FAULT TOLERAN UNTUK NANOSCALE MEMORY MENGGUNAKAN REED SOLOMON CODE

Zaiyan Ahyadi (1)

(1) Staf Pengajar Teknik Elektro Politeknik Negeri Banjarmasin

Ringkasan

Industri chip silikon berlomba mengikuti hukum Moore, yang menjadikan ukuran transistor tambah mengecil menuju orde nano meter yang disebut dengan nanoelektronik. Peralatan pertama yang diprediksi bakal menerapkan teknologi nanolektronik adalah memori, karena keteraturan struktur sel memori memnyebabkan memori lebih mudah dibuat dalam ukuran nanometer, yang disebut dengan nano memori.

Ukuran transistor atau peralatan chip silikon lainnya yang sangat kecil menyebabkan fenomena fisis pada chip silikon tadi akan berubah. Hal ini berarti juga akan mengubah karakteristik dari memori yang terbuat dari chip silikon. Salah satu fenomena tersebut adalah mudahnya nano memori terkena dampak paparan dari partikel kosmis yang menyebabkan error yang tidak permanen, yang disebut sebagai soft error.

Tulisan ini mengusulkan suatu teknik fault toleran untuk mitigasi soft error dengan menggunakan kode Reed Solomon. Teknik ini mempunyai karakteristik mampu memperbaiki error yang terjadi dalam suatu blok pada suatu kode data.

Kata Kunci *: Reed Solomon, memory fault tolerant, error correction code*

1. PENDAHULUAN

Latar Belakang

Kemajuan elektronika tidak dapat dipisahkan dengan ditemukannya transistor dan chip silikon. Tujuan utama dari desain chip silikon adalah bagaimana agar hasil desain mempunyai area yang kecil dan *critical path* yang pendek. Berkaitan dengan hal yang pertama, Gordon Moore salah satu pendiri perusahan Intel mengemukakan sebuah pendapat yang akhirnya menjadi suatu aturan yang disebut hukum Moore. Hukum ini menyatakan bahwa jumlah transistor dalam suatu chip akan menjadi dua kali lipat setiap dua tahun[1]. Semua industri peralatan elektronik berusaha untuk mengikuti hukum ini, karena jika tidak, maka akan menyebabkan seolah-olah produk mereka ketinggalan zaman. Ukuran transistor atau peralatan silikon sekarang ini yang mempunyai ukuran dalam orde mikrometer diprediksi akan terus menyusut menjadi ukuran dalam orde nano meter.

Memori adalah merupakan salah satu peralatan chip silikon yang akan segera memulai menerapkan ukuran dalam orde nanometer. Hal ini disebabkan sel-sel memori mempunyai struktur internal yang teratur sehingga memberikan kemudahan untuk dibuat sangat kecil.

Namun mengecilnya ukuran transistor ini hampir mencapai titik jenuh atau jalan buntu (*face the brick wall*)[2]. Karena mengecilnya ukuran suatu chip menyebabkan perubahan fenomena fisis yang terjadi di dalamnya. Ini berarti sifat-sifat fisis peralatan silikon dengan ukuran mikrometer akan berbeda ketika ukurannya menjadi nanometer. Salah satunya adalah fenomena soft error[3] yang telah ditemukan pada chip memori dengan ukuran sel dibawah 100 mikrometer.

Soft error adalah error yang terjadi karena terpapar oleh suatu partikel dengan kecepatan tinggi, yang bisa menyebabkan suatu sel berubah nilai logika dari satu menjadi nol dan sebaliknya. Error ini bersifat tidak permanen, artinya ketika sistem dimatikan kemudian dihidupkan kembali error tersebut belum tentu terjadi lagi. Hal ini karena error tidak disebabkan oleh kerusakan fisik. Error terjadi karena ukuran sel yang sangat kecil sehingga memudahkannya untuk berubah logika.

Beberapa metode telah ditawarkan untuk mitigasi soft error. Mulai dari perbaikan teknik produksi, penambahan layer pelindung dan sampai modifikasi arsitektur sel memori[3].

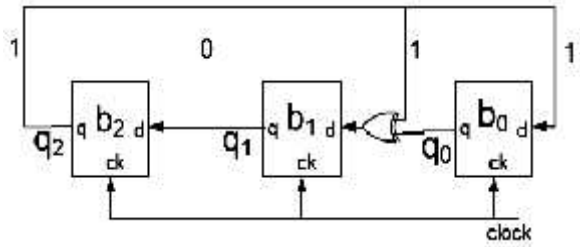
Pengubahan sistem kode data dari biner biasa menjadi kode Reed Solomon adalah salah satu metode fault tolerant yang ditawarkan. Penulis mencoba menggali metode ini dengan mencoba mendesain encoder dekoder Reed Solomon untuk nano-scale memori dengan menggunakan alat bantu VHDL dan FPGA.

2. TINJAUAN PUSTAKA

Kode Reed Solomon (RS) diperkenalkan oleh Reed dan Solomon pada tahun 1960[4].

