( F(X) = X^8 + X^4 + X^3 + X + 1 \)
Error Propagation: \( F(X) = X^8 + X^4 + X^3 + X + 1 \)
Error Propagation: \( F(X) = X^8 + X^4 + X^3 + X + 1 \)
The Proposed Method

- Enabled an error-correcting code in the computation of multiplication operation over GF.

**Figure:** The proposed method for fault tolerant GF multiplication architecture.
The Proposed Method

- Enabled an error-correcting code in the computation of multiplication operation over GF.
- Separated the data and redundancy computation to different components.

Figure: The proposed method for fault tolerant GF multiplication architecture.
The Proposed Method

- Enabled an error-correcting code in the computation of multiplication operation over GF.
- Separated the data and redundancy computation to different components.
- Came up with redundancy predictions for each component.

**Figure:** The proposed method for fault tolerant GF multiplication architecture.
The Proposed Method

- Enabled an error-correcting code in the computation of multiplication operation over GF.
- Separated the data and redundancy computation to different components.
- Came up with redundancy predictions for each component.
- Avoided the error-propagation problem.

**Figure:** The proposed method for fault tolerant GF multiplication architecture.
The Proposed Method

- Enabled an error-correcting code in the computation of multiplication operation over GF.
- Separated the data and redundancy computation to different components.
- Came up with redundancy predictions for each component.
- Avoided the error-propagation problem.
- Placed the components according to the architectural considerations.

Figure: The proposed method for fault tolerant GF multiplication architecture.
Let $\mathbb{F}$ be a field of 2. If $k$ is an integer and $k \geq 2$, $n$ is defined as $n = (2^k - 1)$. Then, a binary Hamming code of type $[n, n - k, 3]$ is the code $c$ defined by the $k \times n$ parity check matrices.

$$H = [v_1 | v_2 | \cdots | v_n],$$
Let $\mathbb{F}$ be a field of 2. If $k$ is an integer and $k \geq 2$, $n$ is defined as $n = (2^k - 1)$. Then, a binary Hamming code of type $[n, n - k, 3]$ is the code $c$ defined by the $k \times n$ parity check matrices.

$$H = [v_1 | v_2 | \cdots | v_n] ,$$

This lets us format the codeword, which consists of data and redundancy bits. It is denoted by $c$ and its elements are named $c_i$. The codeword length is represented by $n$.

$$c = [c_0 | c_1 | \cdots | c_{n-1}]_{1 \times n}$$
Hamming Code

- Let $\mathbb{F}$ be a field of 2. If $k$ is an integer and $k \geq 2$, $n$ is defined as $n = (2^k - 1)$. Then, a binary Hamming code of type $[n, n - k, 3]$ is the code $c$ defined by the $k \times n$ parity check matrices.

$$H = [v_1|v_2|\cdots|v_n],$$

- This lets us format the codeword, which consists of data and redundancy bits. It is denoted by $c$ and its elements are named $c_i$. The codeword length is represented by $n$.

$$c = [c_0|c_1|\cdots|c_{n-1}]_{1 \times n}$$

- The data word is represented by $d$ and its elements are named $d_i$. The data length is denoted by $k$.

$$d = [d_0|d_1|\cdots|d_{k-1}]_{1 \times k}$$
Hamming Code

- Let $\mathbb{F}$ be a field of 2. If $k$ is an integer and $k \geq 2$, $n$ is defined as $n = (2^k - 1)$. Then, a binary Hamming code of type $[n, n - k, 3]$ is the code $c$ defined by the $k \times n$ parity check matrices.

$$H = [v_1 | v_2 | \cdots | v_n],$$

- This lets us format the codeword, which consists of data and redundancy bits. It is denoted by $c$ and its elements are named $c_i$. The codeword length is represented by $n$.

$$c = [c_0 | c_1 | \cdots | c_{n-1}]_{1 \times n}$$

- The data word is represented by $d$ and its elements are named $d_i$. The data length is denoted by $k$.

$$d = [d_0 | d_1 | \cdots | d_{k-1}]_{1 \times k}$$

- The $p$ represents the redundancy word and its elements are named $p_i$. The length of the redundancy is denoted by $r$ and is simply calculated by $r = n - k$.

$$p = [p_0 | p_1 | \cdots | p_{r-1}]_{1 \times r}$$
Redundancy Generation based on Hamming Code

\[ p_i = \sum_{j=0}^{N/2s-1} \sum_{t=0}^{s-1} c_{2sj+t} + s + t + c_s \]

<table>
<thead>
<tr>
<th>Codeword bits</th>
<th>( p_3 )</th>
<th>( p_2 )</th>
<th>( p_1 )</th>
<th>( p_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nop</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>( p_0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>( p_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>( d_0 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>( p_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>( d_1 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>( d_2 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>( d_3 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>( p_3 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>( d_4 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>( d_5 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>( d_6 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>( d_7 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>( d_8 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>( d_9 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>( d_{10} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Parity bit locations in codeword
The remaining bits are data bits

