LAPORAN LATIHAN ILMIAH TAHAP AKHIR

WXES 3181/ WXES 3182

SISTEM PENGESANAN DAN PEMBETULAN RALAT MENGGUNAKAN TEKNIK HAMMING CODES DAN CYCLIC REDUNDENCY CHECK (CRC) (DECODER)

OLEH:

WAN RAMLAH BINTI WAN IBRAHIM
WEK000452

Laporan ini merupakan sebahagian daripada keperluan Ijazah Sarjana Muda

Sains Komputer dan Teknologi Maklumat Universiti Malaya

SESI 2002/2003

Abstrak

Pengesanan dan pembetulan ralat banyak digunakan dalam sistem komunikasi di mana ia menyediakan penyelesaian yang inovatif kepada komunikasi yang bermasalah. Tesis ini dicadangkan untuk mengkaji asas pengesanan dan pembetulan ralat.Tajuk tesis ini ialah Sistem Pengesanan dan Pembetulan ralat menggunakan teknik Hamming Codes dan Cyclic Redundency Checks (Decoder). Teknik pengesanan dan pembetulan ralat yang digunakan dalam kajian ini adalah Hamming Codes dan Cyclic Redundency Check (CRC). Hamming Codes merupakan kod yang boleh mengesan dan membetulkan ralat di mana dua jenis ralat dapat dikesan dan satu jenis ralat dapat dibetulkan. Cyclic Redundency Check (CRC) merupakan kod pengesanan ralat yang menggunakan teknik pengiraan matematik ke atas blok data dan akan mengembalikan kandungan data tersebut. Cyclic Redundency Check (CRC) dapat mengesan lebih daripada dua jenis ralat. Laporan ini akan akan menjelaskan tentang pengenalan kepada projek, kajian yang dilakukan, metodologi yang digunakan dalam membangunkan sistem ini, rekabentuk sistem yang dicadangkan sehinggalah keperingkat perlaksanaan dan pengujian sistem. Dalam membangunkan sistem digital ini, bahasa pengaturcaraan VHDL digunakan untuk mensimulasi dan memaparkan operasi pengesanan dan pembetulan ralat bagi kedua-dua jenis teknik ini.

Penghargaan

Syukur kehadrat ilahi kerana dengan limpah kurnianya, telah mengizinkan saya menyiapkan tugasan laporan Latihan Ilmiah WXES 3181/ WXES 3182 ini dengan sempurna.

Sekalung budi setinggi hati, hanya itu dapat diperi dari saya. Tidak ada kata secantik bahasa, tidak ada madah seindah gurindam untuk diungkapkan kepada pihak-pihak yang terlibat dalam menyiapkan laporan ilmiah ini. Kajian ini tidak akan sempurna jika tiada kerjasama daripada pihak-pihak tertentu .

Sebagai tanda penghargaan, di sini saya ucapkan jutaan terima kasih kepada penyelia saya iaitu En. Mohd Yamani Idna bin Idris yang banyak membantu saya dalam memberikan idea, maklumat, pandangan–pandangan tertentu dan juga perangsang untuk saya mencari idea bagi menyudahkan laporan ini. Tidak lupa juga kepada moderator iaitu En. Zaidi Razak kerana memberikan komen-komen yang bernas semasa sesi VIVA.

Penghargaan tidak terhingga juga diucapkan kepada rakan sekumpulan saya iaitu Cik Fauziah Mahmod yang banyakmenumbangkan idea serta sama-sama bertungkus lumus mendapatkan sumber rujukan dan memberikan pandangan dan penerangan kepada saya dikala saya menghadapi masalah .

Sekalung penghargaan juga dirakamkan untuk semua individu yang terlibat secara langsung atau tidak dalam membantu saya mendapatkan sumber maklumat sebagai bahan rujukan menyiapkan tesis ini .

Segala jasa kalian akan saya kenang sepanjang hayat dan saya sudahi dengan serangkap pantun :

Anak-anak bemain palas ,
Salah seorang terkena duri ,
Jasa kalian tidak dapat ku balas,
Hanya ucapan terima kasih sebagai pengganti diri.

Sekian terima kasih ,Wassalam .

Wan Ramlah Wan Ibrahim

WEK000452

Jabatan Sistem dan Rangkaian Komputer

Isi Kandungan

Kajian Literasi

Senarai Jadual

Senarai Rajah

BAB 1
PENGENALAN

Bab 1: Pengenalan

1.1 Pengenalan

Tujuan asas perlaksanaan pembangunan sistem digital ini adalah untuk mengetahui dengan lebih lanjut tentang cara-cara pengesanan ralat dan pembetulan ralat dilakukan dalam sistem penghantaran data. Oleh itu dalam bahagian ini, beberapa kajian dilakukan melibatkan definisi masalah, skop pemasalahan yang perlu diselesaikan, objektif projek dan kekangan yang dijangka berlaku semasa perlaksanaan sistem ini.

Rangkaian mesti berupaya menghantar data dari satu peranti ke peranti yang lain dengan tepat. Dalam keadaan tertentu data yang dihantar mengalami perubahan sama ada satu atau lebih bit yang dihantar akan hilang atau berubah disebabkan oleh gangguan seperti hingar maka satu mekanisma yang boleh dipercayai yang berupaya membetulkan mengesan dan membetulkan ralat diperlukan. Lewahan data diperkenalkan ke dalam data supaya ralat dapat dikesan dan seterusnya dibetulkan. Masalah yang timbul ialah terdapatnya ralat dalam data yang dihantar di mana ralat ini akan menyebabkan data yang dihantar tidak tepat seterusnya sistem akan beroperasi dengan tidak betul.

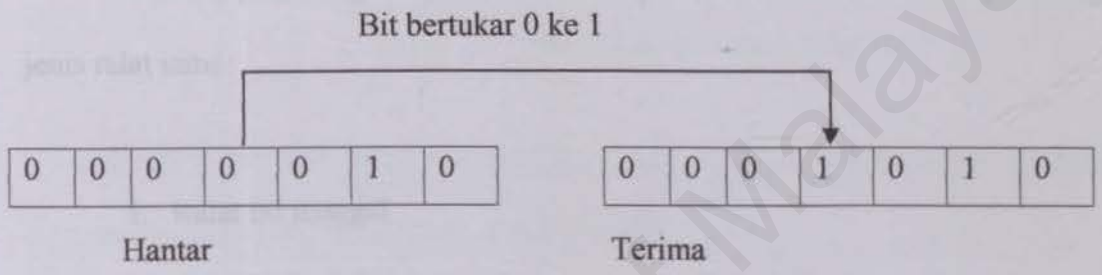
Dalam *Hamming Codes* terdapat 2 jenis ralat yang mungkin wujud semasa penghantaran data iaitu:

i. Ralat bit tunggal

ii. Ralat bit berbilang

1.1.1 Ralat bit tunggal

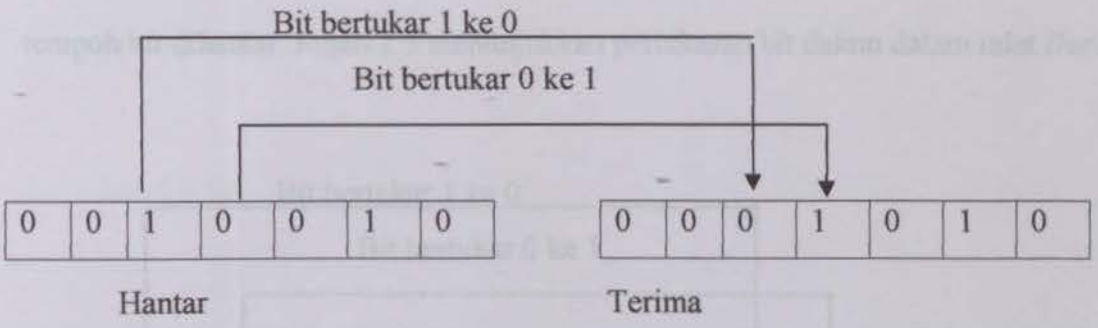
Ralat bit tunggal biasanya berlaku bila hanya satu bit dalam satu unit data dari stesen penghantar ke stesen penerima telah berubah dari 1 ke 0 atau dari 0 ke 1. Ralat bit tunggal mungkin berlaku sekiranya data yang di hantar menggunakan penghantaran selari . Rajah 1.1 menunjukkan pertukaran bit dalam ralat bit tunggal.



Rajah 1.1: Pertukaran bit dalam ralat bit tunggal

1.1.2 Ralat bit berbilang

Ralat bit berbilang merujuk kepada pertukaran bit yang berlaku dalam blok data di mana bit–bit yang berubah samada dari 1 ke 0 atau 0 ke 1 adalah sebanyak 2 kedudukan seperti Rajah 1.2.



Rajah 1.2.: Pertukaran bit dalam ralat bit berbilang

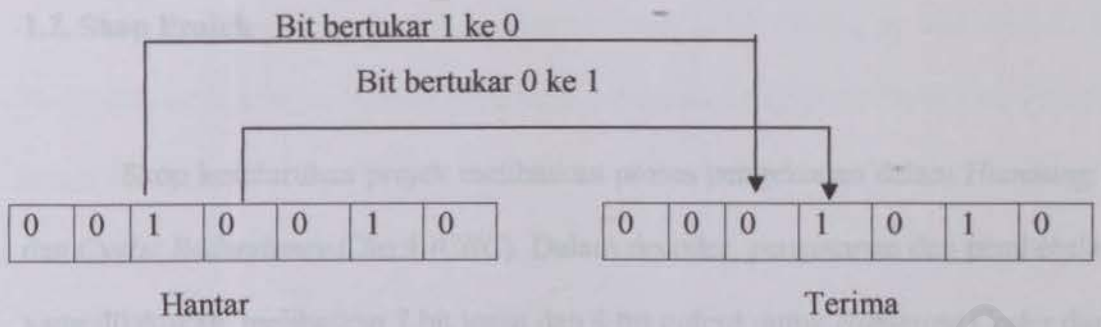
Berbeza pula dengan *Cyclic Redundency Check* kerana kod ini dapat mengesan 4 jenis ralat iaitu :

i. Ralat bit tunggal
ii. Ralat bit berbilang
iii. Ralat nombor genap
iv. Ralat *Burst*

1.1.3 Ralat Burst

Ralat bit tunggal dan ralat berbilang belaku sama seperti yang diterangkan di bahagian atas. Ralat *Burst* adalah ralat di mana 2 atau lebih bit dalam satu unit data berubah dari 1 ke 0 atau dari 0 ke 1. Ralat ini tidak semestinya berlaku pada bit yang berturutan. Panjang *Burst* diukur dari ralat yang pertama hingga ralat yang terakhir. Walaupun tidak terdapat bit yang berubah di tengah-tengah. Ralat ini biasanya berlaku

dalam penghantaran bersiri di mana tempoh berlakunya hingar adalah lebih lama dari tempoh bit dihantar. Rajah 1.3 menunjukkan pertukaran bit dalam dalam ralat *Burst*.



Rajah 1.3: Pertukaran bit dalam ralat *Burst*

Tidak semua masalah dapat diselesaikan. Dalam menyiapkan projek ini kemungkinan terdapat masalah yang tidak dapat di atasi seperti :

i. Dalam *Hamming Codes*, jika ralat yang dikesan mempunyai lebih daripada satu bit yang ralat, kemungkinan *Hamming Codes* tidak boleh membetulkannya

ii. *Hamming Codes* adalah seperti bit pariti di mana ianya hanya boleh digunakan untuk pesanan yang pendek sahaja. Maka, jika pesanan atau jujukan bit yang dihantar adalah terlampau panjang maka pengesanan dan pembetulan ralat mungkin tidak dapat dilaksanakan

iii. Dalam *Cyclic Redundency Check*, walaupun ia dapat menyelesaikan masalah *Hamming Codes* iaitu boleh mengesan lebih daripada 2 ralat namun terdapat juga masalah lain yang di hadapi seperti ia perlu mengulang penghantaran jika terdapat

ralat dikesan. Ini akan menyebabkan sistem terpaksa mengambil masa yang lama untuk mendapatkan data yang betul.

1.2. Skop Projek

Skop keseluruhan projek melibatkan proses pengekodan dalam *Hamming Codes* dan *Cyclic Redundency Check (CRC)*. Dalam decoder, pengesanan dan pembetulan ralat yang dilakukan melibatkan 7 bit input dan 4 bit output untuk *Hamming Codes* dan 7 bit output untuk CRC.

Di sini dapat dibahagikan skop dekoder ini kepada dua segmen dan berikut di terangkan tentang segmen-segmen tersebut iaitu :

i. Segmen 1 : Mengesan ralat dalam sesuatu cip

Pengesanan ini merangkumi kajian dari segi kebolehan cip mengesan bit mana yang mengalami perubahan dalam data yang dihantar kepada penerima melalui saluran penghantaran .

ii. Segmen 2: Pembetulan ralat dalam sesuatu cip

Pembetulan ralat diperlukan untuk melindungi maklumat yang dihantar melalui saluran penghantaran di mana tindakan pembetulan ralat dilakukan selepas pengesanan ralat dilakukan.

1.2.1 Skop pengesanan dan pembetulan ralat dalam *Hamming Codes*

Dalam *Hamming Codes* pengesanan ralat boleh dibuat ke atas semua ralat bit tunggal dan bit berbilang dan pembetulan ralat hanya boleh dilakukan ke atas ralat bit tunggal sahaja. Oleh sebab itu *Hamming Codes* dikatakan dapat mengesan 2 ralat dan membetulkan 1 ralat. Dalam skop penyahkodan *Hamming Codes* 7 bit input, 4 bit output dengan jarak *Hamming* bersamaan dengan 3 digunakan .