Kode ini merupakan kode khusus dari kode yang lebih umum yaitu kode Bose-Chaudhuri dan Hocquenghem (BCH) yang ditemukan tahun 1959. Berikut ini akan dipaparkan RS dengan pendekatan secara praktikal daripada pendekatan secara teori.

Kode RS adalah kode yang berdasarkan himpunan terbatas (*finite-field*) yang disebut juga Galois Field (GF), yaitu suatu field yang mempunyai anggota (elemen) yang terbatas. Kode RS menggunakan notasi *polinomial* GF(2^m) dengan m adalah bilangan bulat positip. Seluruh elemen dalam field dihasilkan oleh *primitive-polynomia*l. Polinomial primitip adalah polinomial yang tidak bisa difaktorkan menjadi polinomial lain. Salah satu contoh polinomial primitip adalah GF(2^3) yaitu $x^3 + x + 1 = 0$. Rangkaian dari pembangkit polinomial primitip ini adalah diperlihatkan oleh gambar 1[5]



Gambar 1. Rangkaian pembangkit polinomial primitip GF(2^3)

Dari rangkaian pada gambar 1, didapatkan elemen-elemen dari GF(2^3). Setiap siklus clock akan menghasilkan elemen yang direpresentasikan sebagai pangkat dari α. Pada gambar α mempunyai nilai biner yang ditentukan oleh status q_2,q_1,q_0. Sebagai nilai awal α^0 adalah 001. Jika didefinisikan $\alpha^n= [q_2,q_1,q_0]$, dimana $[q_2,q_1, q_0]$ didefinisikan sebagai seperti pada gambar 1, maka untuk $\alpha^{n+1} = [q_1,q_2 \oplus q_0,q_2]$. Setiap elemen dari GF adalah pangkat dari polinomial primitip. Elemen yang lengkap dari GF(2^3) dapat dilihat pada tabel 1.

Operasi aritmetika terhadap elemen finite field akan memberikan hasil dalam range yang sama dari finite field tersebut juga. Tabel 1, memperlihatkan $\alpha^2\alpha^3$ akan menghasilkan α^n , dimana $0<n<6$. Operasi aritmetika yang penting untuk RS error correction code adalah perkalian, pembagian, penjumlahan dan pengurangan yang akan dipaparkan berikut ini.

Perkalian dan Pembagian

Secara umum operasi perkalian atau pembagian dikerjakan dengan menambahkan atau mengurangkan nilai pangkat dari α. Penting untuk dicatat bahwa $n=2^m-1$, maka $\alpha^n = \alpha^0=1$. Sebagai contoh GF(2^3), $n=2^3-1=7$, sehingga $\alpha^7=1 =0$. Hal ini sangat berguna untuk menyederhanakan operasi matematika.

Sebagai contoh GF(2^3).

Perkalian

$\alpha^2\alpha^4 =\alpha^6$

$\alpha^5\alpha^6=\alpha^{11} = \alpha^7\alpha^4=(1)\ \alpha^4=\alpha^4$

Pembagian

$\alpha^4/\alpha^2 =\alpha^2$

$\alpha^2\alpha^4=(1)\alpha^2/\alpha^4 = \alpha^7\alpha^2/\alpha^4=\alpha^5$

Pada implementasinya agar lebih efesien nilai pangkat dari alpha disimpan sebagai index dalam bentuk Tabel Look Up.

Tabel 1. Tabel elemen dari field GF(2^3)

Power	Binary value	Decimal value
-	000	0
$\alpha^0=1$	001	1
α^1	010	2
α^2	100	4
α^3	011	3
α^4	110	6
α^5	111	7
α^6	101	5

Penjumlahan dan Pengurangan

Operasi penjumlahan dan perkalian pada fininite filed bekerja dengan cara yang sama yaitu operasi bitwise XOR ($\oplus$)atas nilai logika biner dari elemen finite field yang dioperasikan.

Contoh untuk GF(2^3)

Penjumlahan

$\alpha^5+\alpha^6 =111\oplus101=010 = \alpha^1$

$\alpha^2+\alpha^7 =100\oplus001=101 = \alpha^6$

Penjumlahan

$\alpha^5-\alpha^6 = \alpha^5+\alpha^6 =111\oplus101=010 = \alpha^1$

$\alpha^2-\alpha^7 = \alpha^2+\alpha^7 =100\oplus001=101 = \alpha^6$

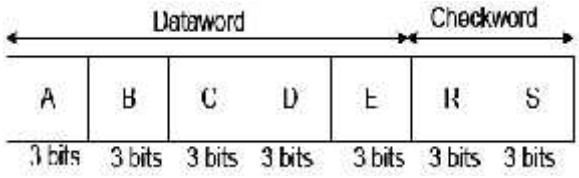
Kedua operasi penjumlahan dan pengurangan di atas adalah benar karena pada sistem bilangan biner operasi penjumlahan dan pengurangan bit akan menghasilkan nilai yang sama. Contoh 1+1=1-1=0, 1+0=1-0=1.

