

Arhitectura Calculatoarelor

32-bit Hamming Error Correction Code Module

Documentație

Efrem Dragoș-Sebastian-Mihaly
Grupa 3.1

Scurtă introducere în temă

De-a lungul anilor, mesajele au putut fi transmise și receptate. În zilele noastre, depindem foarte mult de informațiile stocate în mediul digital, spre exemplu informațiile stocate pe un CD. Cu toate acestea, există momente când informația receptată să nu coincidă cu cea emisă, adică momente în care informația pe care o avem conține erori. Încercăm să găsim modalități de detectare a acestor erori digitale, eventual corectându-le. O metodă care ne poate ajuta în acest scop a fost descoperită de Richard W. Hamming în 1950, în timpul cercetărilor acestuia la Bell Labs. Metoda descoperită de Hamming devine un bloc important fundațional pentru era calculatoarelor moderne și a comunicării digitale.

Prezentarea structurii modului principal

Principalul modul a fost împărțit într-un modul de detecție și un modul de corecție. În mod abstract, schema circuitului poate fi văzută ca fiind următoarea:

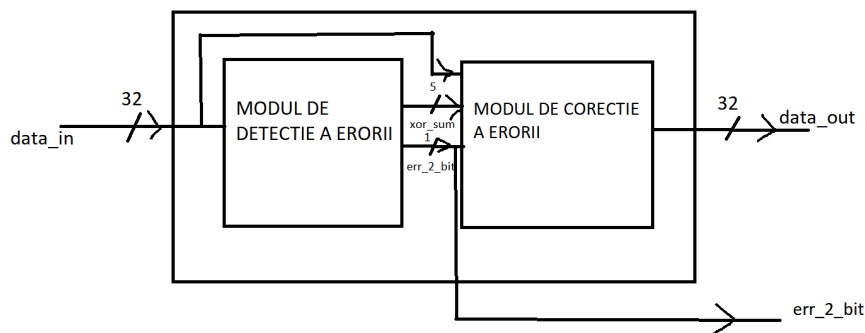


Figura 1: Schema generală pentru ECC-ul implementat

Detaliem, acum, fiecare modul în parte.

Modulul de detecție a erorii

Modulul de detecție a erorii trebuie să fie capabil să furnizeze dacă există o eroare sau dacă există două erori. Acesta primește ca intrare data_in pe 32 biți, iar la ieșire furnizează err_2_bit pe 1 bit, respectiv xor_sum pe 5 biți, adică suma tuturor pozițiilor din numărul binar pentru care bitul de pe acea poziție este 1. Motivul este următorul: în cazul niciunei erori, xor_sum va fi 00000, iar în cazul unei erori, xor_sum va reprezenta exact poziția în binar a bitului flipuit. err_2_bit se determină în următorul mod: dacă suma tuturor biților de la intrare este pară, și avem xor_sum o valoare diferită de 00000, atunci err_2_bit va fi 1. Pentru restul cazurilor, acesta va fi 0.

Implementarea folosită în realizarea modulului de detecție a erorii a fost una bazată atât pe circuite combinatoriale cât și circuite secvențiale ¹. Practic, lucrăm cu un clock și reset interni ai modulului (semnal de tact predefinit) date de un Signal Generator care dirijează toate registrele folosite într-o manieră care duce la îndeplinirea funcției sistemului. Signal Generator aplică la momentul inițial reset tuturor registrelor din arhitectură. Semnalul de tact a fost ales astfel încât ieșirile să se calculeze în mod optim, perioada semnalului de tact fiind de doar 2ns. Mai mult, pentru parcurgerea fiecărui bit de la intrare folosesc un multiplexor 32x1, având biții de selecție dirijați de ieșirea unui registru a cărui intrare se incrementează cu 1 la fiecare front crescător al semnalului de tact intern. Intrarea se modifică cu 1 cu ajutorul unui RCA pe 5 biți modificat, care are o intrare mereu setată pe 1. De asemenea, folosim 4 flip flopuri de tip D ca elemente de memorare de informații importante a modulului, cum ar fi: biții de selecție pentru multiplexor, suma tuturor biților de la intrare, calculul bitului de eroare, respectiv calculul sumei tuturor pozițiilor pentru care bitul de pe acea poziție de la intrare este 1. O schemă a modulului este următoarea:

¹Am ales această implementare, care poate părea mult mai complicată, deoarece am considerat util să combin tot ce am învățat până acum în realizarea acestui proiect

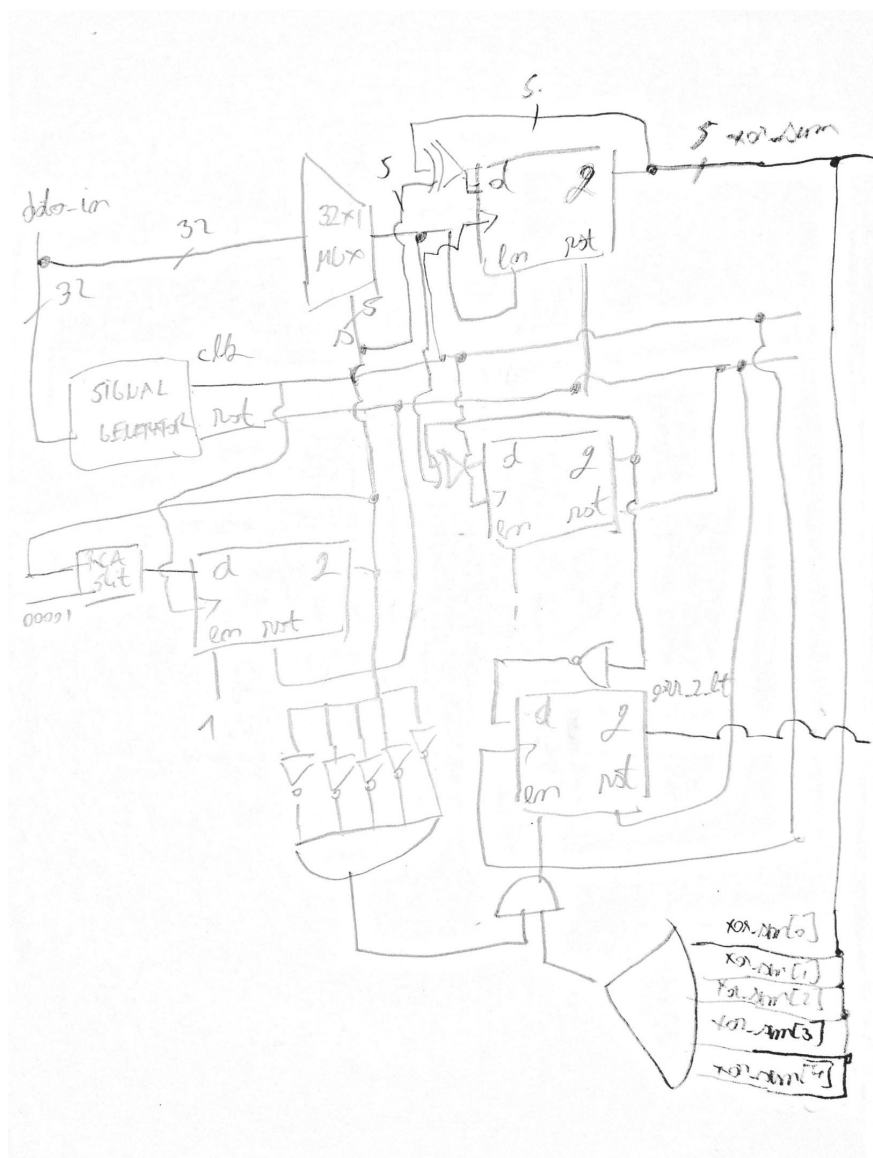


Figura 2: Schema pentru modulul de detecție a erorii implementat

Modulul de corecție a erorii

Modulul de corecție a erorii trebuie să fie capabil să furnizeze la ieșire `data_in` dar modificat, în cazul în care acesta are 1 bit flipuit. Acest modul primește ca intrare `xor_sum` pe 5 biți și `err_2_bit` pe 1 bit de la modulul de detecție și, corectează intrarea pe baza unui raționament corect, cum ar fi acela că intrarea este corectată numai dacă `err_2_bit` este 0 și `xor_sum` este diferit de 00000, în restul cazurilor nemodificând intrarea.