1.2.2 Skop pengesanan dan pembetulan ralat dalam *Cyclic Redundency Check*

Dalam *Cyclic Redundency Check* pula pengesanan ralat boleh dilakukan ke atas lebih daripada 2 ralat. Jenis-jenis ralat yang boleh dikesan adalah ralat bit tunggal, ralat berbilang, ralat nombor genap dan ralat *Burst*. Melalui kaedah *Cyclic Redundency Check codes*, pembetulan ralat tidak sama seperti kaedah *Hamming Codes* yang menggunakan kod pembetulan ralat.

1.2.3 Skop penulisan projek

Penulisan kandungan projek dibahagiakan kepada beberapa bab seperti di bawah:

Bab 1:

Membincangkan tentang pengenalan kepada tesis di mana dalam bab ini akan memberi gambaran tentang tujuan kajian dijalankan, objektif, masalah yang perlu

diselesaikan dalam mengesan dan membetulkan ralat, skop permasalahan yang akan dikaji, kekangan yang wujud dalam melaksanakan projek dan penjadualan kerja.

Bab 2:

Tumpuan akan diberikan kepada kajian literasi yang menghuraikan dengan lebih lanjut tentang sub-sub topik dalam bab 1. Dalam bab ini juga akan dikaji tentang perbezaan antara teknik Hamming dan CRC serta aplikasi kedua-duanya.

Bab 3:

Menyentuh tentang pendekatan pembangunan sistem dan metodologi yang digunakan dalam pembangunan sistem. Selain itu, turut dibincangkan dalam bab ini adalah tentang pengenalan terhadap perkakasan dan perisian yang digunakan. Turut dibincangkan ialah perisian yang digunakan iaitu perisian WebPACK yang menggunakan bahasa pengaturcaraan VHDL serta kelebihan dan keburukan VHDL.

Bab 4:

Menerangkan tentang rekabentuk sistem yang dicadangkan di mana ia meliputi modul-modul yang hendak dibina serta fungsinya, kesinambungannya dan sebab-sebab pembangunannya. Turut dimasukkan ialah gambarajah blok, gambarajah logik dan *top level* bagi *Hamming Codes* dan *Cyclic Redundency Check codes* untuk menjelaskan lagi rekabentuk sistem digital ini.

1.3. Objektif Projek

Objektif utama mengkaji dan membangunkan projek ini adalah untuk :

i. Mengesan dan membetulkan ralat semasa penghantaran data dalam sistem komputer. Ia memerlukan pelaksanaan kod pembetulan ralat (Error-Correcting Codes) yang dikenali *Hamming Codes* dan *Cyclic Redundency Check codes (CRC)*.

ii. Selain itu projek ini juga bertujuan untuk melihat bagaimana decoder berperanan dalam mengesan dan membetulkan ralat dengan menerima 7 bit input dan mengeluarkan 4 bit output untuk Hamming dan 7 bit output untuk CRC setelah proses penyahkodan dilakukan.

iii. Menghasilkan satu rekabentuk cip untuk mengesan dan membetulkan ralat dengan menggunakan bahasa pengaturcaraan VHDL bagi melaksanakan simulasi untuk melihat senibina dan kelakuan cip.

iv. Memahirkan diri dalam penggunaaan bahasa pengaturcaraan VHDL bagi menghasilkan suatu cip.

1.4. Kekangan

Dalam melaksanakan atau membangunkan kajian ini, had-had yang di jangka tidak dapat di atasi adalah seperti :

i. Simulasi bagi VHDL tidak mengeluarkan hasil seperti yang dikendaki kerana kemungkinan berlakunya kesilapan pemahaman terhadap proses yang dilakukan oleh Hamming dan CRC seterusnya menyebabkan kesilapan terhadap *source codes.*

ii. Tidak dapat menggambarkan teori matematik dalam pelaksanaan rekabentuk yang dicadangkan kerana ia bersifat teoritikal .

iii. Satu kesilapan yang dilakukan boleh menyebabkan bahagian lain juga terjejas.

iv. Perisian yang akan digunakan mungkin tamat tempoh lisennya dan mungkin perisian lain yang akan digunakan.

1.5. Penjadualan Projek

Penjadualan sangat diperlukan dalam memastikan projek dapat disiapkan dalam masa yang telah ditetapkan. Oleh itu satu jadual ringkas yang menyatakan tempoh masa aktiviti-aktiviti sepanjang proses pembanguhan diperlukan agar kerja dapat dijalankan dengan sistematik dan efisien.

Pada dasarnya tempoh menyiapkan projek adalah selama 2 semester di mana projek ini dibahagikan kepada 2 fasa iaitu Latihan Ilmiah I dan Latihan Ilmiah II. Jadual di bawah menujukkan aktiviti-aktiviti yang dijalankan.

Jadual 1.1 Jadual Perangcangan LI I (Semester 1)

Minggu	Aktiviti
1	Memilih tajuk tesis
2	Kenalpasti objektif, skop dan pengenalan projek dan pencarian bahan rujukan .
3	Membuat rujukan melalui buku, majalah, tesis dan internet.
4	Membuat rujukan melalui buku, majalah, tesis dan internet
5	Membuat rujukan melalui buku, majalah, tesis dan internet dan analisa data
6	Analisa data dan penyediaan proposal
7	Analisa data dan penyediaan proposal projek
8	Penyedian proposal projek
9	Penyedian proposal projek
10	Penyedian proposal projek dan VIVA
11	VIVA
12	Penyedian proposal projek
13	Hantar laporan WXES 3181

Jadual 1.2 Jadual perangcangan LI II (Semester 2)

Minggu	Aktiviti
1	Rekabentuk pengkodan
2	Rekabentuk pengkodan
3	Rekabentuk pengkodan
4	Pengkodan
5	Pengkodan
6	Pengkodan
7	Pengkodan
8	Pengkodan dan pengujian
9	Pengkodan dan pengujian
10	Pengujian dan penyelenggaraan
11	Penyelenggaraan
12	Penyelenggaraan
13	Dokumentasi projek dan penyediaan manual pengguna
14	Dokumentasi projek dan penyediaan manual pengguna

Gambarajah di bawah menunjukkan garis masa yang menggambarkan aktiviti-aktiviti atau proses kerja secara umum bermula dari awal iaitu Latihan Ilmiah I (WXES 3181) sehinggalah tamat dan diikuti Latihan Ilmiah II (WXES 3182).

Aktiviti/Minggu	3	4	1	2	3	4	1	2	3	4	1	2
Pemilihan tajuk, pemahaman dan persediaan projek												
Kajian Ilmiah,carian bahan dan perancangan projek												
Analisa data												
Penyediaan Proposal												

Rajah 1.4: Garis masa bagi perancangan LI I (WXES 3181)

Aktiviti/Bulan	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Rekabentuk pengkodan														
Pengkodan														
Pengkodan dan Pengujian														
Penyelengaraan														
Penyediaan dokumentasi projek dan manual pengguna														

Rajah 1.5: Garis masa bagi perancangan LI II (WXES 3182)

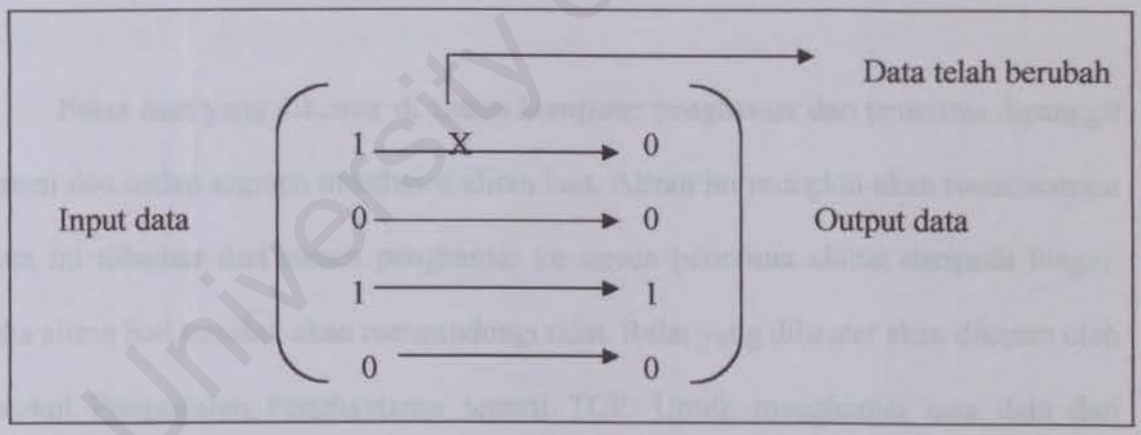
BAB 2
KAJIAN LITERASI

Bab 2: Kajian Literasi

2.1. Pengenalan

Dalam projek Latihan Ilmiah I ini, ditampilkan bagaimana ralat dalam sesuatu perlaksanaan sistem dapat dikesan seterusnya bagaimana cara untuk membetulkan ralat tersebut. Hasil yang dipaparkan adalah dalam bentuk jujukan binari '1' dan '0'. Pengesanan dan pembetulan ralat ini dilaksanakan menggunakan perisian VHDL iaitu satu perisian khas yang digunakan untuk merekabentuk suatu perkakasan.

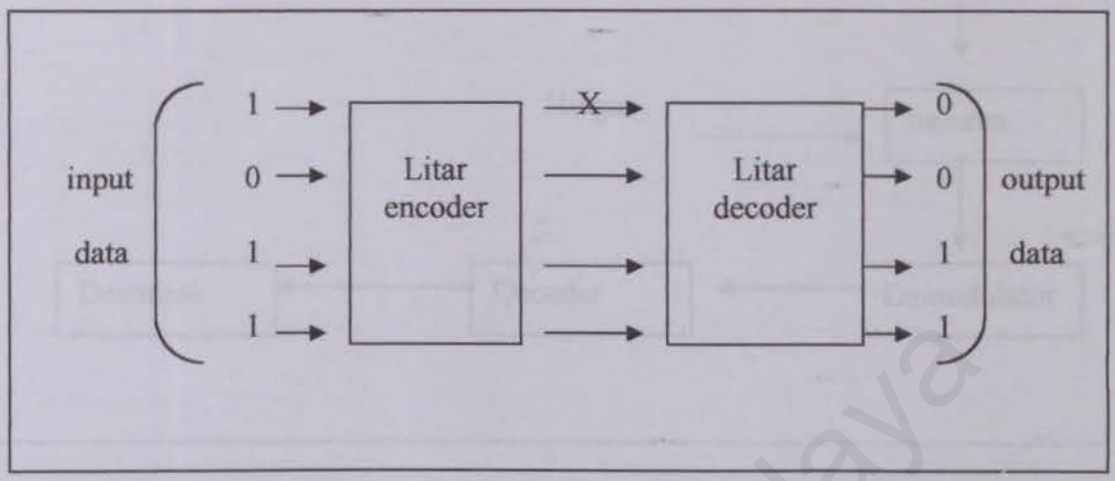
Pengesanan dan pembetulan ralat adalah teknik yang penting untuk memastikan realibiliti data. Kebanyakan ralat menyebabkan kemusnahan dan kerosakan data semasa menghantarnya melalui saluran komunikasi.



Rajah 2.1: Ralat dalam penghantaran data

Dalam Rajah 2.1, hingar telah menyebabkan 1 atau lebih bit bertukar nilainya, ralat yang berlaku mesti dikesan dan dibetulkan. Dalam Rajah 2.2, data asal dikodkan

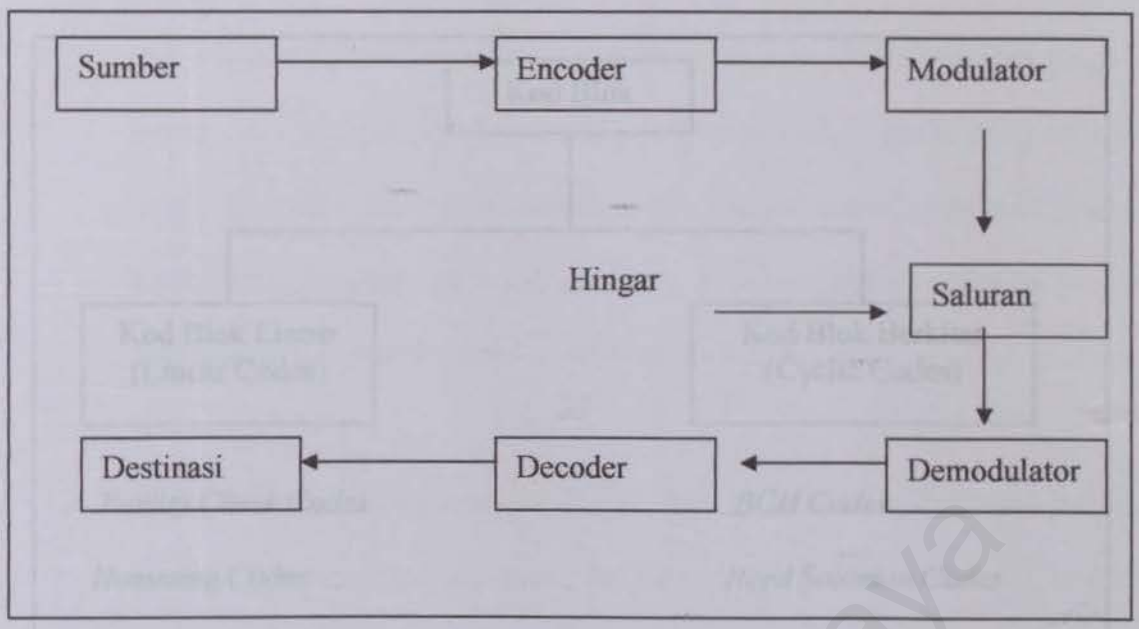
oleh encoder sebelum penghantaran ke komputer penerima dan ianya akan diperiksa oleh litar pembetulan ralat.