Kode RS bekerja menekankan bekerja pada simbol daripada bekerja pada bit. Sebuah RS codeword terdiri dari bagian simbol dataword dan bagian simbol checkword. Suatu simbol tidak lain adalah elemen dari finite field dari GF(2^m). Suatu RS code yang komplit terdiri dari 2^m-1 simbol. Sehingga jumlah bit pada RS yang komplit adalah m x (2^m-1). Parameter dan notasi untuk GF(2^m) dengan $m\geq3$ adalah sebagai berikut:

Banyaknya bit per simbol	*m*
Kemampuan memperbaiki error	*t*
Banyaknya bit per dataword	k_{bit}

Banyaknya simbol dataword $k=k_{bit}/m$
Banyaknya bit per checkword $j=2xt$
Banyaknya simbol checkword $j_{bit}=(2xt)xm$
Banyaknya simbol codeword $n=j+k$
Banyaknya bit per codeword $n_{bit}=nxm$

Gambar berikut memperlihatkan dataword dan checkword untuk RS code dengan GF(2^3) dengan kemampuan memperbaiki 1 blok error. Pada gambar, jumlah simbol pada codeword adalah n=7, dimana A, B, C, D dan E adalah simbol dataword sedangkan R dan S adalah simbol untuk checkword. Setiap simbol terdiri dari m=3 bit, sehingga total bit dalam codeword adalah 3x7 = 21 bit. Kode RS ini mempunyai kemampuan memperbaiki error sebanyak t=1 simbol.



Gambar 2. Codeword Reed Solomon dengan GF(2^3)

Reed Solomon Encoding

Seperti yang dijelaskan sebelumnya, sebuah simbol terdiri dari beberapa bit, yang panjangnya ditentukan oleh finite field yang digunakan. Algoritma untuk menghitung check-word dari dataword untuk membentuk sebuah codeword adalah sebagai berikut[2]:

1. Tentukan finite field GF yang akan digunakan. Hal ini tergantung dari jumlah bit yang akan digunakan untuk membentuk sebuah simbol. Hasil dari langkah ini adalah banyaknya simbol codeword, dataword dan checkword.
2. Bentuklah sebanyak *j* buah persamaan polinomial yang ortogonal dari codeword, dimana *j* adalah banyaknya blok checkword. Hal ini dapat dilakukan dengan mengalikan setiap simbol dengan koefesien yang berbeda untuk setiap simbol. Karena koefesien adalah elemen dari GF, cara yang mudah adalah dengan menaikkan pangkat dari koefesian α secara sekuensial untuk setiap simbol menggunakan sebuah konstanta dalam range[1, 2^m-1].
3. Selesaikan persamaan dengan cara substitusi atau eliminasi untuk mendapatkan rumus untuk semua simbol checkword. Pada langkah ini, digunakan operasi perkalian, pembagian, penjumlahan dan pengurangan untuk finite field.
4. Gabungkan chekword dengan dataword untuk mendapatkan codeword. Untuk codeword yang sistematik checkword diletakkan pada posisi kanan dataword.

Agar lebih jelasnya bagaimana proses encoding dengan algoritma di atas berikut ini akan diberikan contoh. Sebanyak 15 bit data input akan di-*encode* menjadi sebuah codeword RS. Kemampuan koreksi errornya di set t=1 blok, sehingga checkword harus terdiri dari 2 simbol. Untuk contoh ini dipilih GF(2^3) yang berarti satu simbol direpresentasikan oleh 3 bit. Sehingga data input akan diubah menjadi k = t/m = 15/3= 5 simbol dataword yang diberi nama A, B, C, D, dan E. Serta 2 buah simbol checkword yang diberi nama R dan S. Sehingga total simbol codeword adalah 7 buah simbol dengan panjang total 21 bit.

Dari simbol A, B, C, D, E, R dan S akan dibangun dua buah persamaan ortogonal. Banyak cara untuk melakukan ini. Salah satunya adalah sebagai berikut:

$A + B + C + D + E + R + S = 0$ (2.1)

$\alpha^1A+\alpha^2B+\alpha^3C+\alpha^4D+\alpha^5E+\alpha^6R+\alpha^7S =0$ (2.2)

Pada persamaan 2.2 simbol pertama dikalikan dengan koefesien α dan simbol berikutnya dikalikan dengan α^1, α^2, ..., α^7. Susun ulang persamaan 2.1, dan ingat terhadap operasi penjumlahan modulo 2, sehingga menjadi sebagai berikut:

$S= A + B + C + D + E + R$ (2.3)

Substitusi ke dalam persaman 2.2

$\alpha^1A + \alpha^2B + \alpha^3C + \alpha^4D + \alpha^5E + \alpha^6R + \alpha^7 (A + B + C + D + E + R) =0$ (2.4)

Selesaikan, dan ingat $\alpha^7=\alpha^0=1$

$(\alpha^6+1)R= (\alpha^1+1)A + (\alpha^2+1)B + (\alpha^3+1)C + (\alpha^4+1)D + (\alpha^5+1)E$

