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 transmite si receptate. In 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, adica momente în care informația pe care o avem conține erori. Încercăm să gasim 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 in 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 modulului principal

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

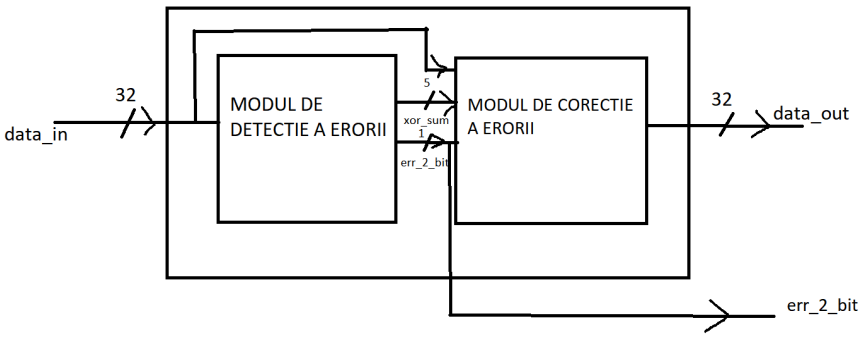

Figura 1: Schema generală pentru ECC-ul implementat

Modulul de detecție al ECC ne oferă informații despre *data_in*, mai exact dacă s-a descoperit o eroare sau 2 erori. Această informație este utilizată mai departe de modulul de corecție, ce corecteaza eroarea flipuind bitul eronat daca a existat doar o eroare, respctiv nemodificând intrarea dacă au existat 2 erori.

Detaliem, acum, fiecare modul in 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: daca suma tuturor biților de la intrare este pară, și avem xor_sum o valoare diferita de 00000, atunci err_2_bit va fi 1. Pentru restul cazurilor, acesta va fi 0.

Implementarea folosita în realizarea modulului de detecție a erorii a fost una bazată atât pe circuite combinationale 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 maniera 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 incrementeaza 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.

Un fapt mai interesant este alegerea perioadei semnalului de tact. S-ar putea să pară interesant, dar semnalul de tact are perioada $T = 33ns$. Răspunsul nu e prea intuitiv, deoarece lucram cu un număr pe 32 de biți, si totusi am crede ca sunt de ajuns 32 de fronturi crescătoare pentru a opera pe toți biții din acel număr. Motivul este urmatorul: la primul front crescător de tact, operam pe primul bit al intrării, adică bitul 0. Prin urmare, avem rezultatele *înaintea* prelucrării bltului 0, ce e drept, cele default, în urma resetului. Prin urmare, la al 32-lea front crescator de tact, operam cu ultimul bit, dar avem doar rezultatele până la ultimul bit. Astfel, abia la al 33-lea front crescator de tact, când variabila de selecție pentru multiplexor are overflow (trece iară la 00000), abia atunci putem spune că am facut toate operațiile necesare. În implementare, am făcut uz de acest aspect pentru a calcula bitul ce ne spune dacă există 2 erori. Acesta poate fi calculat numai atunci cand am ajuns din nou cu biții de selecție la 00000, pentru că doar atunci avem variantele finale pentru xor_sum și pentru bitul de paritate pentru toate blocurile.

De notat faptul că Signal Generator are o intrare *data_in*. Motivul este acela de a putea da la intrarea ECC-ului mai multe valori la intervale de timp și obținerea de răspunsuri corecte pentru fiecare în parte. Practic, de fiecare dată cand semnalul de la intrare se modifică, atunci Signal Generator se resetează (un reset global), resetând semnalul de tact si cel de reset local. Odată ce resetarea globală a avut loc, pentru noua intrare de date se realizează exact aceleasi operații sub exact același clock si semnal de reset ca pentru intrarile trecute.

[1] Am 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, dar și pentru că o astfel de implementare mi s-a parut interesantă din punctul de vedere al realizabilității.

O schemă a modulului de detecție este următoarea:

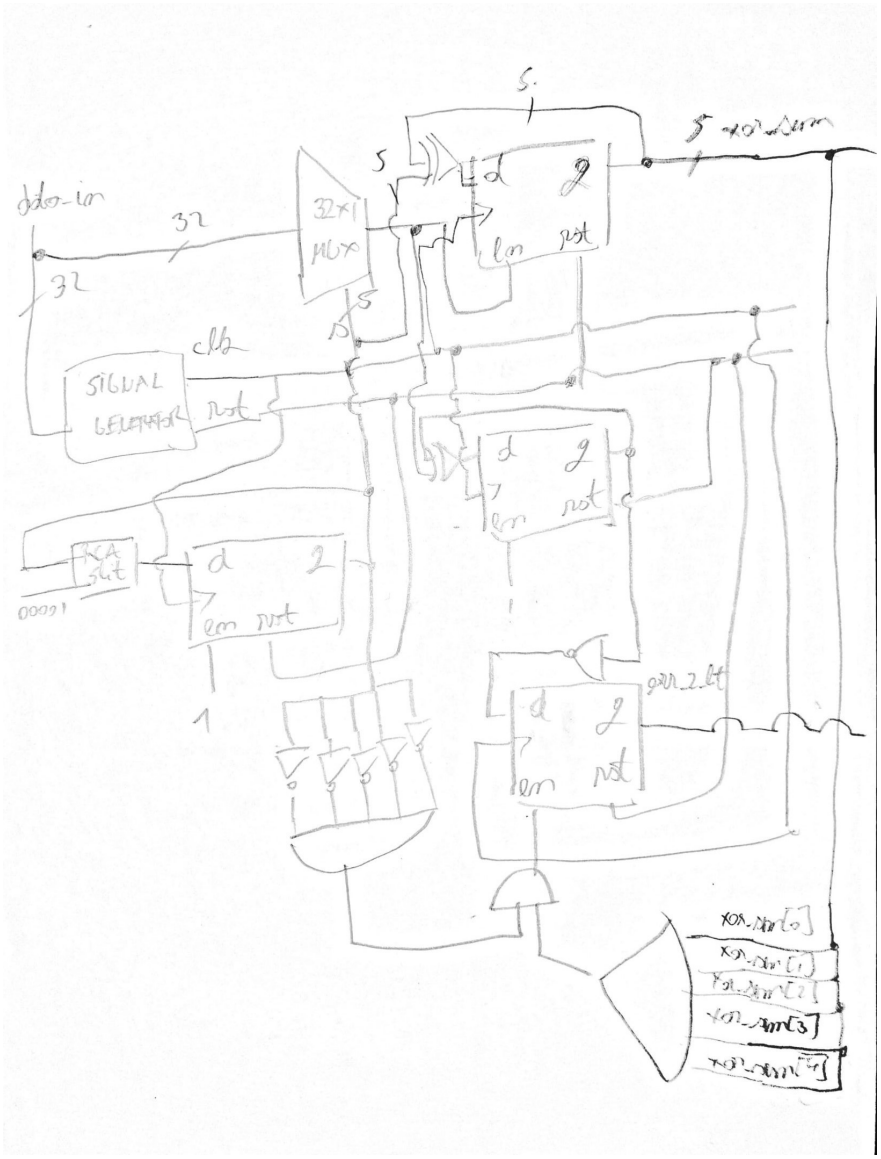

Figura 2: Schema pentru modulul de detecție a erorii implementat folosind regiștri, generatoare de semnal, multiplexoare, sumatoare și porți logice primitive

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 nemodificand intrarea.

Implementarea folosită în realizarea modulului de corecție a erorii a fost una bazată pe un circuit simplu combinațional, cu un decodificator 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 decodificatorului 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 decodificatorului 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. Un aspect interesant poate fi considerat prezența biților lui xor_sum în schema de mai jos, si realizarea operației SAU pe ei. Acest fapt se realizează, in principiu, datorită modulului în care am implementat decodificatorul. Dacă intrarea de validare este activă, și xor_sum este 00000, iesirea de la decodificator va avea un 1 pe poziția cea mai semnificativă și un 0 în rest. Cu toate acestea, dacă am 00000 înseamnă că nu am eroare, deci teoretic ieșirea decodificatorului ar trebui sa fie alcătuită numai din biți de 0, ca să nu modifice intrarea. Astfel, verificarea biților lui xor_sum a fost necesară.

O schemă a modulului de corecție este următoarea:

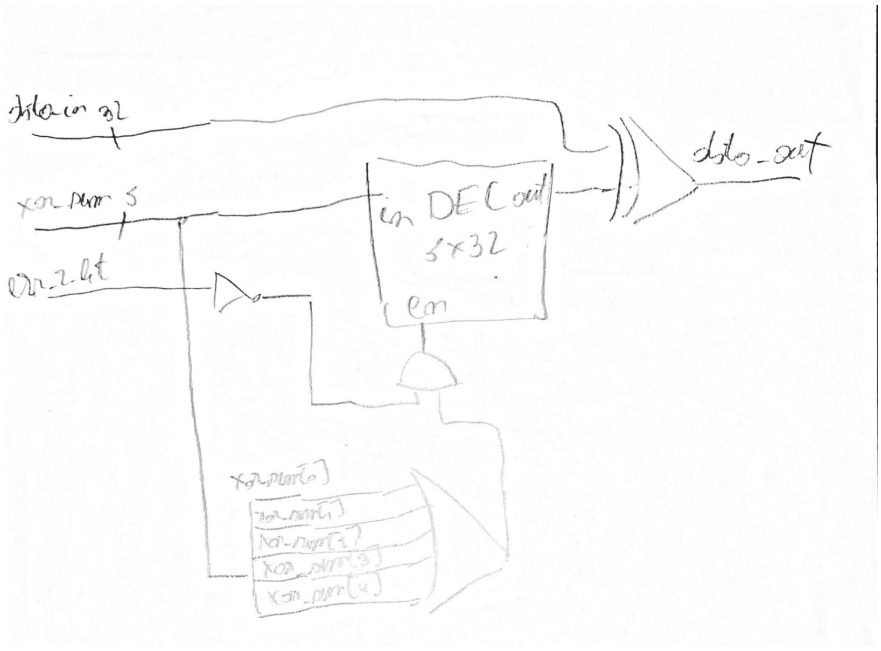

Figura 3: Schema pentru modulul de corecție a erorii implementat folosind decodificatoare și porți logice primitive

Testarea aplicației

În test bench, trebuie sa se ofere un timp necesar intre modificările variabilei de intrare. Variabila de intrare nu se poate modifica decât după aproximativ 66 ns de când a fost aplicată o altă intrare, pentru a permite un timp necesar circuitelor secvențiale de realizare a funcțiilor esențiale. În test bench-ul folosit de mine, am folosit un delay de 100 ns.

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=X8jsijhllIA`

[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`