Rajah 2.2: Pembetulan ralat menggunakan encoder dan decoder

2.2. Penghantaran data

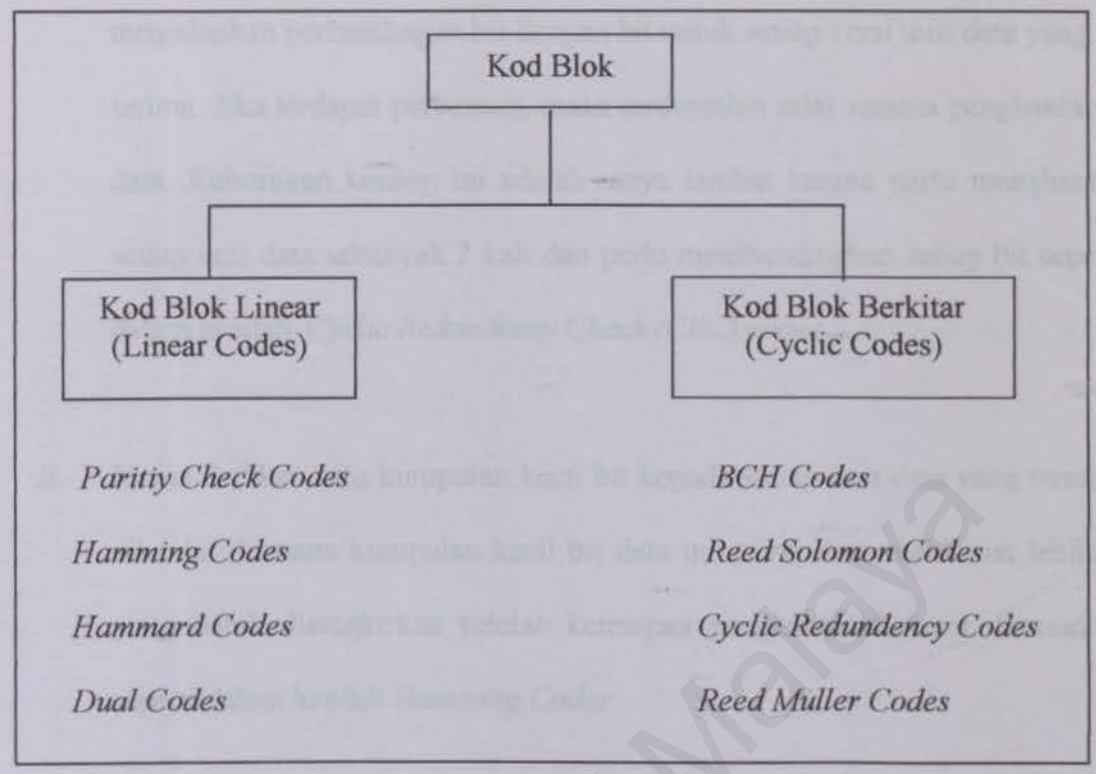
Peket data yang dihantar di antara komputer penghantar dan penerima dipanggil segmen dan setiap segmen membawa aliran bait. Aliran ini mungkin akan rosak semasa aliran ini dihantar dari stesen penghantar ke stesen penerima akibat daripada hingar. Maka aliran bait tersebut akan mengandungi ralat. Ralat yang dihantar akan dikesan oleh Protokol Pengawalan Penghantaran seperti TCP. Untuk menghantar satu data dari penghantar ke penerima, beberapa proses perlu dilakukan terlebih dahulu ke atas data tersebut. Salah satunya adalah proses Rajah 2.3 menunjukkan proses yang berlaku dalam *channel coding* dalam sistem penghantaran:



Rajah 2.3: Gambarajah Blok bagi sistem penghantaran

Channel coding berfungsi untuk melindungi data dari ralat yang mungkin berlaku semasa proses penghantaran dengan cara menambahkan bit lewahan (*Redundency bit)* ke dalam data yang hendak dihantar. *Channel coding* yang digunakan untuk mengesan ralat juga mampu membetulkan ralat. Tujuan utama teknik pengesanan dan pembetulan ralat ialah untuk membaiki prestasi sistem penghantaran digital.

Terdapat 2 jenis *Channel coding* iaitu kod blok dan kod *convolutiona*l. Teknik pengesanan dan pembetulan ralat yang akan di bincangkan adalah tergolong dalam ralat kod blok iaitu kod blok linear (linear code) dan kod blok berkitar (cyclic code). Jenis kod linear yang di pilih adalah adalah *Hamming Codes* dan bagi kod blok berkitar pula jenis kod blok yang dipilih adalah *Cyclic Redundency Check (CRC).* Rajah 2.4 menunjukkan pembahagian jenis kod blok bersama contohnya.



Rajah 2.4: Contoh Kod Blok

Kod Blok linear yang akan dibincangkan adalah *Hamming Codes* dan *Cyclic Redundency Check (CRC).*

2.3. Pengesanan dan Pembetulan Ralat

Pengesanan dan pembetulan ralat merupakan teknik yang penting untuk memastikan data yang digunakan untuk berkomunikasi adalah asli. Di sini akan dibincangkan bagaimana ralat dalam sistem dikesan dan dibetulkan. Secara amnya terdapat 2 konsep umum dalam pengesanan ralat iaitu:

i. Menghantar unit data sebanyak 2 kali di mana peranti penerima akan menjalankan perbandingan bit dengan bit untuk setiap versi unit data yang di terima. Jika terdapat perbezaan, maka terdapatlah ralat semasa penghantaran data. Keburukan konsep ini adalah ianya lambat kerana perlu menghantar setiap unit data sebanyak 2 kali dan perlu membandingkan setiap bit seperti dalam kaedah *Cyclic Redundency Check (CRC) codes*.

ii. Menambahkan satu kumpulan kecil bit kepada setiap unit data yang hendak dihantar di mana kumpulan kecil bit data ini merupakan maklumat lebihan yang boleh disingkirkan setelah ketetapan penghantaran dapat ditentukan seperti dalam kaedah *Hamming Codes*.

Dalam pembetulan ralat pula, terdapat 2 cara membetulkannya iaitu:

i. Apabila ralat ditemui, penerima boleh meminta penghantar menghantar semula keseluruhan unit data yang dihantar sebelumnya.

ii. Apabila ralat ditemui, penerima menggunakan kod pembetulan-ralat (error correcting code) untuk membetulkan ralat tertentu yang wujud. Bagi ralat bit tunggal, penerima hanya perlu mengubah nilai 1 ke 0 atau 0 ke 1 apabila ralat dijumpai dalam data yang dihantar. Oleh itu penerima hanya perlu tentukan lokasi bit ralat berada. Oleh itu bit lewahan diperlukan untuk melakukan

pembetulan ralat. Bit lewahan akan ditambah kepada unit data yang dihantar seperti yang digunakan dalam kaedah Hamming.

2.4. Kaedah Pembetulan Ralat

Dalam kebanyakan sistem komunikasi data, kaedah pembetulan ralat yang digunakan sekarang ini ialah:

i. Maklum balas maklumat
ii. *Forward Error Control* (FEC)
iii. *Automatic Repeat Request* (ARQ)

Kaedah maklum balas maklumat digunakan dalam penghantaran tak segerak, sementara kaedah FEC dan ARQ digunakan dalam penghantaran segerak.

2.4.1. Kaedah FEC

Menurut kaedah kawalan ralat ke depan (FEC), ralat dibetulkan di stesen penerima tanpa menggunakan tatacara hantaran semula. Oleh itu, blok yang mengandungi bit data yang dihantar oleh stesen penerima perlu mengandungi bit lewahan (*Redundency bit*) yang digunakan untuk mengesan ralat, menentukan kedudukan bit ralat di dalam blok dan membetulkan ralat tersebut. Teknik kod pembetulan ralat bit tunggal Hamming merupakan satu kaedah FEC yang luas digunakan pada masa kini.

2.4.2 Kaedah ARQ

Dalam kaedah mengulang automatik (ARQ), fungsi kandungan blok perlu dikira oleh stesen penghantar dan hasil daripada pengiraan ini dihantar bersama-sama dengan blok tersebut. Stesen penerima akan melaksanakan pengiraan yang sama dan membandingkan hasil pengiraannya dengan hasil pengiraan yang diterima daripada stesen penghantar. Jika hasil pengiraan yang dibuat oleh stesen penerima, sama dengan hasil pengiraan yang dibuat oleh stesen penghantar maka stesen penerima akan menghantar aksara perakuan (ACK) ke stesen penghantar. Sebaliknya jika jika hasil pengiraan itu tidak sama, stesen penerima stesen penerima akan menghantar perakuan negatif (NAK) ke stesen penghantar dan ini mengakibatkan stesen penghantar akan menghantar semula blok data yang mengdungi ralat. Kaedah ini biasa digunakan dalam teknik polinomial berkitar yang merupakan asas kepada kaedah pengesanan dan pembetulan ralat *Cyclic Redundency Check (CRC).*

2.5. Pengenalan kepada *Hamming Codes*

Hamming Codes merupakan satu kod nontrivial yang boleh melakukan pengesanan dan pembetulan kod binari yang berlaku semasa pemindahan data. Kod ini telah dicipta pada tahun 1950, oleh Richard Hamming. Teknik Hamming dapat mengesan satu bit ralat tunggal, mengesan dua bit ralat tunggal dan membetulkan satu bit ralat tunggal. Untuk sebarang integer positif $m \geq 3$, maka wujud *Hamming Codes* dengan parameter seperti dibawah:

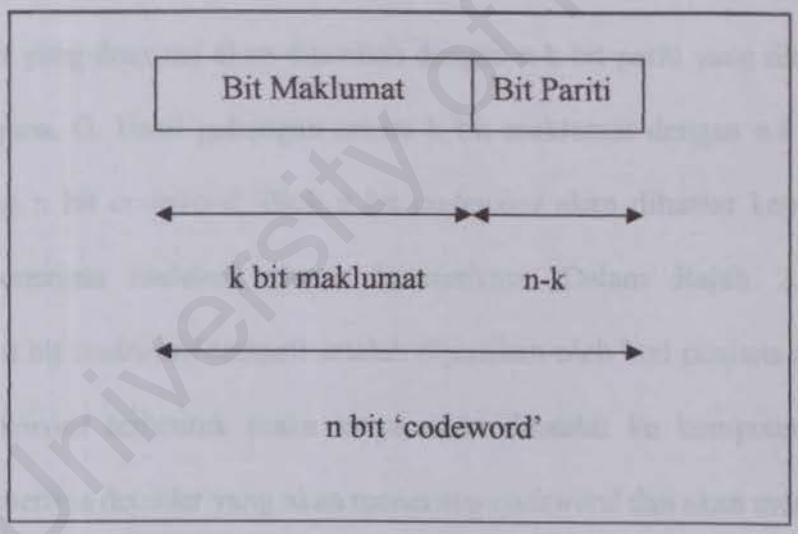
Panjang kod : $n = 2^m - 1$

Jumlah bit maklumat : $k = 2^m - m - 1$

Jumlah bit pariti : $n - k = m$

Jumlah ralat yang boleh di betulkan : $t = 1$ (jarak hamming, $d = 3$)

Kod yang dihantar adalah dalam bentuk blok k yang dinamakan bit maklumat dari n bit yang dinamakan *codeword* di mana jarak Hamming adalah d iaitu ia rujuk sebagai (n , k , d) atau biasanya digambarkan sebagai kod (n , k). Bentuk kod blok bagi (n ,k) bit data adalah seperti rajah 2.5.



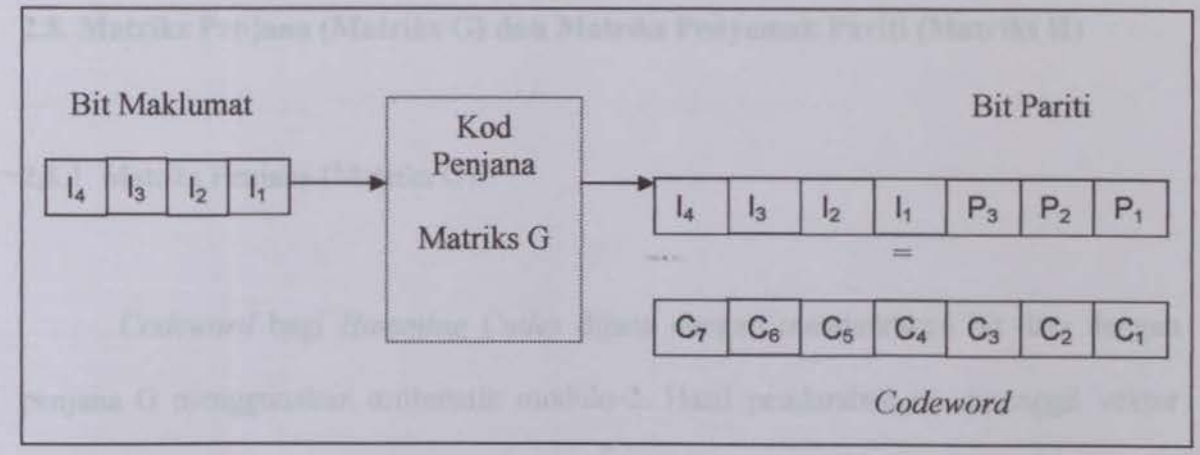
Rajah 2.5: Kod blok (k ,n)

2.6. Jarak Hamming

Di sini *Hamming Codes* (7,4) akan dibincangkan iaitu n=7 dan k=4. Jarak Hamming bagi (7,4) adalah 3. Jarak Hamming adalah jumlah bit yang berbeza antara 2 kod. Untuk menentukan jarak Hamming bagi sesuatu *codeword,* maka operasi OR digunakan. Katakan 2 kod adalah 10010101 dan 10111001 maka perbezaan nombor binari 2 kod tersebut adalah 00101100 maka ini menunjukkan terdapat 3 perbezaan. Selaras dengan itu jarak Hamming adalah 3.