Sederhanakan penjumlahan

$(\alpha^2)R= (\alpha^3)A + (\alpha^6)B + (\alpha^1)C + (\alpha^5)D + (\alpha^4)E$

Dibagi dengan α^2

$R= \alpha A + \alpha^4B + \alpha^6C + \alpha^3D + \alpha^2E$

Dapatkan S dengan mensubstitusi balik R pada persamaan 2.1 dan selesaikan S

$S= A + B + C + D + E + R + \alpha A + \alpha^4B + \alpha^6C + + \alpha^3D + \alpha^2E$

$S=(\alpha+1)A +(\alpha^4+1)B +(\alpha^6+1)C +(\alpha^3+1)D + (\alpha^2+1)E$

$S=\alpha^3A + \alpha^5B + \alpha^2C + \alpha D + \alpha^6E$

Langkah terakhir adalah menggabungkan ceckword dengan dataword untuk membentuk codeword. Seperti yang ditunjukkan pada gambar 2.6. Sebagai contoh sebuah data 15 bit 110000010100111, yang berarti A=110=α^4, B= 000=0, C=010= α^1,D=100=α^2 dan E=111=α^5. Dari persamaan 2.3 dan 2.4, checkword R dan S didapatkan sebagai berikut:

$R= \alpha A + \alpha^4B + \alpha^6C + \alpha^3D + \alpha^2E$

$R= \alpha\alpha^4 + \alpha^40 + \alpha^6\alpha^1 + \alpha^3\alpha^2 + \alpha^2\alpha^5$

$R= \alpha^5 + 0 + \alpha^7 + \alpha^5 + \alpha^7$

$R= 111 \oplus 000 \oplus 001 \oplus 111 \oplus 001$

R=000

dan

$S=\alpha^3A + \alpha^5B + \alpha^2C + \alpha D + \alpha^6E$
$S=\alpha^3\alpha^4 + \alpha^50 + \alpha^2\alpha^1 + \alpha\alpha^2 + \alpha^6\alpha^5$
$S=\alpha^7 + 0 + \alpha^3 + \alpha^3 + \alpha^4$
$S=001\oplus000\oplus011\oplus011\oplus110$
$S=111$
Akhirnya codeword adalah
110000010100111100011111.

Reed Solomon Decoding

Algoritma untuk decoding RS codeword adalah sebagai berikut[6]:

1. Hitung sindrom, dengan menggunakan polinomial encoder seperti pada persamaan 2.1 dan 2.2, dan lihat apakah sindrom bernilai nol atau tidak. Jika sindrom tidak bernilai nol maka error terjadi pada code-word.
2. Cari lokasi error dengan melakukan kalkulasi sindrom-sindrom.
3. Cari nilai simbol yang terkorupsi error.
4. Perbaiki codeword yang diterima berdasarkan hasil kalkulasi dan lokasi error.

Berikut ini ilustrasi teknik decoding untuk RS codeword yang sama dengan yang digunakan pada bagian encoding.

Tahap pertama adalah menghitung sindrom S_0 dan S_1 dari dataword yang diterima.

$S_0= A + B + C + D + E + R+S$ (2.5)

$S_1=\alpha^1A + \alpha^2B + \alpha^3C + \alpha^4D + \alpha^5E+\alpha^6R+\alpha^7S$ (2.6)

Nilai-nilai sindrom menyatakan ada atau tidaknya error pada codeword. Jika nilai seluruh sindrom sama dengan nol maka tidak ada error, sebaliknya jika nilainya tidak nol maka mengindikasikan ada error pada codeword.

Misalkan simbol B terkorupsi karena error ε, sehingga menjadi $B' = B + \varepsilon = B \oplus \varepsilon$, maka

$S_0= A + (B+\varepsilon)+ C + D + E + R+S$
$S_0= (A + B+ C + D + E + R+S)+ \varepsilon= \varepsilon$
$S_1= \alpha_1A + \alpha^{2(}B+\varepsilon)+ \alpha^3C + \alpha^4D + \alpha^5E+\alpha^6R+\alpha^7S$
$S_1= (\alpha_1A + \alpha^2B+ \alpha^3C + \alpha^4D + \alpha^5E+\alpha^6R+\alpha^7S) + \alpha^2\varepsilon = \alpha^2\varepsilon$

Terlihat S_0 adalah ε dan S_1 adalah $\alpha^2\varepsilon$. Dengan kata lain S_0 adalah error dan S_1 mengandung informasi posisi error. Jika *k* adalah lokasi simbol yang terkorupsi karena error, maka *k* dapat dicari dengan rumus $k = S_1/S_0 = (\alpha^2\varepsilon)/\varepsilon = \alpha^2$. Dari hasilnya *k* menunjukkan lokasi simbol yang terkena error[5]. Simbol dataword B adalah simbol yang rusak. Untuk memperbaiki error, simbol yang diterima yaitu B' dan tambahkan ε untuk mendapatkan B= B'-ε= B'+ε (ingat operasi modulo dua antara penjumlahan dan pengurangan menghasilkan nilai yang sama).