Implementarea folosită în realizarea modulului de corecție a erorii a fost una bazată pe un circuit simplu combinațional, cu un codicator 5x32 ce are intrare de validare. `xor_sum` aduce un ajutor imens în acest sens, întrucât el ne arată poziția erorii. Prin urmare, la intrarea codicatorului vom avea `xor_sum`, și ca bit de validare vom avea pe `NOT(err_2_bit)` dar și un șir de porți SAU între

biții lui `xor_sum`. Dacă condiția de validare este acceptată, avem la ieșirea codicatorului un sir de 32 de biți în care doar un bit este 1, adică bitul care se afla pe poziția dirijată de `xor_sum`. Prin urmare, cu un simplu XOR, putem modifica `data_in`. O schemă a modului este următoarea:

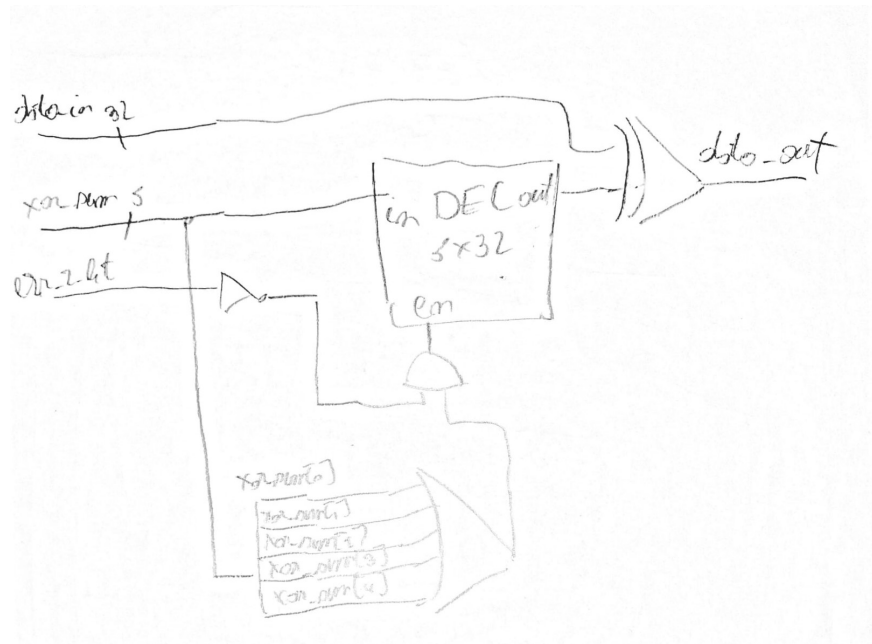


Figura 3: Schema pentru modulul de corecție a erorii implementat

În testbench, trebuie să se ofere un timp necesar între modificările variabilei de intrare. Variabila de intrare nu se poate modifica decât după aproximativ 65 ns de când a fost aplicată o altă intrare, pentru a permite un timp necesar circuitelor secvențiale de realizare a funcțiilor esențiale.

Bibliografie

- [1] Richard Hamming (1997) The Art of Doing Science and Engineering.
- [2] 3Blue1Brown. (4 sept. 2020). How to send a self-correcting message (Hamming codes) [Video]. YouTube. URL <https://www.youtube.com/watch?v=X8jsijh11IA>
- [3] 3Blue1Brown. (4 sept. 2020). Hamming codes part 2, the elegance of it all [Video]. YouTube. URL https://www.youtube.com/watch?v=b3NxrZOu_CE