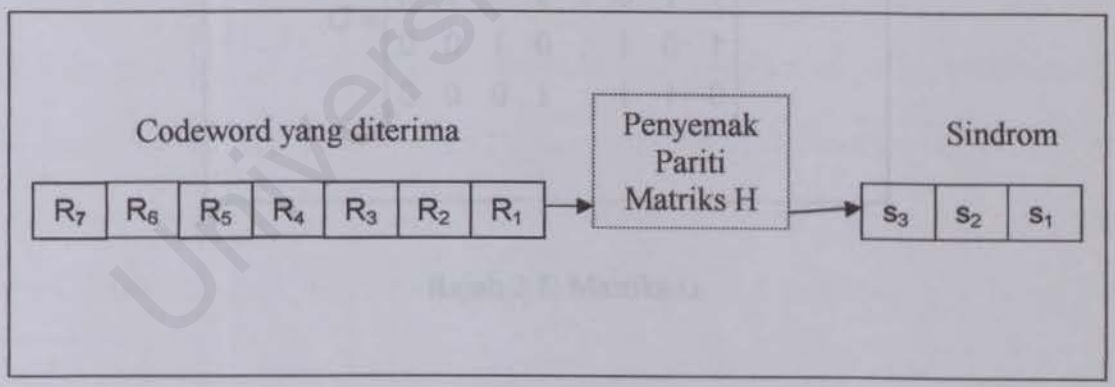
2.7. Encoder dan Decoder

Bit-bit yang dihantar akan melalui encoder dan decoder . Pada bahagian encoder, k bit maklumat yang diterima akan ditambah dengan n-k bit pariti yang ditakrifkan dalam matriks penjana, G. Hasil gabungan antara k bit maklumat dengan n-k bit pariti akan menghasilkan n bit *codeword.* Blok n bit *codeword* akan dihantar kepada decoder di bahagian penerima melalui saluran komunikasi. Dalam Rajah 2.6 ditunjukkan bagaimana n bit *codeword* terhasil setelah dijanakan oleh kod penjana atau matriks G. Apabila *codeword* terbentuk maka ianya akan dihantar ke komputer penerima. Di komputer penerima decoder yang akan menerima *codeword* dan akan menyahkodkannya untuk menghasilkan 4 bit output. Sebelum membuang bit parity, decoder akan menggunakan bit pariti ini untuk mengesan kedudukan ralat dalam bit yang dihantar dan seterusnya membetulkan ralat tersebut.



Rajah 2.6: Encoderan bit maklumat untuk menghasilkan *codeword*

Decoder menggunakan Matriks Penyemak Pariti (Parity Check Matriks) iaitu matriks H untuk mendapatkan kedudukan ralat. Rajah 2.7 menunjukkan bagaimana *codeword* menghasilkan sindrom dengan menggunakan matriks H. Sindrom merujuk kepada kedudukan ralat.



Rajah 2.7: Penghasilan sindrom dalam decoder

2.8. Matriks Penjana (Matriks G) dan Matriks Penyemak Pariti (Matriks H)

2.8.1. Matriks Penjana (Matriks G)

Codeword bagi *Hamming Codes* dijana dengan mendarabkan bit data dengan penjana G menggunakan arithmatik modulo-2. Hasil pendaraban ini dipenggil vektor *codeword* (C1, C2, and C3.....Cn) di mana ia terdiri daripada bit data asal dan bit pariti yang dijana. Matrik penjana G yang di gunakan terdiri daripada matriks identiti, I dan matriks penjana pariti, P. Matriks penjana di rumuskan sebagai G=[I : P]. P adalah k x (n-k) simbol matriks penyemak pariti di mana bagi *Hamming Codes* (7 ,4) P adalah 4 x 3 yang membawa maksud 4 baris dan 3 lajur. Rajah 2.8 menunjukkan bentuk yang mungkin bagi matriks G (7, 4) *Hamming Codes*.

$$G=\left[\begin{array}{cccc:ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

Rajah 2.8: Matriks G

Pendaraban antara 4 bit vektor (I1, I2, I3, I4) dengan matriks G akan menghasilkan keputusan 7 bit *code vector* (I1, I2, I3, I4, P1, P2, P3). P dalam matriks penjana bertanggungjawab untuk menjana tindakan pariti bit. Setiap lajur dalam P di

tunjukkan oleh bit pariti yang dikira dalam subset I. Jika I= (I1, I2 ...I $n-_1$) adalah mesej yang hendak dikodkan, maka *codeword yang* terhasil adalah:

$$C = I * G$$

$$= (I1, I2 \ldots I_{n-1}) * \begin{pmatrix} g0 \\ g1 \\ \cdot \\ gn \end{pmatrix}$$

2.8.2. Matriks Penyemak Pariti (Matriks H)

Matriks H adalah hasil daripada pertukaran matriks G. Dalam vektor 7 bit *codeword*, r = (I1, I2 ,I3, I4, P1 ,P2, P3) akan dihantar ke matriks H. Pendaraban antara 7 vektor dengan matriks H akan menghasilkan sindrom atau vektor penyemak pariti yang digambarkan dalam bentuk s. Bentuk am matriks H adalah H =[PT: I]. Jika pendaraban H* r = s menghasilkan elemen 0 maka *codeword*, r yang diterima adalah sama seperti *codeword* yang dihantar iaitu tiada ralat yang berlaku. Jika pendaraban H*r = s menghasilkan nilai bukan 0 maka bit yang dihantar telah mengalami ralat. Bit yang ralat dapat dikesan atau disemak dengan melihat penyemak pariti mana yang gagal dalam ralat bit tunggal. Rajah 2.9 menunjukkan bentuk matriks H bagi *Hamming Codes* (7, 4).

$$H = \left[\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Rajah 2.9: Matriks H

2.9. Kaedah pengesanan dan pembetulan ralat oleh decoder

2.9.1. Pengesanan ralat

Katakan bit yang dihantar oleh encoder adalah I = [1 0 0 1 0 0 1] dan pada bahagian penerima , bit vektor yang diterima adalah r = [1 0 0 1 0 0 1] maka hasil yang diperolehi adalah seperti :

$$H * r = s$$

$$\begin{pmatrix} 1011 & :100 \\ 1101 & :010 \\ 1110 & :001 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Nilai sindrom yang diperolihi adalah (s1, s2, s3) = (0 0 0). Ini menunjukkan tiada ralat yang berlaku dalam penghantaran bit data. Katakan bit yang dihantar oleh encoder adalah I = [1 0 0 1 0 0 1] dan bit yang diterima adalah r = [1 0 1 1 0 0 1]. Maka pengiraannya adalah seperti:

$$H * r = s$$

$$\begin{pmatrix} 1011 : 100 \\ 1101 : 010 \\ 1110 : 001 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Oleh kerana nilai sindrom adalah tidak bersamaan dengan 0, maka ini menunjukkan ralat telah berlaku pada bit yang dihantar. Untuk mengetahui kedudukan ralat pada bit vektor, maka s = [1 0 1] ditukarkan ke dalam nombor decimal dan akan menghasilkan 5. Ini menunjukkan bit yang telah mengalami ralat berada pada kedudukan yang kelima.

2.9.2. Pembetulan ralat

Untuk membetulkan ralat yang berlaku maka kedudukan ralat perlu dikesan. sindrom yang diperolihi menunjukkan kedudukan ralat tersebut. Bagi contoh di atas,

nilai sindrom adalah 1 0 1. Corak ralat bagi sindrom di atas adalah e= 0 0 1 0 0 0 0 iaitu pada kedudukan kelima ralat telah berlaku. Untuk membetulkannya, maka bit vektor yang mengalami ralat akan di tambah dengan corak ralat. Maka

$$I^* = r + e$$
$$= (1011001) + (0010000)$$
$$= (1001001)$$

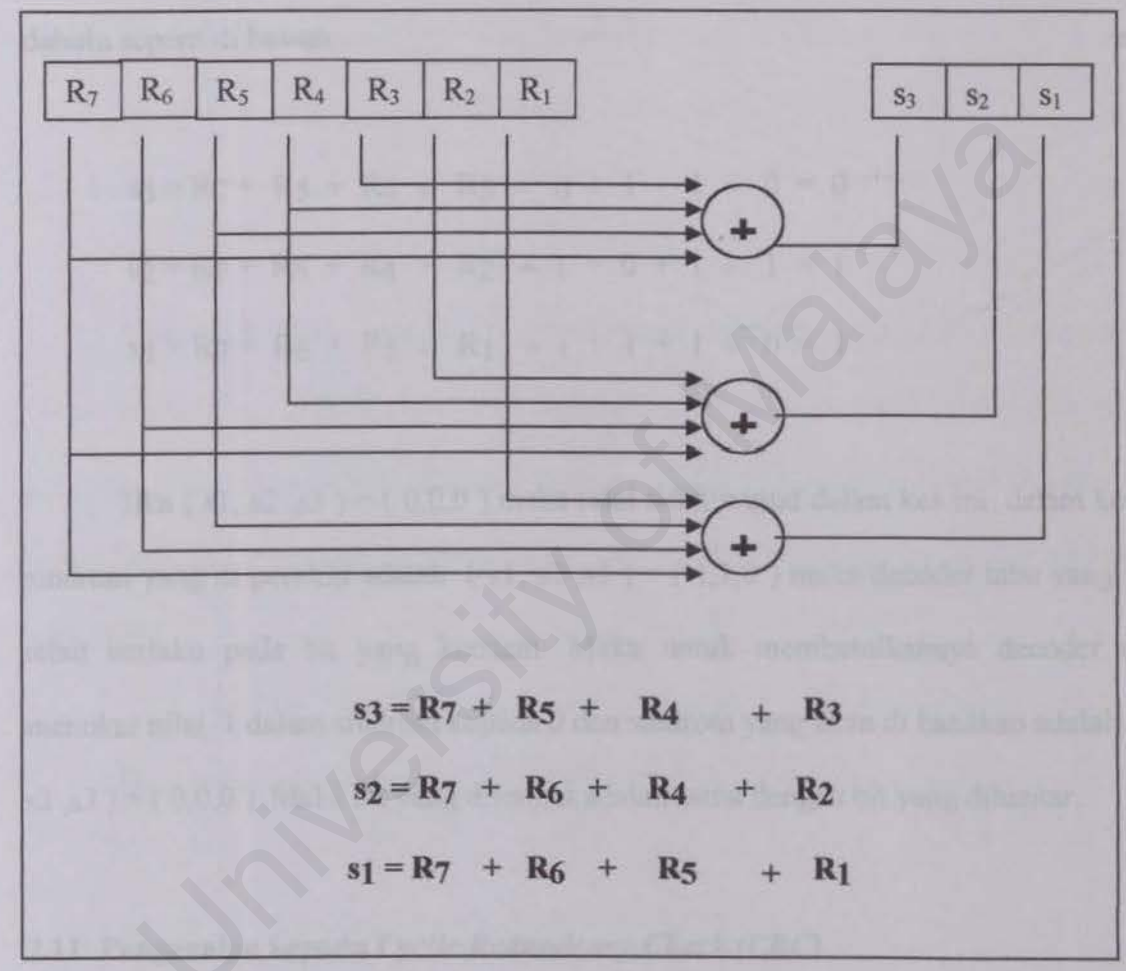
Ralat telah di betulkan dimana I* = I seperti yang dihantar oleh komputer penghantar. Corak ralat adalah seperti jadual 2.1.

Jadual 2.1: Corak Ralat

SINDROM	CORAK RALAT
0 0 0	0 0 0 0 0 0 0
1 0 0	1 0 0 0 0 0 0
0 1 0	0 1 0 0 0 0 0
0 0 1	0 0 1 0 0 0 0
1 0 0	0 0 0 1 0 0 0
0 1 1	0 0 0 0 1 0 0
1 1 1	0 0 0 0 0 1 0
1 0 1	0 0 0 0 0 0 1

2.10. Implementasi melalui Get XOR dalam decoder

Pengesanan dan pembetulan ralat juga boleh di gambarkan dalam bentuk operasi XOR di mana sindrom akan diperolihi dengan melakukan operasi XOR ke atas *codeword* yang diterima seperti dalam Rajah 2.10:



Rajah 2.10: Litar XOR bagi decoder dan sindrom

Blok data yang di terima adalah (R7, R6, R5, R4, R3, R2, R1) tidak sama dengan blok data yang dihantar (I4, I3 ,I2 ,I1 ,P 3, P2 ,P1). Katakan *codeword* yang dihantar adalah I = [0 0 11 0 10] dan bit yang diterima adalah r = [1 0 1 1 0 1 0], di sini dapat dilihat yang ralat telah berlaku pada bit yang ke-6. Bagaimanapun blok decoder tidak tahu bentuk asal data yang dihantar. Jadi untuk memastikan data yang diterima adalah sama seperti data yang dihantar maka perlu dapatkan sindrom terlebih dahulu seperti di bawah:

$$s_3 = R_7 + R_5 + R_4 + R_3 = 0 + 1 + 1 + 0 = 0$$

$$s_2 = R_7 + R_6 + R_4 + R_2 = 1 + 0 + 1 + 1 = 1$$

$$s_1 = R_7 + R_6 + R_5 + R_1 = 1 + 1 + 1 + 0 = 1$$

Jika (s1, s2 ,s3) = (0,0,0) maka ralat tidak wujud dalam kes ini .dalam kes ini sindrom yang di perolihi adalah (s1, s2 ,s3) = (1,1,0) maka decoder tahu yang ralat telah berlaku pada bit yang keenam. Maka untuk membetulkannya decoder akan menukar nilai 1 dalam sindrom kepada 0 dan sindrom yang akan di hasilkan adalah (s1, s2 ,s3) = (0,0,0) .Maka bit yang diterima adalah sama dengan bit yang dihantar.