Sebagai contoh, codeword pada contoh bagian encoding terkena error pada simbol *E*, dengan $E=011=\alpha^3$, sehingga $E=111=\alpha^5$ berubah menjadi $E'=100=\alpha^2$. Hitung seluruh sindrom menggunakan persamaan 2.5 dan 2.6

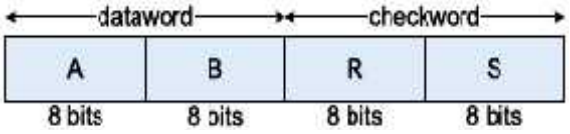
$S_0= A + B + C + D + E' + R + S$
$S_0=110\oplus000\oplus010\oplus100\oplus100\oplus000\oplus111$
$S_0=011=\alpha^3$
$S_1= \alpha^1A + \alpha^2B + \alpha^3C + \alpha^4D + \alpha^5E+\alpha^6R+\alpha^7S$
$S_0=111\oplus000\oplus110\oplus101\oplus001\oplus000\oplus111$
$S_1=010=\alpha$

Lokasi simbol yang terkorupsi error adalah $k=S_1/S_0= (\alpha^7)\alpha/\alpha^3= \alpha^5$. Hasil ini menunjukkan lokasi pada simbol kelima yaitu E. Error $\varepsilon=S_0=\alpha^3$, sehingga perbaikan error dapat dilakukan dengan operasi sebagai berikut:

$E=E'\oplus S_0=100 \oplus011=111$

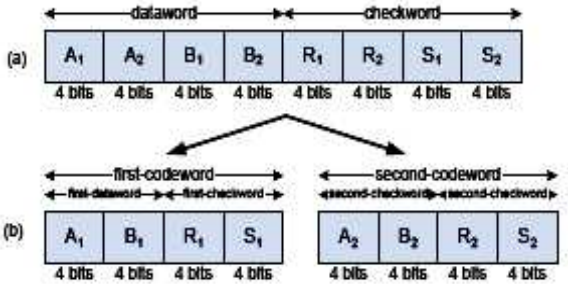
3. IMPELEMENTASI RS ENCODER DAN DECODER

Dalam penelitian ini encoder RS dibuat sedemikian hingga suatu data 16 bit akan di-encode menjadi 2 buah simbol dataword A dan B, masing-masing terdiri dari 8 bit. Pemilihan ini didasarkan pada kenyataan bahwa pada saat sekarang minimal suatu data utuh terdiri dari 16 bit. Untuk ini, GF(2^8) yang digunakan yang berarti maksimum ada 255 simbol dalam satu codeword. Namun dalam hal ini codeword yang dibentuk hanya akan berjumlah 4 simbol masing-masing simbol terdiri dari 8 bit, sehingga total codeword adalah 32 bit, seperti yang ditunjukkan oleh gambar berikut ini.



Gambar 3. Desain RS codeword untuk 16 bit dataword

Namun elemen GF(2^8) sangat banyak sekali, sehingga menghasilkan konversi data yang kompleks dari biner ke GF maupun sebaliknya. Atau jika menggunakan tabel look-up maka akan memerlukan ruang yang besar untuk tabel 256 elemen. Untuk mereduksi masalah ini satu simbol 8 bit akan dibagi menjadi 2 simbol 4 bit, kemudian posisi bit dilakukan interleave sehingga 4 simbol dari 4 bit akan membentuk suatu codeword, seperti gambar berikut.



Gambar 4. RS codeword yang dimodifikasi
(a) Sebelum interleave
(b) Setelah interleave

Dengan cara ini dapat digunakan GF(2^4) yang menghasilkan konversi dan tabel yang lebih sederhana.

Desain RS Encoder

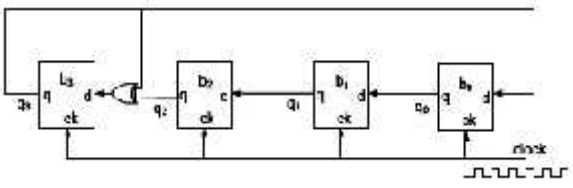
RS encoder mengikuti algoritma encoding RS seperti yang telah dipaparkan pada bagian sebelumnya namun terdapat sedikit modifikasi sesuai dengan implementasi.

1. Buatlah tabel GF(2^4)
2. Bagi dataword pertama menjadi A_1 dan B_1, setiap simbol terdiri dari 4 bit
3. Buatlah dua buah persamaan polinomial ortogonal dengan menggunakan empat simbol codword yang pertama.
   $A_1 + B_1 + R_1 + S_1 = 0$ (3.1)
   $\alpha^1 A_1 + \alpha^2 B_1 + \alpha^3 R_1 + \alpha^4 S_1 = 0$ (3.2)
4. Selesaikan persamaan untuk menemukan checkword R_1 dan S_1.
5. Gabungkan checkword R_1 dan S_1 untuk membentuk RS codeword yang pertama.
6. Ulangi langkah 2 sampai dengan 5 untuk dataword yang kedua. Dimana dataword A_2 dan B_2 akan menghasilkan checkword R_2 dan S_2.
7. Gabungkan kedua codeword dengan menggunakan metoda interleave.