0 shows not used bits
1 shows used bits

The bits used for generating \( p_2 \)

Figure: Codeword pattern based on Hamming code
Redundancy Generation based on Hamming Code

- The format
  \[ c = p \times U + d \times V \]
  \[ c = [p_0|p_1|\cdots|p_{r-1}]_{1\times r} \times U_{r\times n} + [d_0|d_1|\cdots|d_{k-1}]_{1\times k} \times V_{k\times n} \]  

- Two algorithms:

  **Input:** \( k \geq r \) and \( e_1, \cdots, e_k \) unit column vectors

  **Output:** \( U \) and \( V \) matrices

  1. \( j \leftarrow 1 \)
  2. **for** \( i \leftarrow 1 \) **to** \( n \) **do**
  3.     **if** \( (i \land (i-1)) = 0 \) **then**
  4.         \( u_i \leftarrow e_j \)
  5.         \( v_i \leftarrow 0 \)
  6.         \( j \leftarrow j + 1 \)
  7.     **else**
  8.         \( u_i \leftarrow 0 \)
  9.         \( v_i \leftarrow e_t \)
  10. **end if**
  11. **end for**
  12. \( U \leftarrow [u_0|u_1|\cdots|u_n] \)
  13. \( V \leftarrow [v_0|v_1|\cdots|v_n] \)

  **Input:** the codeword \( c \)

  **Output:** the parity \( p \)

  1. \( c = [c_0|c_1|\cdots|c_{n-1}] \)
  2. **for** \( i \leftarrow 0 \) **to** \( r - 1 \) **do**
  3.     \( s \leftarrow 2^i \)
  4.     \( p_i \leftarrow c_s \)
  5. **for** \( j \leftarrow 1 \) **to** \( n \) **do**
  6.     **if** \( (j \land s) = s \) **then**
  7.         \( p_i \leftarrow c_j \oplus p_i \)
  8.     **end if**
  9. **end for**
  10. **end for**
  11. \( p = [p_0|p_1|\cdots|p_{r-1}] \)
Redundancy Predictions

Figure: Bit-parallel polynomial basis $\mathbb{GF}(2^m)$ multiplication (Figure adopted from [Reyhani-Masoleh and Hasan, 2006].)
**Redundancy Predictions**

![Diagram of Bit-parallel polynomial basis GF($2^m$) multiplication](image)

**Figure:** Bit-parallel polynomial basis GF($2^m$) multiplication (Figure adopted from [Reyhani-Masoleh and Hasan, 2006].)

- **Redundancy Prediction of a Sum (⊕) Unit**

\[
\hat{p}(z) = p(x) + p(y).
\]  

(5)
Redundancy Predictions

Figure: Bit-parallel polynomial basis $\mathbb{GF}(2^m)$ multiplication (Figure adopted from [Reyhani-Masoleh and Hasan, 2006].

- Redundancy Prediction of a Sum ($\oplus$) Unit
  \[ \hat{p}(z) = p(x) + p(y). \]  
- Redundancy Prediction of a Product ($\land$) Unit
  \[ \hat{p}(w) = b_i \cdot p(x). \]
Redundancy Predictions

Figure: Bit-parallel polynomial basis $GF(2^m)$ multiplication (Figure adopted from [Reyhani-Masoleh and Hasan, 2006].

- **Redundancy Prediction of a Sum ($\oplus$) Unit**

$$\hat{p}^{(z)} = p^{(x)} + p^{(y)}.$$ (5)

- **Redundancy Prediction of a Product ($\wedge$) Unit**

$$\hat{p}^{(w)} = b_i \cdot p^{(x)}$$ (6)

- **Redundancy Prediction of the Multiplier ($\times02$) Unit**

$$\hat{p}_i = p_i + \sum_{j=0}^{\frac{N}{2s}-1} (c_{2sj+s-1} + c_{2sj+2s-1}) + \sum_{j=0}^{\frac{N}{2s}-1} a_{m-1} \cdot g_j \times \mathcal{V}$$ (7)
Figure: The proposed method for fault tolerant GF multiplication architecture.
Error Propagation in Feedback Signal