2.11. Pengenalan kepada *Cyclic Redundency Check (CRC)*

Cyclic Redundency codes (CRC) merupakan kod pengesanan ralat yang sangat popular dalam skim penghantaran data. *Cyclic Redundency Check (CRC)* atau penyemak lewahan berkitar adalah satu bentuk pengiraan matematik ke atas blok data dan akan

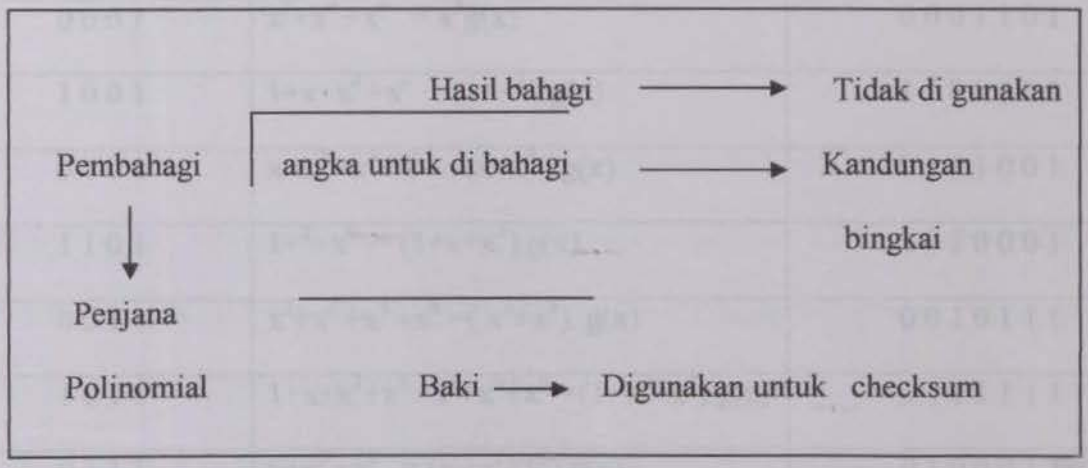
mengembalikan kandungan data tersebut di mana kaedah pengesanan ralat ini akan menambah kod 16 bit pada paket yang dihantar untuk mengetahui maklumat yang di hantar adalah benar atau salah. Hasil tindakan CRC ke atas data akan menghasilkan *checksum*. Bila perbandingan di buat ke atas blok data yang telah di *checksum* dengan blok data *checksum* yang lain, maka pengesanan terhadap ralat dalam data boleh di buat dengan cara melihat samaada data yang dihantar sama ataau tidak dengan data yang diterima. Kaedah ini banyak di gunakan dalam protokol rangkaian.

2.12. Polinomial

Arithmatik CRC merujuk kepada arithmatik polinomial modulo-2 di mana dalam *Cyclic Redundency Check (CRC)*, pembahagi perlu dipilih untuk digunakan dalam pengiraan CRC. Pembahagi dalam pengiraan CRC dipanggil polinomial atau poli. Dalam kes CRC, binari adalah merujuk kepada nombor binari. Di bawah adalah bentuk polinomial.

$$G(X) = X^7 + X^4 + X^3 + X^1 + X^0 \quad = 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1$$

Polinomial terdiri dalam pelbagai saiz. Polinomial yang paling popular digunakan adalah polinomial yang menggunakan panjang bit 16 dan 32. Panjang sesuatu polinomial ditentukan oleh bit yang bernilai 1 yang paling terkiri. Dalam contoh di atas bit paling terkiri adalah X^7. Rajah 2.11 menunjukkan bentuk keadaan pembahagian polinomial.



Rajah 2.11: Pembahagian polinomial

Jadual di bawah menunjukkan jadual encoderan polinomial. Dari nilai kod vektor dan kod polynomial, baki akan di perolihi di mana baki ini merujuk kepada *checksum.*

Jadual 2.2 : Jadual encoderan CRC (7,4 kod berkitar yang dijana oleh $1+x+x^3$)

Baki	Kod polinomial	Kod vektor
0 0 0 0	$0 = 0.\ g(x)$	0 0 0 0 0 0 0
1 0 0 0	$1+x^2+x^3 = 1.g(x)$	1 1 0 1 0 0 0
0 1 0 0	$x^2+x^3+x^4 = x.g(x)$	0 1 1 0 1 0 0
1 1 0 0	$1+x^2+x^3+x^4 = 1+ x.g(x)$	1 0 1 1 1 0 0
0 0 1 0	$x^2+x^3+x^5 = x^2.g(x)$	0 0 1 1 0 1 0
1 0 1 0	$1+ x+x^3+ x^5 = 1+x^5.g(x)$	1 1 1 0 0 1 0
0 1 1 0	$x+x^2+x^3+x^4 = (x+x^2).g(x)$	0 1 0 1 1 1 0
1 1 1 0	$1+ x^4+x^5 = (1+ x+x^2).g(x)$	1 0 0 0 1 1 0

0 0 0 1	$x^3+x^4+x^6 = x^3g(x)$	0 0 0 1 1 0 1
1 0 0 1	$1+x+x^4+x^6 = (1+x^3).g(x)$	1 1 0 0 1 0 1
0 1 0 1	$x+x^2+x^3+x^6 = (x+x^3).g(x)$	0 1 1 1 0 0 1
1 1 0 1	$1+^2+x^6 = (1+x+x^3).g(x)$.	1 0 1 0 0 0 1
0 0 1 1	$x^2+x^4+x^5+x^6 = (x^2+x^3).g(x)$	0 0 1 0 1 1 1
1 0 1 1	$1+x+x^2+x^3+x^4+x^5+x^6 = (1+x^2+x^3).g(x)$	1 1 1 1 1 1 1
0 1 1 1	$x+x^5+x^6 = (x+x^2+x^3).g(x)$	0 1 0 0 0 1 1
1 1 1 1	$1+x^3+x^5+x^6 = (1+x+x^2+x^3.).g(x)$	1 0 0 1 0 1 1

2.13. Pengesanan ralat oleh CRC

Cyclic Redundency Check sangat berguna dalam pengesanan dan pembetulan ralat kod CRC di mana ia berasaskan kepada bit yang ditunjukan oleh polinomial. Contohnya 1 1 0 0 0 1 mempunyai 6 bit, maka penjanaan polynomial di tunjukkan oleh:

$$G(X) = X^5 + X^4 + 1$$

Untuk memudakan pemahaman maka di umpukkan setiap operasai kepada berikut:

- M(X) – Input data
- G(X) – penjana polynomial
- R(X) – Baki
- W(X) – hasil M(X) + R(X)
- Q(X) – Hasil Bahagi

- E(X) – corak ralat

Arithmetik polinomial dilakukan oleh modulo–2 berdasarkan kepada teori algebra. Untuk mendapatkan *cheksum* pada akhir bingkai M(X), maka perlulah membahagikan bingkai yang diterima W(X) dengan pembahagi atau penjana polinomial, G(X). Jika pembahagian W(X) / G(X) adalah 0 maka tiada ralat dikesan dalam data yang di hantar melalui penghantaran sistem .Jika penghantaran mengalami ralat maka hasil pembahagian mennjukkan nilai selain daripada 0 .

2.13.1. Pengesanan data yang tiada ralat

Katakan M(X) = 1 0 1 0 1 1 1 dan G(X) = X5 + X4 + 1 = 1 1 0 0 1

M(X) perlu di XOR dengan 0 0 0 0 dan ini akan menghasilkan 1 0 1 0 1 1 1 0 0 0 0. Hasilkan tersebut perlu dibahagikan dengan penjana, G(X). Berikut adalah pengiraan

```
                 Q(x) =  1 0 0 1
              ------------------------
      10011  |  1 0 1 0 1 1 1 0 0 0
                1 0 0 1 1
                ---------
                  1 1 0 1 1
                  1 0 0 1 1
                  ---------
                    1 0 0 0 0
                    1 0 0 1 1
                    ---------
                        1 1 0 0  = R(x)
```

Hasil baki R(X) akan diletakkan dibahagian akhir bingkai iaitu W(X) = M(X) XOR R(X) menghasilkan 1 0 1 0 1 1 1 1 1 0 0 sebelum menghantar ke bahagian penerima. Katakan T(X) adalah perkataan yang dihantar. Di bahagian penerima baki R(X) akan diperolihi hasil dari operasi XOR di antara T(X) dan G(X). Jika R(X) = 0 0 0 0 maka T(X) yang diterima sama dengan W(X) pada data yang dihantar oleh penghantar. Ini bermakna tiada ralat berlaku.

2.13.2. Pengesanan data yang mempunyai ralat

Jika T(X) yang diterima adalah 1 0 1 0 1 1 0 1 1 0 0. Bila operasi XOR di kenakan keatas T(X) dengan G(X) maka hasil baki seperti berikut akan di perolihi:

```
                1 1 0 0 0 0 1
            ----------------------------------
1 1 0 0 1 | 1 0 1 0 1 1 0 1 1 0 0
            1 1 0 0 1
            -----------
              1 1 0 0 1
              1 1 0 0 1
              ---------
                0 1 1 0 0 0
                  1 1 0 0 1
                  ---------
                    0 0 0 1 = R(x)
```

Oleh kerana baki bagi R(X) adalah 0 0 0 1 dan tidak sama dengan nilai 0 0 0 0 maka corak ralat telah terhasil iaitu, E(X). Maka ini menunjukkan yang ralat telah dikesan.

2.13.3. Pembetulan ralat

Untuk membetulkan ralat yang berlaku, maka apabila ralat dikesan komputer yang menerima data yang ralat akan menghantar semula data tersebut pada bahagian penghantar. Di bahagian penghantar data tersebut akan di hantar semula dengan melaksanakan semula kaedah-kaedah pengesanan ralat.

2.14. Aplikasi *Hamming Codes* dan *Cyclic Redundency Check*

Aplikasi *Hamming Codes* dan *Cyclic Redundency Check* biasanya boleh dilakukan dalam storan seperti memori komputer (RAM) dan dalam magnetic serta storan data seperti Hard disk dan CD ROM's. Selain itu ia juga boleh diaplikasikan dalam sistem komunikasi seperti satelit dan rang komunikasi, rangkaian komunikasi sperti TCP/IP Protokol suite, rangkaian telefon cellular dan audio digital dan penghantaran video.

Pengesanan dan pembetulan ralat Hamming biasa digunakan dalam memori iaitu dalam SRAM. Dalam SRAM, ralat dijana dalam simpanan data. Ini biasanya berlaku dalam penghantaran data melalui satelit. Ini kerana *Hamming Codes* tidak dapat mengesan ralat dengan sempurna. Ralat juga berlaku dalam pemindahan data dari CPU

ke memori. Untuk memastikan pemindahan data dari CPU ke SRAM tiada ralat, maka pengesanan dan pembetulan ralat *Hamming codes* digunakan.

Dalam operasi pemindahan data dari CPU ke memori, ini terdapat dua kitar iaitu kitar baca dan kitar tulis di mana dalam kitar baca, data di baca dari tatasusunan memori ke penimbal. Tujuan pembacaan ini adalah untuk memeriksa nilai bit input data. Jika ralat di temui ketika pengesanan dilakukan, maka data akan dihantar ke penimbal dan seterusnya ke bas data bagi membolehkan data tersebut dihantar keperanti output. Dalam kitaran tulis pula, data dari persekitaran bas di hantar ke unit pengesanan dan pembetulan ralat . Semakan ke atas bit dilaksanakan seperti yang di sebelum ini. Bagi komputer dalam satelit, RAM yang digunakan adalah berbeza dengan RAM yang digunakan dalam komputer biasa di mana dalam satelit RAM yang digunakan adalah RAM yang mempunyai *high-desinty-byte-wide.*

Selain itu teknik Hamming banyak digunakan dalam sistem komersial. Antara syarikat yang menggunakan teknik ini adalah IBM. Dalam *IBM 30XX AND 43XX* kaedah pariti dan pelengkap digunakan. Dalam *UNIVAC 1100/60*, jika terdapat 2 ralat maka isyarat yang mempunyai ralat dihantar kepada peranti peminta. Bagi CRAY-XMP & YMP 8 bit *codeword* digunakan dalam pengesanan dan pembetulan ralat.

Cyclic Redundency Check banyak digunakan untuk aplikasi tertentu pada masa kini. Ini kerana kod ini mudah untuk dilaksanakan terutama dalam perkakasan elektronik. Penggunaannya dalam skim encoderan dan decoderan adalah sangat efisien. Walaupun teknik ini hanya dapat mengesan ralat sahaja, namun ia boleh dihubungkan

dengan mana-mana kod yang lain untuk mrlaksanakan aktiviti pembetulan ralat. Berikut adalah beberapa aplikasi yang dibagunkan melalui teknik pengesanan dan pembetulan ralat oleh CRC.

i. *Ultra 160 SCSI*

Bagi *Ultra 160 SCSI* pengeluaran bagi setiap saluran adalah 160 Mbps hingga 320 Mbps bagi setiap pertukaran masa. Dalam cip ini CRC membaiki realibiliti penghantaran data SCSI semasa komunikasi.CRC juga menyediakan perlidungan kepada data tambahan bagi peranti dalaman. Algoritma CRC juga boleh digunakan dalam FDDI, Ethernet, dan saluran kaca bagi tujuan mengesan ralat bit tunggal, ralat bit berbilang, ralat nombor genap dan ralat *Burst*. Algoritma ini hanya digunakan untuk panjang bit yang lebih daripada 32 bit sahaja.

ii. *Ethernet*

Aplikasi CRC turut digunakan dalam Ethernet di mana 32 bit di gunakan untuk pengesahan bingkai maklumat Ethernet. Oleh itu, *Frame Check Sequence* (FEC) yang di jumpai menggunakan penjana polinomial untuk mentakrifkan masalah matematk. Saiz minimun bagi bingkai Ethernet adalah 72 bait dan saiz maksimum pula adalah 1526 bait.

