IIS5008 Hardware Security
Final Project (In Group)
Data Integrity with Checkpoint and ECC
(Due:2025/06/15 23:59:59)

Introduction

As computing systems become increasingly integrated into every aspect of modern life, ensuring the integrity and security of data has become important. In particular, memory systems are vulnerable to transient errors, hardware faults, and malicious tampering that can corrupt data during read and write operations. Such corruption can lead to system crashes, incorrect computations, or security issues.

In this project, we will modify Gem5's memory system to support ECC encoding on writes and decoding on reads, simulate hardware-induced memory errors, and analyze the trade-offs between reliability and performance overhead. By comparing single-bit correction by Hamming code and multi-symbol correction by Reed–Solomon code. You will have hand-on experience in hardware security techniques and understand how different ECC algorithms affect system behavior under fault conditions.

Background

1. Error-Correcting Codes (ECC)

   Error-correcting codes are algorithms that add structured redundancy to digital data to detect and correct errors that occur during storage or transmission. By encoding original data with additional parity or check symbols, ECC schemes allow the recovery of corrupted bits or symbols without the need for retransmission. In memory systems, ECC is critical for protecting against soft errors caused by cosmic rays, electrical noise, and other transient faults, thereby improving system reliability and uptime.

2. Hamming Code

   Hamming codes are a family of linear block codes that provide single-bit error correction. Introduced by Richard Hamming in 1950, a Hamming (n,k) code uses n total bits to encode k data bits along with (n−k) parity bits placed at specific positions. The arrangement enables the detection and localization of erroneous bit positions through syndrome computation. Hamming codes are simple to implement in hardware, require low overhead, and are widely used in applications where single-bit error correction is sufficient.

3. Reed–Solomon Code

Reed–Solomon codes are non-binary cyclic block codes that operate on symbols over a Galois field, allowing the correction of multiple symbol errors. A Reed–Solomon (n,k) code encodes k data symbols into n total symbols by adding (n−k) parity symbols. The error-correction capability depends on the number of parity symbols. It can correct up to ⌊(n−k)/2⌋ symbol errors. Reed–Solomon codes are widely deployed in storage devices, communication systems, and Linux, offering robust protection against burst errors and symbol-level corruption.

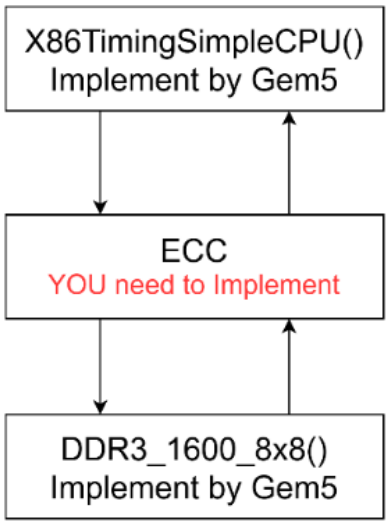
Problem Description

In this project, you will build Gem5, the CPU simulator on your own Linux machine. You will then implement two or three ECC modules in Gem5, including a Hamming-code based ECC module and a Reed–Solomon ECC module. Note that you must not use any existing libraries for encoding or decoding.

To simulate hardware errors, you need to implement a module that generates error bits. The Hamming-code module should correct up to one bit error. The Reed–Solomon module's error-correction capability depends on the number of parity symbols. For this project, configure it to tolerate up to two symbol errors for any data width. After, compare their performance in terms of CPU cycles.

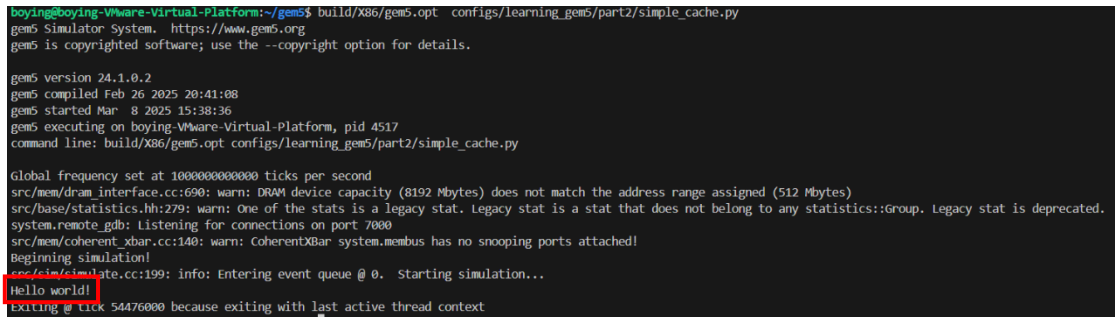
We will test your modules by executing "Hello World" at "/gem5/tests/test-progs/hello/bin/x86/linux/hello". If your modules are implemented correctly, the program will run successfully. Otherwise, it may produce undetermined behavior.

Important note: In this project, errors occur only during memory reads. Data stored in memory is assumed to be correct, writes do not introduce errors.



- Step 1: Setting Up Gem5

  Install Gem5 and run a basic simulation to ensure the setup is correct. And can execute “Hello World”.



- Step 2: Software simulation

  Implement Hamming Code and Reed-Solomon encoder and decoder in C++. And explain their dataflow and paste the program execution results in the report. In this project Reed-Solomon encoding should base on GF(2^8).