Pada implementasi ini persamaan polinomial primitip dari GF(2^4) adalah $x^4+x^3+1=0$, sehingga tabel elemennya adalah sebagai berikut:

Tabel 2. Tabel elemen dari GF(2^4)

Power	Binary value	Decimal value
$\alpha^0 = 1$	0001	1
α^1	0010	2
α^2	0100	4
α^3	1000	8
α^4	1001	9
α^5	1011	11
α^6	1111	15
α^7	0111	7
α^8	1110	14
α^9	0101	5
α^{10}	1010	10
α^{11}	1101	13
α^{12}	0011	3
α^{13}	0110	6
α^{14}	1100	12



Gambar 5. Rangkaian pembangkit nilai tabel 2.

Dengan menyusun ulang 3.1 dan 3.2 akan didapatkan

$S_1 = A_1 + B_1 + R_1$

$\alpha^1 A_1 + \alpha^2 B_1 + \alpha^3 R_1 + \alpha^4 (A_1 + B_1 + R_1) = 0$

Selesaikan:

$\alpha^1 A_1 + \alpha^2 B_1 + \alpha^3 R_1 + \alpha^4 (A_1 + B_1 + R_1) = 0$

$(\alpha^3+\alpha^4)R_1=(\alpha^1+\alpha^4)A_1 +(\alpha^2+\alpha^4)B_1$

$\alpha^{15}R_1=\alpha^5 A_1+\alpha^{11}B_1$

$R_1=\alpha^5 A_1+\alpha^{11}B_1$ (3.3)

Substitusi R_1 ke dalam S_1

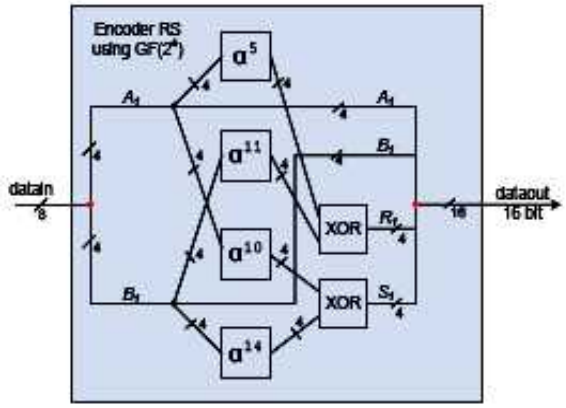
$S_1 = A_1 + B_1 + \alpha^5 A_1+\alpha^{11}B_1$

$S_1 = (1+ \alpha^5)A_1+(1+\alpha^{11})B_1$

$S_1 = (\alpha^0+ \alpha^5)A_1+(\alpha^0+\alpha^{11})B_1$

$S_1 = \alpha^{10}A_1+\alpha^{14}B_1$ (3.4)

Diagram blok dari persamaan di atas adalah ditunjukkan pada gambar 6.



Gambar 6. Diagram Blok rangkaian encoder

Dari persamaan 3.3 dan 3.4 dapat dilihat bahwa simbol A_1 dan B_1 adalah perkalian dari suatu konstanta elemen GF yaitu α^i. Perkalian ini tidak lain adalah operasi XOR antara simbol A dan B dengan elemen yang bersesuaian seperti yang terdapat dalam tabel. Tabel ini dapat dihasilkan dengan suatu rangkaian seperti gambar 3.

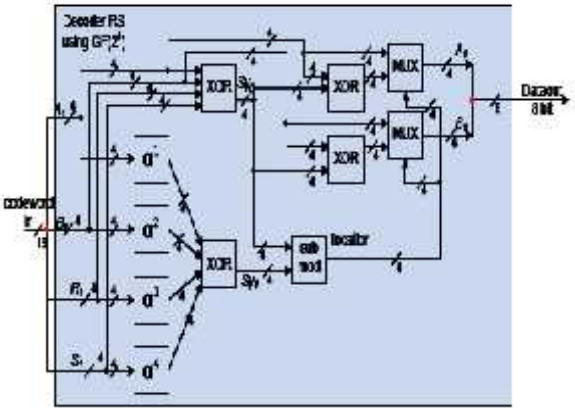
Desain RS Decoder

Decoder RS yang dibuat adalah decoder 32 bit codeword menjadi 16 bit dataword. Perlu diingat bahwa pada desain encoder codeword yang terbentuk adalah merupakan *interleaving* dari 2 buah codeword 16 bit. Sehingga algoritma decoder RS pada implementasi ini adalah:

1. Bagilah codeword pertama menjadi 4 buah simbol A_1, B_1, R_1 dan S_1. Setiap simbol terdiri dari 4 bit
2. Hitung sindrom S_{y0}= A_1 XOR B_1 XOR R_1 XOR S_1 (notasi S_{y0} danS_{y1} digunakan untuk sindrom karena S_0 dan S_1 digunakan untuk notasi simbol)
3. Hitung sindrom S_{y1}
   $S_{y1}=(\alpha^1 A_1)$ XOR $(\alpha^2 B_1)$ XOR $(\alpha^3 R_1)$ XOR $(\alpha^4 S_1)$