2.15. Perbandingan antara *Cyclic Redundency Check* dengan *Hamming Codes*

Jadual 2.3 :Perbandingan antara *Hamming Codes* dan *Cyclic Redundency Check*

Hamming Codes	***Cyclic Redundency Check***
• mempunyai satu encoder • tidak bergantung kepada blok sebelumnya • tidak mempunyai *multiplexer* • tidak mempunyai turutan memori • tidak boleh mengesan ralat lebih daripada 2 bit ralat dan hanya boleh membetulkan 1 bit ralat semasa penghantaran • mempunyai kaedah yang agak kompleks	• mempunyai n encoder • .bergantung kepada input blok sebelumnya • mempunyai multiplexer yang berfungsi untuk memilih input dan output • mempunyai turutan memori • hanya boleh melaksanakan pengesanan ralat sahaja iaitu mengesan lebih daripad 2 bit ralat seperti dalam ralat berbilang dan ralat burst • penggunaan XOR dalam CRC adalah hasil dari pengiraan dan ia lebih efisien berbanding Hamming

2.16. Kelebihan dan kekurangan *Hamming Codes* dan *Cyclic Redundency Check*

2.16.1. Kelebihan dan kelemahan *Hamming Codes*

Antara kelebihan *Hamming Codes* ialah:

- *Hamming Codes* mudah dilaksanakan dalam perkakasan kerana perlaksanaanya tidak kompleks.
- Boleh dinyahkodkan dalam masa yang malar dan ia mempunyai kadar data yang tinggi iaitu 0.95.
- Jika terdapat ralat, tidak perlu menghantar semula data ke komputer penerima kerana ralat boleh dibetulkan dengan menggunakan kaedah pembetulan ralat.
- Oleh kerana pembetulan ralat dilakukan di komputer penerima, maka ini dapat mengurangkan *overhead* disebabkan tidak perlu menghantar semula data yang ralat.

Kelemahan Hamming *Codes* pula ialah:

- Hanya boleh membetulkan satu ralat sahaja dalam data yang dihantar dan tidak boleh mengesan ralat yang lebih daripada 2.

2.16.2. Kelebihan dan kelemahan *Cyclic Redundancy Check*

Antara kelebihan Cyclic Redundency Check adalah:

- Boleh mengesan ralat berbilang dan ralat burst dalam data yang dihantar
- Penggunaan *shift* daftar memudahkan operasi encoderan dan penyahlodan

Kelemahan teknik pengesanan ralat *Cyclic Redundency Check* ialah:

- Tidak boleh membetulkan ralat yang dikesan di lokasi destinasi
- Perlu menghantar data sebanyak 2 kali ke komputer penghantar jika ingin membetulkan ralat, jadi ini akan meningkatkan *overhead.*

BAB 3
METODOLOGI

3.1 Pengenalan kepada metodologi

Metodologi adalah satu set panduan lengkap yang mengandungi model-model, kemudahan peralatan dan teknik-teknik khusus yang mesti diikuti dalam melaksanakan setiap aktiviti semasa membangunkan sistem. Untuk itu, perkara yang diambil kira sebelum mensintesis peralatan logik adalah melihat kepada kelakuan senibina yang akan dibangunkan. Apa yang perlu diambil berat oleh pembangun adalah bagaimana mahu membangunkan atau membina kod yang boleh disintesiskan, berfungsi dengan baik dan memenuhi tujuan dalam membangunkan sistem dari segi saiz komponen atau sistem, kelajuan dan penggunaan kuasa sesuatu komponen itu.

Apa yang menjadi masalah kepada perekabentuk adalah kesukaran untuk menangani kekompleksan rekabentuk yang semakin meningkat. Selain itu, perekabentuk juga perlu mempunyai kebolehan menggunakan peralatan Automasi Rekabentuk Elektronik (ARE) dengan cekap dalam masa singkat. Kesukaran yang dihadapi menggalakkan permintaan untuk mendapatkan atau menggunakan metodologi moden dalam merekabentuk dan perlaksanaan pengujian sistem yang hendak dibangunkan. Oleh itu dalam merekabentuk cip, bahasa pengaturcaraan VHDL merupakan elemen yang penting kepada metodologi tersebut.

3.2 Sintesis Peringkat Tinggi

Pada masa kini, Pembangunan Rekabentuk–Terbantu Komputer (RTK) di gunakan untuk mengatasi masalah kekompleksan litar dalam sesuatu cip. Penggunaan RTK adalah seiring dengan kekompleksan litar, di mana pembangunannya telah beralih dari mensintesis litar dengan menggunakan transistor ke sintesis litar menggunakan peringkat logik. Dengan bantuan peralatan sintesis peringkat logik, pereka boleh melihat dan melaksanakan perlaksanaan spesifikasi sistem pada peringkat yang lebih tinggi. Kelebihan Sintesis Peringkat Tinggi (SPT) adalah ia dapat mengurangkan bilangan objek yang patut dimanipulasikan oleh pereka dan membolehkan pereka merekabentuk litar yang lebih besar dan kompleks dalam masa yang lebih singkat. Selain itu sintesis peringkat tinggi juga memberi peluang kepada pengatucara atau pereka yang tiada latar belakang berkenaan perkakasan untuk merekabentuk litar dan seterusnya menghasilkan cip.

Oleh itu metodologi rekabentuk yang sesuai yang ingin dipilih oleh seseorang pereka adalah metodologi rekabentuk yang boleh meningkatkan kecekapan rekabentuk seperti metodologi Rekabentuk Peringkat Tinggi (RTK). Metodologi ini mempunyai ciri-ciri seperti pengabstrakan yang tinggi, penganalisaan saling tukar yang cepat, bebas daripada teknologi dan penggunaan semula rekaan.. lanya juga boleh berulang dan keputusan yang dihasilkan adalah terjamin serta perlakuan sistem juga adalah lengkap.

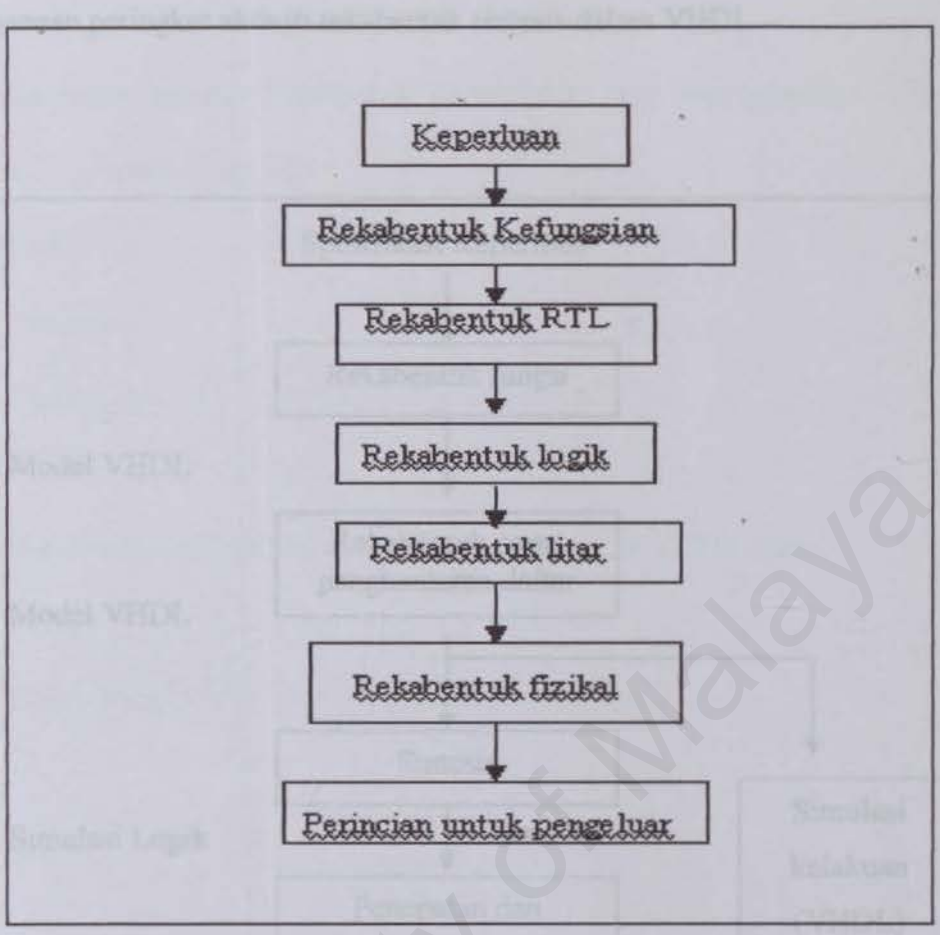
Pada masa kini, hanya 2 bahasa sahaja yang boleh melaksanakan metodologi tersebut di dalam merekabentuk perkakasan logik iaitu VHDL dan Verilog. Verilog

dikatakan kurang meleret daripada VHDL, tetapi perkara ini hanya melibatkan ciri penggunaan bahasa semata-mata. Namun begitu, bagi tujuan dokumentasi, sintesis dan penyelakuan untuk peranti dan sistem, VHDL adalah pilihan terbaik.

3.3. VHDL

VHDL adalah singkatan bagi *Very High Speed Integrated Circuit Hardware Description Languages*. Ia adalah produk yang dihasilkan daripada program VHSIC (Very High Speed Integrated Circuit) yang di tubuhkan oleh *United State Department of Defense (DoD)* dalam tahun 1970-an dan 1980-an. Ia merupakan satu bahasa pengaturcaraan yang boleh digunakan untuk mencadangkan simulasi, pemodelan, percubaan sintesis rekabentuk dan mendokumentasikan sistem digital. Dengan menggunakan VHDL rekabentuk sesuatu cip menjadi lebih mudah berbanding dengan kaedah lama seperti *floor design*, *gate arrangement* dan *schematic capture*. Ini kerana VHDL memberikan format yang sesuai untuk menggambarkan hieraki sesuatu fungsi dan pendawaian sistem digital. Apabila bahasa ini ditubuhkan, ia telah dijadikan sebagai bahasa piawai untuk mendokumentasikan litar kompleks.

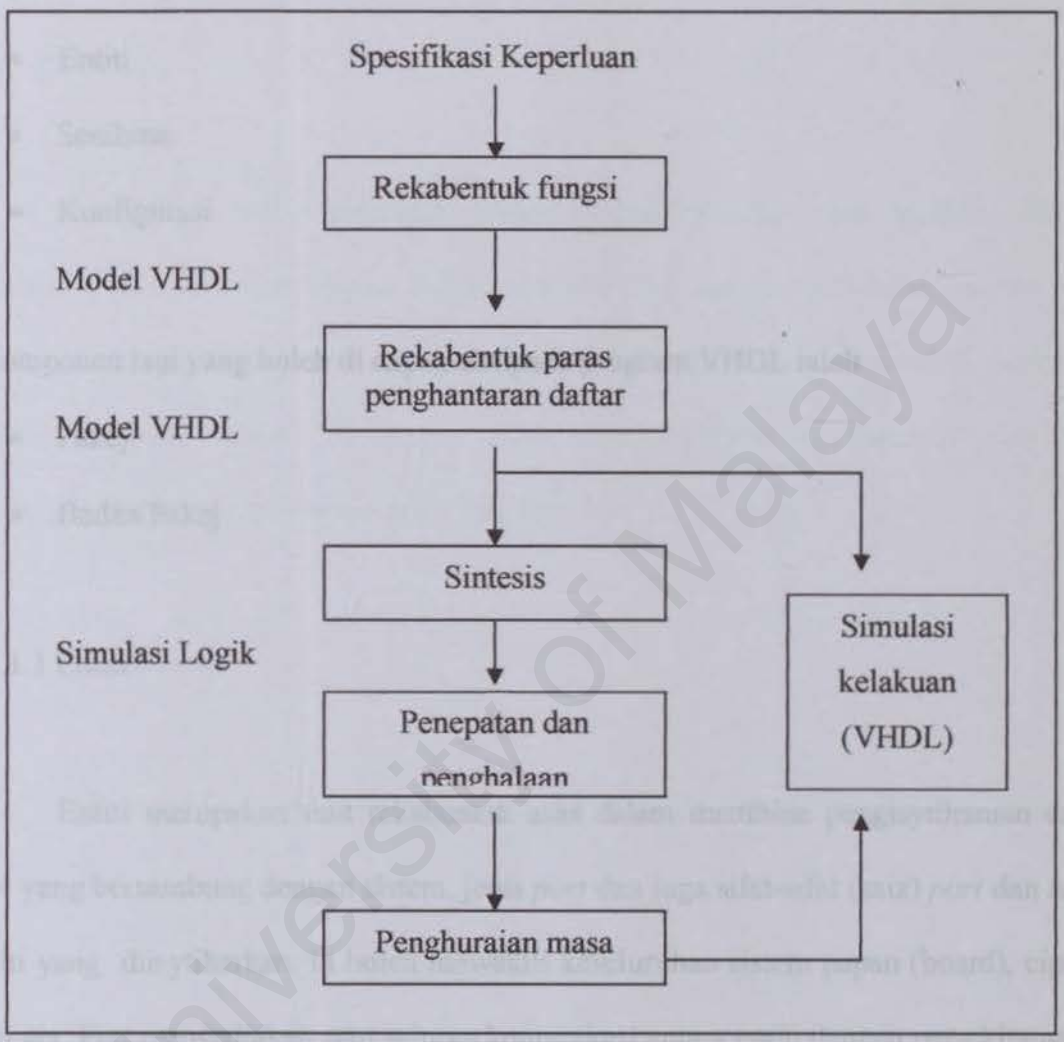
Tujuannya adalah untuk membolehkan kontraktor-kontraktor mempunyai pemahaman yang sama. Ia juga dirancang supaya boleh digunakan sebagai bahasa model (modeling languages) iaitu boleh diproses oleh perisian untuk tujuan simulasi. Dalam menghasilkan output dalam perlaksanaan rekabentuk sistem digital, beberapa peringkat aktiviti–aktiviti berikut akan dilaksanakan seperti Rajah 3.1.