  Encoding example:

  1. Hamming Code Example:
     Input: 0x058B420000000000
     Parity: 0x73 (115)
  2. Reed-Solomon(12,8) based on G(2^8) example:
     Input: 0x0123456789ABCDEF
     Parity: 0x2106A681
     Input: 0xDEADC0DE1234ABCD
     Parity: 0xA7294CEF

- Step 3: Designing the Hamming Code ECC Module

  Design a simple ECC module capable of detecting and correcting single-bit errors by Hamming Code. And paste the program execution results in the report. This module should be a C++ class.

- Step 4: Designing the Reed-Solomon ECC Module:

  Design a simple ECC module capable of detecting and correcting 2-bit errors by Reed-Solomon(12,8). And paste the program execution results in the report. This module should be a C++ class.

- Step 5: Integrating ECC into Gem5

  Modify Gem5's memory system to integrate the ECC module. You also need to implement a module that generates a signal bit/2-symbols error with a 5% error rate. This involves modifying memory read/write operations to include ECC encoding and decoding. You should integrate Hamming code ECC module and Reed-Solomon Code ECC module in the same project and using them separately by different config file. TA will execute the "Hello World" program by your config file, then paste the "Hello World" execute result in report.

- Step 6: Simulation

  Run simulations with the modified Gem5 to evaluate the impact of ECC on memory reliability and system performance. You should introduce errors into memory accesses and demonstrate how the ECC module (RS and Hamming) detects and corrects these errors.

  You need to demonstrate how the ECC module detects and corrects errors during simulation in the report. For example:

```
554000: system.memobj: ----------------------------handleRequest-----------------------------
575000: system.memobj: Got response for addr 0x94d60
625000: system.memobj: message in binary format is 104 237 255 255 255 127 0 0
625000: system.memobj: randomFlip in binary format is 104 74 255 255 255 127 0 236
625000: system.memobj: The data after random flip is 104 74 255 255 255 127 0 236
625000: system.memobj: The data is wrong
625000: system.memobj: The data in binary format is 104 74 255 255 255 127 0 236
625000: system.memobj: The parity is 253 237 47 113
625000: system.memobj: The eccStore is 107 24 93 43
625000: system.memobj: The true parity is 727521387 107 24 93 43
625000: system.memobj: The data after correct is 104 237 255 255 255 127 0 0
```

  And make a chart comparing the execution time before and after integrating the ECC module, using data from "gem5/m5out/stats.txt". And discuss the impact of ECC on system performance and reliability in the report. Chart example:

Name	Without ECC	RS	Hamming
simSeconds			
simTicks			

- Bonus: More ECC algorithms!

  Based on research papers on error-correcting codes (other than Hamming Code and Reed-Solomon), please implement an ECC module capable of detecting errors affecting more than two bits. And write the paper title and provide a brief explanation of the encoding method in the report.

Report

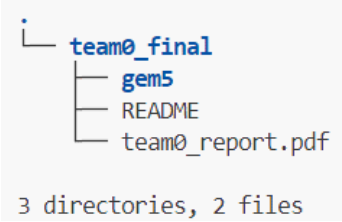
Your report must contain the following contents, and you can add more as you wish.

1. Team members' names and student IDs.
2. Explain how the Hamming Code and Reed–Solomon Code algorithms work.
3. The result for each part. DO NOT paste all the code in the report but you can paste piece of code in need.
4. How your team divide the work.
5. What have you learned from this project? What problem(s) have you encountered in this project?
6. Suggestions and feedback for this project.

Submission Requirement

You have to write this program in C/C++ and any form of plagiarism is strictly prohibited. If you refer to any external materials, you must cite them properly.

Compress all the item with the name team{ID}_final.tar.gz before uploading it to eeclass. And the folder structure would be like the following. Otherwise, your program will not be graded.



1. gem5: All the file in the gem5 folder, but without gem5/build folder. TA will build your gem5 at server.
2. README: In README need to contain two parts.
    1. How to compile and execute your config file including Hamming Code ECC module and Reed-Solomon Code module.
    2. The file you modified or added.
3. teamID_report.pdf

You can use the following command to compress your directory on a workstation:

$ tar -zcvf team{ID}_final.tar.gz <directory>

For example:

$ tar -zcvf team0.tar.gz team0_final/

Grading

The grading is as follows:
Step 1(25%): Success build the gem5 and show the “Hello World”.

Step 2(10%): Success software simulate Hamming Code (5%) and Reed–Solomon Code (5%) and show the result.

Step 3(10%): Success build the Hamming Code ECC module and show the test results.

Step 4(10%): Success build the Reed-Solomon ECC module and show the test results.

Step 5(25%): Success integrate Hamming Code ECC module and Reed-Solomon ECC module in gem5 and show them can execute “Hello World” separately and correctly.

Step 6(10%): Post the debug message, complete the chart and compare their differences.

Report (10%): Base on the completeness of report.

Bonus (10%): Success integrate another ECC module in gem5 and show them can execute “Hello World” separately and correctly.

No late submissions will be accepted for the final project; please submit it on time.

Contact

For all questions about PA3, please send Email to TA 王柏穎. Email to 1546298@gmail.com with the title [HW Security Final].

Reference

[1] Gem5 simulator source code: https://github.com/gem5/gem5

[2] Gem5 documents:
https://www.gem5.org/documentation/learning_gem5/introduction/

[3] Example:
https://www.gem5.org/documentation/learning_gem5/part2/memoryobject/

[4] Reed-Solomon calculator:
https://repo.progsbase.com/repoviewer/no.inductive.libraries/ReedSolomon/latest/
//ComputeReadSolomonCodes/online/