4. Cari lokasi error dengan menghitung S_{y0}/S_{y1}. Operasi pembagian ini tidak lain adalah pengurangan pangkat dari α.
5. Jika S_{y0}/S_{y1} =0 maka tidak ada error pada codeword. Jika tidak sama dengan nol maka ada error pada codeword. Jika S_{y0}/S_{y1} =1 maka error berada pada simbol A, jika S_{y0}/S_{y1} =2 maka error berada pada simbol B. Jika S_{y0}/S_{y1} =3 maka pada simbol R, dan jika S_{y0}/S_{y1} =4 maka pada simbol S.
6. Perbaikan error pada simbol A atau B dapat dilakukan dengan melakukan operasi XOR antara simbol yang terkorupsi error dengan S_{y0}. Sedangkan jika error berada pada R atau S tidak perlu diperbaiki karena pada output decoder R dan S akan hilang.

Diagram blok untuk decoder dapat dilihat pada gambar 7 berikut ini.



Gambar 7. Diagram blok rangkaian decoder

4. HASIL PENGUJIAN DESAIN DAN ANALISA

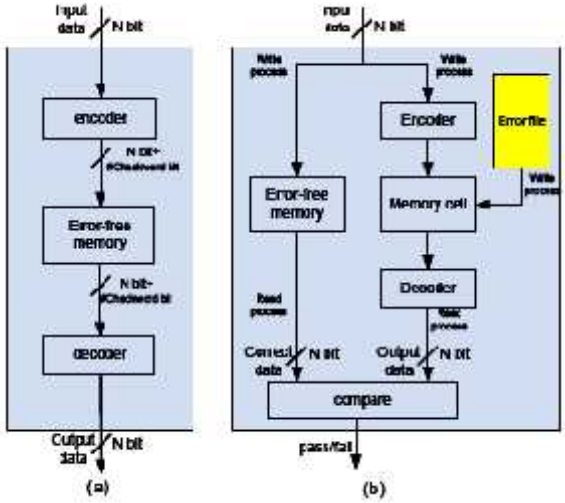
Tahapan pengujian desain dilakuakn dengan tahapan simulasi dengan menggunakan tool ModelSim dan verifikasi desain dengan menggunakan FPGA.

Simulasi

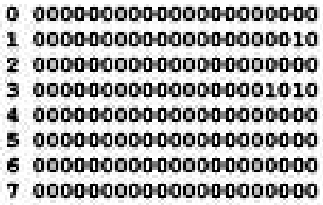
Simulasi dilakukan dengan menggunakan software simulator ModelSim. Simulasi dilakukan dengan dua tahapan yaitu simulasi bebas error dan simulasi dengan error.

Simulasi bebas error digunakan untuk memastikan bahwa desain encoder dan decoder telah berjalan sebagaimana mestinya. Sedangkan simulasi dengan error digunakan untuk mengetahui kemampuan desain encoder/ decoder Reed Solomon yang dibuat dalam menangani error.

Encoder dan decoder RS yang telah dibuat bersama-sama dengan RAM 1024x16 bit didesain dengan menggunakan VHDL. Simulasi dilakukan dengan melakukan input sejumlah data ke dalam RAM yang telah dilengkapi dengan encoder RS. Untuk kemudahan proses deteksi, maka data isi RAM adalah nilai biner dari alamatnya. Kemudian dilakukan pembacaan data pada RAM dengan hasil yang dapat dilihat pada output decoder. Simulasi error dilakukan dengan membuat sebuah file biner yang akan dioperasikan XOR dengan isi RAM, dimana nilai 1 pada file biner menyatakan bahwa pada sel tersebut RAM terkena error.



Gambar 8. (a) Diagram blok simulasi tanpa error (b) Diagram blok simulasi dengan error



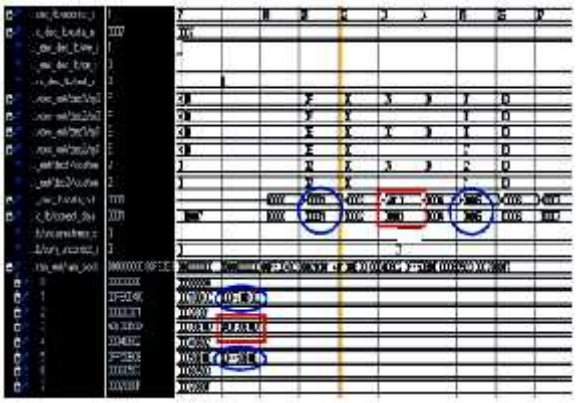
Gambar 9. File eror yang akan dioperasikan XOR (mask) dengan isi RAM.

Simulasi tanpa error menunjukkan bahwa encoder dan decoder telah bekerja sesuai dengan desain. Simulasi dengan error juga menunjukkan hasil bahwa RS codeword yang dibuat mampu menangani error sampai 8 bit dalam satu simbol.