Rajah 3.1: Peringkat aktiviti dalam membangunkan sistem digital

VHDL bukan maklumat model, pangkalan data, penyelaku alat sentesis, metodologi atau set peralatan . Walaubagaimanapun metodologi dan set peralatan adalah sangat mustahak untuk membolehkan VHDL digunakan secara berkesan .Oleh itu penyelidikan yang di lakukan bukan membabitkan bahasa sahaja tetapi juga metodologi dan peralatan penyelidikan terhadap teknologi rekabentuk elektronik digital atau analog mungkin akan menghasilkan peralatan yang membolehkan pereka atau pembina mereka

bentuk carta aliran pada pengabstrakan peringkat tinggi. Rajah 3.2 menunjukkan perlaksanaan peringkat aktiviti rekabentuk sintesis dalam VHDL



Rajah 3.2: Aliran rekabentuk sistesis dalam VHDL

3.3.1 Unit Rekabentuk VHDL

Perlaksanaan sesuatu rekabentuk perkakasan yang menggunakan VHDL terdiri daripada 3 komponen asas iaitu:

- Entiti
- Senibina
- Konfigurasi

2 komponen lagi yang boleh di dapati daripada program VHDL ialah

- Pakej
- Badan Pakej

3.3.1.1 Entiti

Entiti merupakan unit rekabentuk asas dalam membina pengisytiharaan untuk *port* yang bersambung dengan sistem, jenis *port* dan juga sifat-sifat (saiz) *port* dan nama entiti yang diisytiharkan. Ia boleh mewakili keseluruhan sistem papan (board), cip, sel atau get. Port menyediakan satu saluran komunikasi antara entiti dengan persekitarannya Setiap port mesti mempunyai nama, mod dan jenis. Pengisytiharaan bus di buat jika *port* disambung lebih dari satu keluaran dari modul/sistem lain.

Berikut adalah jenis mod yang boleh diisytiharkan dalam penyataan *port* :

- in – masukan ke modul

- out –keluaran daripada modul
- input – isyarat 2 arah (biderectional signal)
- buffer –pendaftar yang di lampirkan (attached) pada keluaran

3.3.1.2 Senibina

Merupakan unit rekabentuk kedua (secondry) dan juga struktur sebenar rekabentuk cip. Ia menerangkan kelakuan (behavior) fungsi rekabentuk.sesuatu entiti rekabentuk boleh mempunyai banyak senibina.. Setiap satunya mewakili perlakuan rekabentuk yang berbeza. Nama port entiti dalam senibina memberikan sambungan (linkage) antara pengisytiharaan entiti dan senibina.

3.3.1.3. Konfigurasi

Dalam unit ini entiti dan senibina boleh digunakan untuk membentuk satu unit rekabentuk .

3.3.1.4 Pakej dan badan Pakej

Pakej adalah satu unit perpustakaan yang terdiri daripada pengisytiharaan yang boleh digunakan dalam unit rekabentuk yang lain. Unit pakej ini terbahagi kepada 2 bahagian iaitu :

- Pengisytiharan

- Badan

Pengisytiharan sesuatu pakej mungkin terdiri daripada nilai tetap,jenis dan pengisytiharan, fungsi dan prosedur. Format ini adalah sama seperti pengisytiharan entiti. Kelakuan fungsi pula dimasukkan kedalam badan pakej.

3.4. Kelebihan dan keburukan VHDL

3.4.1 Kelebihan VHDL berbanding Bahasa yang lain.

i. VHDL digunakan untuk perbincangan diantara satu sama lain berkenaan sesebuah rekabentuk dengan lebih mudah tanpa mendedahkan rekabentuk secara terperinci.

ii. VHDL merupakan bahasa peringkat tinggi yang membolehkan pereka bentuk memperihalkan litar digital yang bersaiz besar dan secara tidak langsung ia mempercepatkan penghasialn produk .Ia menyediakan ruang untuk pembinaan pustaka rekabentuk iaitu kompone-komponen didalamnya boleh digunakan semula dalam rekabentuk berikutnya.

iii. Oleh kerana VHDL adalah bahasa piawai maka ia merupakan bahasa yang mudah alih di antara perkakasan sintesis dan penyelakuan. Ia juga memudahkan pertukaran rekabentuk dari pada logik teratucara ke perlaksanaan IC kegunaan khursus.

iv. VHDL mempunyai cirri-ciri yang menarik yang membenarkan espek-espek elektrikal litar dijelaskan. Contoh espek litar ialah lengahan masa dan operasi.

v. VHDL boleh *capture* prestasi litar dalam bentuk *test bench* .*Test Bench* adalah gambaran VHDL terhadap simulasi litar dan output yang berkenaan.

vi. VHDL dapat menerangkan kelakuan litar yang ingin direka.

3.4.2. Kelemahan VHDL berbanding dengan bahasa pengaturcaraan yang lain

i. Bahasa VHDL adalah mudah dipelajari tetapi susah di kuasai.

ii. Bahasa VHDL agak meleret-leret berbanding dengan bahasa pengaturcaraan Verilog.

3.5. Keperluan Sistem Perisian

3.5.1 WebPACK 4.20

Dalam kajian ini perisian yang digunakan untuk menjana, mensintesis kod-kod VHDL adalah satu perisian yang dinamakan WebPack 4.20. Perisian ini juga sama seperti projek manager yang menggunakan Xilinx di mana ia mengandungi :

1. Keperluan modul rekabentuk
2. Modul penyokong peranti optional "backPACK"
3. Keperluan platfom
4. Dokumentasi

3.5.1.1. Keperluan modul rekabentuk:

Rekabentuk masukan diperlukan sebelum menggunakan WEbPACK. Ia merupakam modul pemasangan sistem perisian WebPack. Di sini salah satu peranti *family implementation modules* seperti CPLD, Virtex or Spartan Fitter boleh digunakan. Dalam perisian ini *design entry* dan *synthesis tools* adalah bahagian utama dalam perisian ini dimana ia digunakan oleh proses-proses sebelum pergi ke bahagian menu-menu berikutnya.

3.5.1.2 CPLD Fitter

Modul peranti CPLD Fitter menyediakan semua peranti dan perisian yang diperlukan untuk memasukkan rekabentuk ke dalam Xilinx. CPLD Fillter boleh digunakan untuk melaksanakan rekabentuk dari dua sumber iaitu :

- Bahasa VHDL, Verilog HDL, atau ABEL.
- EDIF (or XNF)

Modul CPLD Fitter termasuk:

- Perisian CPLD fitter
- Xilinx CPLD families.
- Penjana modul simulasi permasaan CPLD
- Penyunting kekangan .
- Analisis permasaan.

3.5.1.3. Spartan Implementation

Modul perlaksanaan Spartan menyediakan semua peranti pelaksana di mana ia di gunakan untuk untuk memetakan rekabentuk ke dalam keluarga Spartan II atau Spartan-IIE dan perlaksanaanya adalah seperti :

- *Multi-pass place-and-route.*
- *Floorplanner.*
- Menjana model simulasi permasaan FPGA untuk mengeluarkan model VHDL atau Verilog untuk *ModelSim* atau simulator HDL yang lain.
- Penyunting kekangan
- Penganalisa masa

3.5.1.4 Perlaksanaan Virtex

Modul perlaksanaan Virtex menyediakan semua peranti dan perisian yang di perlukan dalam menyokong Virtex-E dan Virtex-II. Sama seperti CPLD Fillter dan Sparta Virtex juga boleh digunakan untuk melaksanakan rekabentuk dari dua sumber iaitu :

- Bahasa VHDL, Verilog HDL yang digunakan dalam modul WebPACK *Design Entry*
- EDIF

Modul perlaksanaan Virtex adalah termasuk :

- *Multi-pass place-and-route.*

- *Floorplanner.*
- Menjana model simulasi permasaan FPGA untuk mengeluarkan model VHDL atau Verilog untuk *ModelSim* atau simulator HDL yang lain.
- Penyunting kekangan.
- Penganalisa masa

3.5.2. Modul Optional "backPACK"

Modul ini mengandungi *ModelSim XE Simulator* di mana ia merupakaan simulator untuk memanipulasikan rekabentuk. *ModelSim XE* lengkap dengan simulasi HDL yang digunakan dalam rekabentuk pogram logik, ia juga membolehkan kod sumber, fungsi dan model masa dikenalpasti bagi rekabentuk tersebut. Bagi WebPACK ini, *ModelSim XE library* di perlukan dimana salah satu daripada model simulasi berikut diperlukan :

- Model simulasi Verilog CPLD
- Model simulasi VHDL CPLD.
- Model simulasi Verilog FPGA.
- Model simulasi VHDL FPGA

3.5.3. Keperluan Platfom

WebPACK dilarikan dalam platfom Pentium-compatible PC tetapi hanya untuk sistem pengoperasian berikut sahaja:

- Windows Windows 2000
- Windows Millennium

- Windows NT 4.0 (Service Pack 5 atau ke atas)
- Windows 98 (original and SE)

3.5.4. Dokumentasi

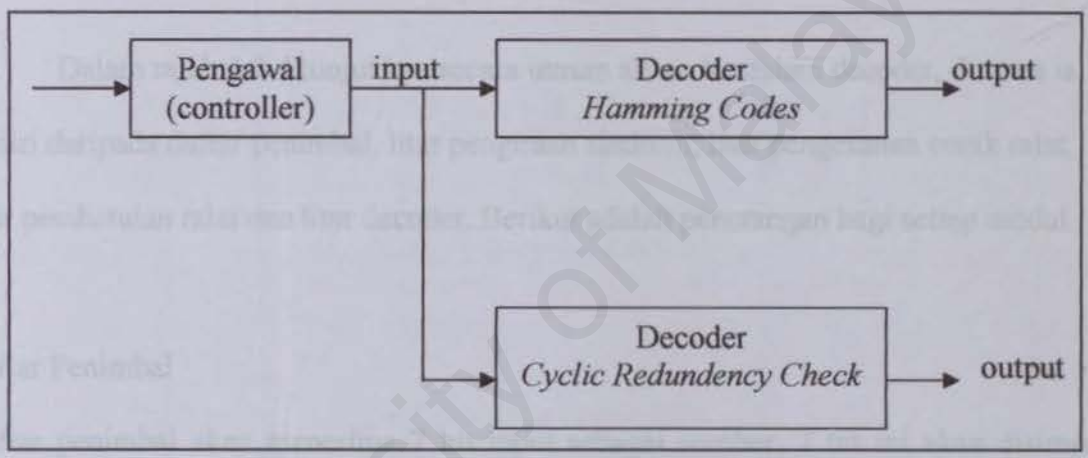
Web PACK menyediakan dokumen *Help* secara online bagi membantu pengguna menggunakan perisian ini di mana ia boleh didapati dengan memilih *ISE Help Contents* dari *menu Help pull-down* dalam Project Navigator, atau memilih *Help and Technical Suppor* dari kumpulan program Xilinx WebPACK .

BAB 4
REKABENTUK SISTEM

4.1. Pengenalan

Bab ini menerangkan tentang cadangan rekabentuk sistem yang akan di bangunkan dalam WXES 3182. Rekabentuk ini meliputi gambarajah blok, gambarajah logik yang terlibat dan gambarajah top level bagi membangunkan sistem.

4.2. Rekabentuk Top Lavel



Rajah 4.1: Top Level bagi rekabentuk decoder *Hamming Codes* dan CRC

Rekabentuk *Top Level* bagi rekabentuk yang dicadangkan adalah seperti dalam rajah 4.1 dimana ia terdiri daripada pengawal (controller), decoder *Hamming Codes* dan decoder *Cyclic Redundency Check codes*. Fungsi pengawal adalah untuk mengawal pemilihan decoder untuk pengesanan dan pembetulan ralat. Terdapat 2 bahagian decoder iaitu decoder yamg menggunakan teknik *Hamming Codes* dan decoder yang menggunakan teknik *Cyclic Redundency Check* di sini, pelaksanaan yang dirancang adalah, jika pegguna memilih masukan '1' maka teknik *Hamming Codes* akan

digunakan untuk mengesan dan membetulkan ralat, sebaliknya nilai '0' dipilih maka teknik *Cyclic Redundency Check codes* yang akan digunakan. Setiap decoder akan mengeluarkan output masing-masing. Rajah 4.3 menunjukkan rekabentuk gambarajah logik bagi decoder *Hamming Codes* dan rajah 4.5 pula adalah menunjukkan gambarajah logik bagi decoder *Cyclic Redundency Check*. Gambarajah logik dan gambarajah blok bagi gabungan antara encoder *Hamming Codes* dan *Cyclic Redundency Check* pula dapat dilihat dalam Rajah 4.6 dan Rajah 4.7.

4.3. Decoder (7, 4) Hamming Codes.

Dalam rajah 4.2 ditunjukkan secara umum aliran bit dalam decoder, dimana ia terdiri daripada daftar penimbal, litar pengiraan sindrom, litar pengesanan corak ralat, litar pembetulan ralat dan litar decoder. Berikut adalah penerangan bagi setiap modul.

Daftar Penimbal

Daftar penimbal akan menerima 7 bit input sebagai sumber. 7 bit ini akan disimpan dalam 7 bit daftar sebelum pengesanan ralat di buat keatas bit yang dihantar.

