ECC Protection in Software

Motivation

- COTS Board in ARGOS
  - Transient errors (SEUs) in memory
  - No hardware ECC
  - Application corruption
    - Long repair time
    - Lost experiment time
  - FT in applications
    - Added complexity due to redundancy
    - Single-point-of-failure
    - Operating system corruption

Software Implemented ECC

- Code Segments
  - Fixed content after link stage
  - Generate ECC bits on the board
  - Scrub periodically
- Data Segments
  - Read-only data
  - Stored results
  - Random read and writes
    - Intercept store instructions
    - Inefficient in software

Requirements

- Data Bits Preserved in Codewords
  - Systematic code
- Small Overhead for ECC Bits
- Fast and Small Program
  - Encoding/decoding and correction
- Handling Multiple Error
- Background Task
- Self-Repair

Previous Work

- Encoding/Decoding for Comm. Systems
- Software Impl. of Error Detection Codes
  - Performance comparison
  - Multiple checksum schemes [1]
  - Computation of CRC via table look-up [2]
  - Quasi-cyclic vs. convolution code [3]
  - Byte-wide SEC-DED (255, 252) RS code [4]
    - RAM disks of satellite memories
ECC Protection in Software

Different Codes
- Block Code
  - Communication
  - Storage
- Horizontal Code
  - e.g., over a 64-bit word
  - Memory ECC in hardware
- Vertical Code
  - Bit-sliced
  - Flexibility in Software

Horizontal vs. Vertical Code
- Horizontal
  - Wider memory architecture
- Vertical
  - Bit-wise logical instr.
  - Multiple Bit Error
    - In a word
    - In a bit-slice

Coding Schemes
- Detection and Correction Capability
  - Single errors
  - Multiple errors
  - Errors in words, bit-slices
- Ease of Software Implementation
  - Code size and speed
- Overhead of Check Bits

Scheme 1 - Hamming
- (72, 64) Hamming Code
  \[ C[d_0, d_1, ..., d_{63}, c_0, c_1, ..., c_6] = D[d_0, d_1, ..., d_{63}] \times \left[ \begin{array}{c} \ldots \end{array} \right] \]
  \[ = \left[ \begin{array}{c} c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6 \ c_7 \end{array} \right] \times \left[ \begin{array}{c} d_0 \ d_1 \ d_2 \ d_3 \ d_4 \ d_5 \ d_6 \ d_7 \ldots \end{array} \right] \]
- SEC-DED
  - Independently for each bit-slice

Scheme 2 - Cyclic
- (72, 64) Cyclic Code: \[ P(X) = X^8 + X^7 + X^2 + 1 \]
- Polynomial Division
  - LFSR
- Software Implementation
  - 32 parallel LFSRs
  - SEC-DED
  - Independently for each bit-slice

Scheme 3 - Parity
- Vertical + Diagonal Parity
- Single Bit Error Correction
  - One in whole block of 32x32 bits
  - Double Bit Detection
ECC Protection in Software

Scheme 4 - RS
- RS Code, G(2^32): \( P(X) = X^{32} + X^{32} + X^2 + X + 1 \)
- \( d = n-k+1 \)
  - \( d=3: \) SbEC
  - \( d=4: \) SbEC-DbED

\[
\begin{align*}
\alpha_i &= \sum d_i \\
\alpha_i &= \sum d_i \alpha \\
\alpha_i &= \sum d_i \alpha^2 \\
\end{align*}
\]

Comparison

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Prog. Size (bytes)</th>
<th>Check-bit Overhead (%)</th>
<th>Performance (Dec. MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming</td>
<td>14307</td>
<td>12.5%</td>
<td>187.80</td>
</tr>
<tr>
<td>Cyclic</td>
<td>6731</td>
<td>12.5%</td>
<td>29.24</td>
</tr>
<tr>
<td>Parity</td>
<td>6747</td>
<td>6.25%</td>
<td>34.68</td>
</tr>
<tr>
<td>RS (d=3)</td>
<td>6723</td>
<td>3.125%*</td>
<td>24.41</td>
</tr>
</tbody>
</table>

* Block = 64 data + 2 check words

Multiple Error Handling
- Single Particle Hit
  - Multiple errors [5-7]
- ECC Software
  - Logical view of memory
  - Programmer’s view of bytes and addresses
- Mapping of Physical to Logical Bits
  - Important in code design
  - System design dependent
  - Memory structure dependent

System Level Dependency
- Memory Chip Data Width
  - Example:
    - 32-bit data bus
    - 2MB memory
      - 512K x 8-bit chips \( \times 4 \)
      - 512K x 1-bit chips \( \times 32 \)
- Errors in Diff. Chips
  - Independent

Memory Structure Dependency
- Physical Layout of Memory Cells
  - One or multiple arrays
  - Connection of address bits to the decoders
  - Mux layouts
  - Variable among designs

Memory Structure 1
- 512K x 8-bit

[Diagram of Memory Structure 1]
ECC Protection in Software

Memory Structure 2
- 4 \times 128K \times 8bit

Memory Structure 3
- 2 \times 256K \times 4bit

Interleaving
- Multiple Errors in Adjacent Addresses
  - e.g., 4-way interleaving

Implementation in ARGOS
- Periodic Scrubbing

Design Framework
- Multitasking
  - Separate, high priority task for ECC
- Memory Access Right to Code Segments
  - No protection in VxWorks
- Synchronization
  - Timer for periodic scrubbing
  - Message-passing for wake-up calls
- Automatic Addition of New Protected Blocks
  - Initialization code of each module
  - Block info sent to ECC by message

Cache Memory
- Cache Coherency in a Split Cache
- Instructions in D-cache
- Flush D-cache, invalidate I-cache
- Software ECC for Cache?
  - Supervisor mode; OS level
  - Direct access to data and tag
ECC Protection in Software

Error Recovery

- Error Detection
  - SSM, SAI, WD timer
- First Retry
  - Computation errors
- Second Retry
  - Code corrupted
  - Force ECC scrub
    - Message-passing
- Reload Module

Summary

- Hardware ECC
  - Recommended when possible
- ECC in Software
  - Provide protection for code segments
    - Coding schemes compared
- ARGOS Project
  - Continuous error collection (ECC and others)
  - Automatic recovery

Future Work

- Self-Checking and Correction for ECC Module
  - Different schemes
  - Reliability analysis
- More Efficient Schemes
- Observe Error Recoveries
  - Percentage of successful ones
  - Improvements

References (1)


References (2)