Protecting error distribution in feedback for $x_0$ unit

$am-1 \ a_{m-2} \ ai \ a_0$

$X_{m-1} \ X_{i+1} \ X_1 \ X_0$

$A \ll 1$

Addition of $am-1$ with $g$

Protecting error distribution in feedback for $x_0$ unit
Figure: Tripling the generation of feedback signal to solve the error propagation.
Mapping the Equations onto the Architecture

\[ c_a = p_a \times U + a \times V \]

\[ p_i^{(a)} = \sum_{j=0}^{s-1} \sum_{t=0}^{s-1} c_{2s_j+s+t}^{(a)} + c_s^{(a)} \]

\[ i = \sum_{j=0}^{N/2s-1} \sum_{t=0}^{s-1} c_{2s_j+s+t}^{(a)} + c_s^{(a)} \]

\[ c_x = p_x \times U + x \times V \]

\[ p_i^{(x)} = \sum_{j=0}^{N/2s-1} \sum_{t=0}^{s-1} c_{2s_j+s+t}^{(x)} + c_s^{(x)} \]

\[ \hat{p}_x = p_x \times U + a \ll 1 \times V + a_{m-1} \cdot g \times V \]

\[ c_x = \hat{p}_x \times U + a \ll 1 \times V + a_{m-1} \cdot g \times V \]

Figure: Mapping the equations.
The required components for *Redundant GF Multiplier*

- **A**
- **pA**
- **B**
- **Shift**
- **C**
- **pC**

The diagram illustrates the flow of data through various stages, including:

1. **Init**
2. **Error Correction Mechanism**
3. **Corrected Data**
4. **Redundancy Prediction**
5. **Redundancy Data**

The process involves the multiplication of **A** and **B**, followed by error correction and redundancy prediction, leading to the corrected data **C** and the parity check bit **pC**.
## Performance Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple Redundant GF Multiplier</td>
<td>4†</td>
<td>83</td>
<td>66</td>
<td>105</td>
<td>522</td>
<td>522</td>
<td>6.29</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>204</td>
<td>177</td>
<td>309</td>
<td>493</td>
<td>493</td>
<td>2.42</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>480</td>
<td>405</td>
<td>678</td>
<td>436</td>
<td>436</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>727</td>
<td>873</td>
<td>1363</td>
<td>435</td>
<td>435</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>1468</td>
<td>1821</td>
<td>3215</td>
<td>416</td>
<td>416</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>247</td>
<td>2638</td>
<td>4470</td>
<td>6724</td>
<td>206</td>
<td>206</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>502</td>
<td>4089</td>
<td>9063</td>
<td>12870</td>
<td>160</td>
<td>160</td>
<td>0.04</td>
</tr>
<tr>
<td>Redundant GF Multiplier</td>
<td>4†</td>
<td>27</td>
<td>36</td>
<td>61</td>
<td>498</td>
<td>498</td>
<td>18.44</td>
</tr>
<tr>
<td>The proposed method in this study</td>
<td>11</td>
<td>70</td>
<td>85</td>
<td>178</td>
<td>387</td>
<td>387</td>
<td>5.53</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>156</td>
<td>157</td>
<td>404</td>
<td>296</td>
<td>296</td>
<td>1.90</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>257</td>
<td>316</td>
<td>676</td>
<td>233</td>
<td>233</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>572</td>
<td>636</td>
<td>1331</td>
<td>186</td>
<td>186</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>247</td>
<td>812</td>
<td>2020</td>
<td>2429</td>
<td>160</td>
<td>160</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>502</td>
<td>1413</td>
<td>4063</td>
<td>4580</td>
<td>133</td>
<td>133</td>
<td>0.09</td>
</tr>
</tbody>
</table>

†Primitive trinomial is used.

Note that primitive 7-nomials are used for the binary extension fields of interest for other multiplier widths.
Fault tolerant GF multiplier method

- The presented Redundant GF multiplication method continue to operate in the presence of single-bit faults.
- The method is able to tolerate single-bit errors and does not allow them to distribute in the architecture to create multi-bit errors at the output.
Conclusions

- **Fault tolerant GF multiplier method**
  - The presented Redundant GF multiplication method continue to operate in the presence of single-bit faults.
  - The method is able to tolerate single-bit errors and does not allow them to distribute in the architecture to create multi-bit errors at the output.

- **Hardware architecture of the proposed method**
  - Parameterizable hardware architecture
  - Efficient performance results.
Conclusions

- **Fault tolerant GF multiplier method**
  - The presented Redundant GF multiplication method continues to operate in the presence of single-bit faults.
  - The method is able to tolerate single-bit errors and does not allow them to distribute in the architecture to create multi-bit errors at the output.

- **Hardware architecture of the proposed method**
  - Parameterizable hardware architecture
  - Efficient performance results.

- **Compact and explicit formulations for the generation and prediction of parity bits based on Hamming code for GF multiplier are proposed in the paper.**
Future Works

- A single-bit Resistant AES Architecture.
Future Works

- A single-bit Resistant AES Architecture.
- Elliptic Curve extensively uses GF arithmetic.
Future Works

- A single-bit Resistant AES Architecture.
- Elliptic Curve extensively uses GF arithmetic.
- Other ECC codes can be evaluated.
I would like to acknowledge:

- Peter Mueller and Ismail San. *Scalable Fault Correction Mechanism for the Multiplication over GF(2^m)*, During internship at IBM Research, 2012.
Thank you for your attention

Questions?