Diagram *wave* pada yang ditunjukkan oleh hasil simulator ModeSim pada gambar 9 memperlihatkan pada area yang ditandai dengan kotak dan lingkaran adalah merupakan data yang diganggu oleh error, yaitu pada RAM alamat 1, 3 dan 5. Pada RAM dengan lamat 1 dimasking dengan error $00FF0000_h$ sehingga menjadi $00010D0C_h$, alamat 3 dengan 40100000_h sehingga menjadi $40130E0D_h$ sedangkan alamat 5 di-*mask* dengan $0FF00000_h$ sehingga menjadi $00050B0E_h$.

Pada siklus pembacaan data dapat dilihat bahwa untuk alamat 1 dan 5 error dapat diperbaiki karena nilai data sesuai dengan nomor

alamatnya. Sedangkan pada alamat 3, hasilnya berbeda yang menunjukkan error tidak dapat diperbaiki. Hal ini benar karena error pada alamat 5 lebih dari satu simbol.



Gambar 10. Tampilan wave diagram simulator ModelSim untuk proses write dan read RAM

Verifikasi

Verifikasi dilakukan dengan melakukan sintesis rangkaian VHDL dengan memetakan hasil desain VHDL ke papan FPGA Virtex 4. Tahapan ini menghasilkan sintesis pada tabel 3. sebagai berikut:

Tabel 3. hasil sintesis encoder-decoder RS pada Xilinx Virtex 4.

Unit	Encoder	decoder
4 input LUT	22	86
Delay(ns)	5,54	10,37

Analisa

Besarnya area sel memori tergantung dari lebar bit codewordnya. Sehingga pada bagian sintesis tidak diperlukan sintesis area sel memori. Untuk satu dataword 16 bit pada desain ini akan memberikan penambahan 16 bit checkword untuk membentuk 32 bit codeword. Ini berarti *overhead* untuk desain RS yang digunakan pada percobaan ini adalah sebesar 100 persen. Ini masih lebih bagus dibandingkan dengan error correction code dengan menggunakan Triple Modular Redundant (TMR) yang menduplikasi hardware rangkaian menjadi 3 buah. Dimana overhead areanya adalah 200 persen. Jika dibandingkan dengan error correction code dengan menggunakan Hamming, memang overhead area Hamming lebih kecil, namun kemampuan memperbaiki errornya cuma satu bit.

Kecepatan transfer data akan mengalami penambahan sebesar delay yang ditunjukkan oleh tabel 3. Namun jika sistem komputer yang dibangun menggunakan pipeline, penambahan delay tulis baca ini tidak menjadi masalah.

5. PENUTUP

Kesimpulan

Dari penelitian ini dapat disimpulkan bahwa:

1. Soft error adalah error yang terjadi pada suatu desain elektronik, misalnya memori, yang disebabkan oleh pancaran sinar kosmik yang ada di alam. Sifatnya tidak permanen dan akan hilang jika sistem direset.
2. Fenomena soft error akan lebih sering terjadi pada memori dengan ukuran nanometer. Sehingga diperlukan suatu mekanisme untuk mengurangi dampaknya.
3. Error correction code dengan Reed Solomon code dapat digunakan sebagai fault-tolerant untuk soft error yang terjadi pada memori.
4. Pengunaan metode interleave sel memori dapat menyederhanakan penggunaan finite field GF(2^m) yang digunakan dalam pembentukan RS codeword.

Saran

Dari penelitian yang telah dilakukan, penulis dapat memberikan saran yang terkait dengan penelitian ini antara lain:

- Perlu dicoba desain RS untuk lebar data yang lain, karena kemungkinan dimasa depan, ukuran terkecil sebuah data tidak lagi 16 bit tapi mungkin saja 64 bit atau lebih. Yang perlu diperhatikan adalah penggunaan finite field GF, sebaiknya jangan melebihi dari GF(2^8) karena akan menyebabkan rancangan menjadi lebih kompleks.
- Desain encoder/decoder RS ini dapat ditambahkan suatu mekanisme yang akan menulis balik data pada RAM jika terdapat error. Sehingga error tidak akan terakumulasi lebih dari satu simbol yang dapat menyebabkan error tidak dapat diperbaiki.

6. DAFTAR PUSTAKA

1. Anonim, t.t, *Internasional Technology Raodmap of Semiconductor 2008*
2. A. Houghton, 1997, *The Engineer's Error Coding Handbook*. Chapman and Hall.
3. G.E. Moore, 2006, *Cramming more components onto integrated circuits, Electronics Magazine.* p. 4. Retrieved 2006-11-11.
4. R. Mastipuram and E. Wee, 2004, *Soft Errors' Impact on System Reliability*, *EDN*, vol. 30, pp. 69–74, 2004.
5. S. Wicker and V. K. Bargava, 2009, *Reed Solomon Codes and Their Application*. Wiley, 2009.
6. R. H. Morelos, 2002, *The Art of Error Correcting Coding*. John Wiley and Sons Inc.