Litar Pengiraan Sindrom

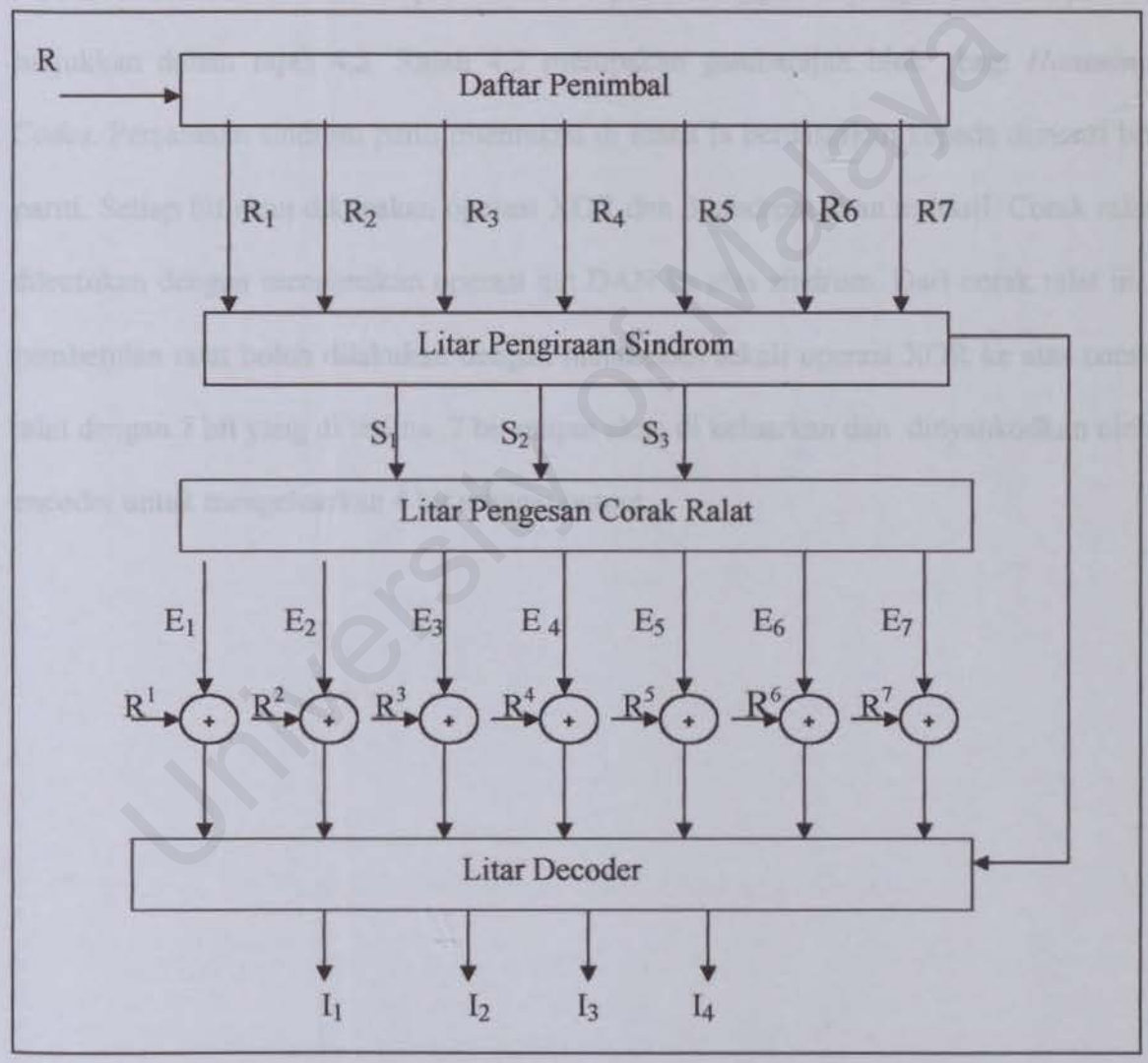
Dalam litar ini 7 bit vektor akan didarabkan dengan dengan H matriks yang terdiri daripada 7 X 3 matriks. Hasil daripada pendaraban antara 7 bit vektor dengan H matriks akan menghasilkan 3 bit vektor yang dipanggil sindrom yang digunakan dalam menentukan kedudukan ralat. Jika terdapat ralat maka corak ralat akan di tentukan, jika tiada ralat maka bit vektor yang dihantar akan dinyahkodkan oleh litar decoder.

Get XOR

Corak ralat yang dihasilkan daripada sindrom akan dikenakan operasi XOR dengan 7 bit vektor yang mengandungi ralat. 7 bit output vektor yang bebas ralat akan terhasil.

Litar Decoder

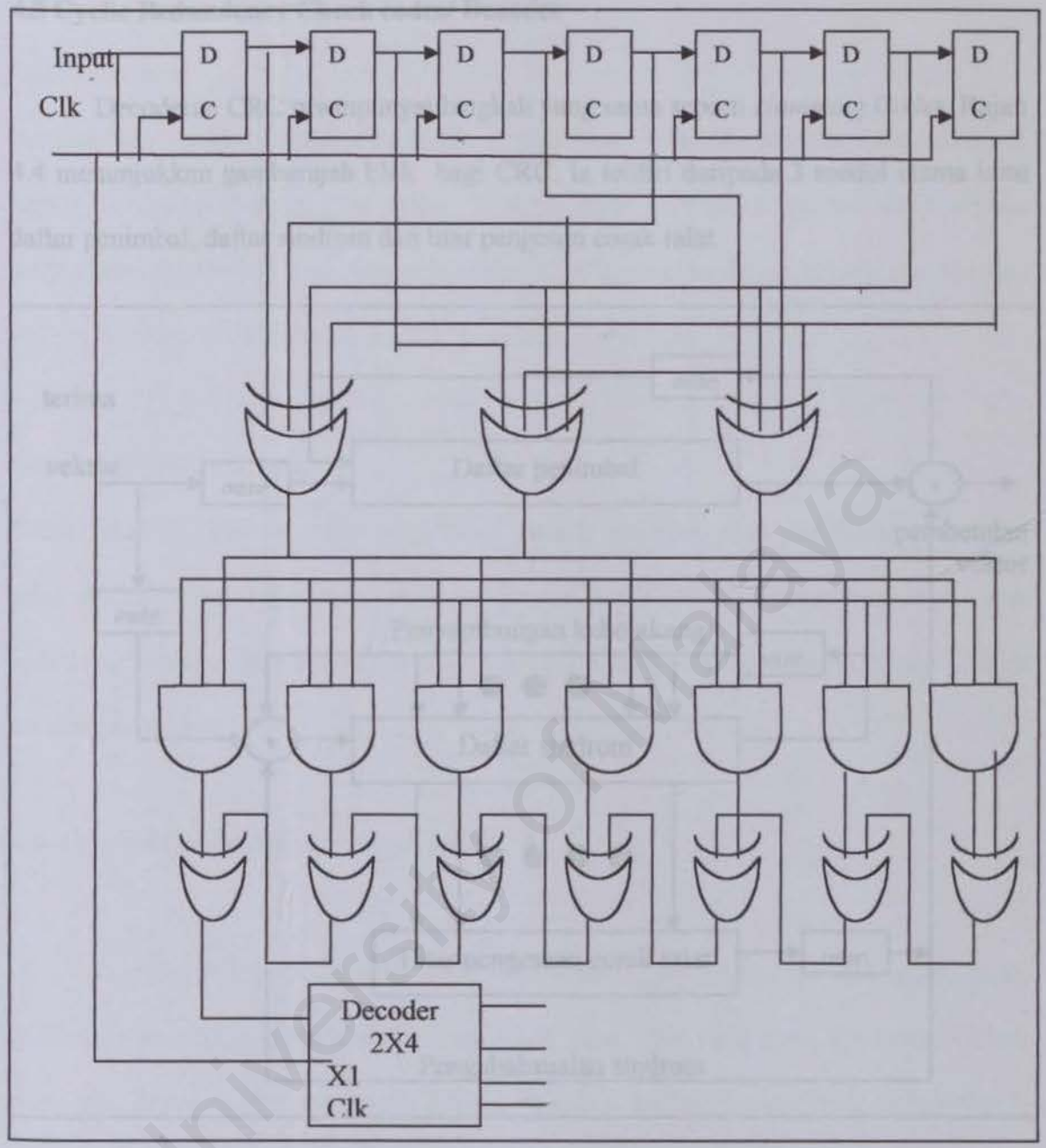
Subsistem decoder menerima 7 bit vector sebagai input. 3 bit pariti akan dinyahkodkan daripada 7 bit vektor untuk mengeluarkan 4 bit vektor sebagai output.



Rajah 4.2: Gambarajah blok diagram Hamming decoder

4.4. Operasi *Hamming Codes*

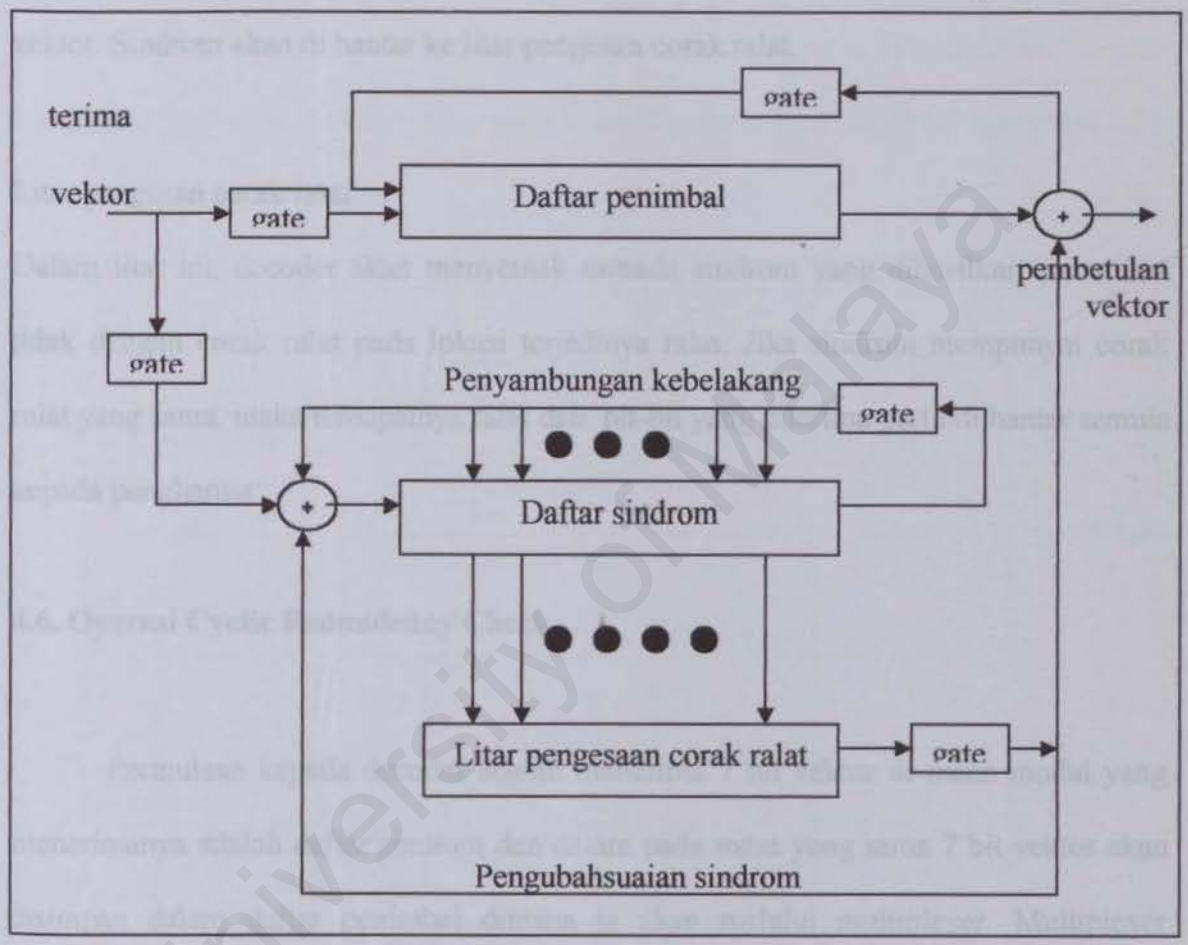
Cip mengandungi input dan output. Ia akan menerima 7 bit vektor dari sumber luar. Input ini akan disimpan dalam 7 bit daftar. 7 bit ini akan dimasukkan satu persatu dalam daftar mengikut T masa. Pada masa T1, bit pertama akan di letakkan dalam daftar R1. Pada T2 pula, bit kedua akan di tempatkan dalam daftar R1 dan bit pertama akan beranjak ke daftar R2. Proses ini berulang sehingga kesemua 7 bit telah dimasukkan ke dalam 7 bit daftar. Proses penerimaan input sehinggalah pengeluaran output di tunjukkan dalam rajah 4.3. Rajah 4.3 merupakan gambarajah blok bagi *Hamming Codes*. Persamaan sindrom perlu ditentukan di mana ia berdasarkan kepada dimensi bit pariti. Setiap bit akan dikenakan operasi XOR dan 3 sindrom akan terhasil. Corak ralat ditentukan dengan mengenakan operasi get DAN ke atas sindrom. Dari corak ralat ini, pembetulan ralat boleh dilakukan dengan melakukan sekali operasi XOR ke atas corak ralat dengan 7 bit yang di terima .7 bit output akan di keluarkan dan dinyahkodkan oleh encoder untuk mengeluarkan 4 bit sebagai output.



Rajah 4.3: Gambarajah Logik Hamming decoder

4.5 Cyclic Redundency Check codes/ Decoder

Decoderan CRC mwmpunyai langkah yang sama seperti *Hamming Codes*. Rajah 4.4 menunjukkan gambarajah blok bagi CRC. Ia terdiri daripada 3 modul utama iaitu daftar penimbal, daftar sindrom dan litar pengesan corak ralat.



Rajah 4.4: Gambarajah blok diagram *Cyclic Redundency Check*

Daftar Penimbal

Dalam daftar penimbal 7 bit vektor akan di terima dimana vertor tersebut akan disimpan buat semantara sebelum di hantar keluar daripada penimbal. Sebelum menerima vektor, daftar mesti di setkan kepada 0.

Daftar sindrom

Semasa 7 bit vektor dihantar ke daftar penimbal serentak dengan itu 7 bit vektor di hantar ke daftar sindrom. Bit vektor tersebut akan dikenakan operasi XOR supaya pengesanan ralat boleh dilakukan. Bit vektor akan ditukar ke dalam bentuk sindrom dan vektor. Sindrom akan di hantar ke litar pengesan corak ralat.

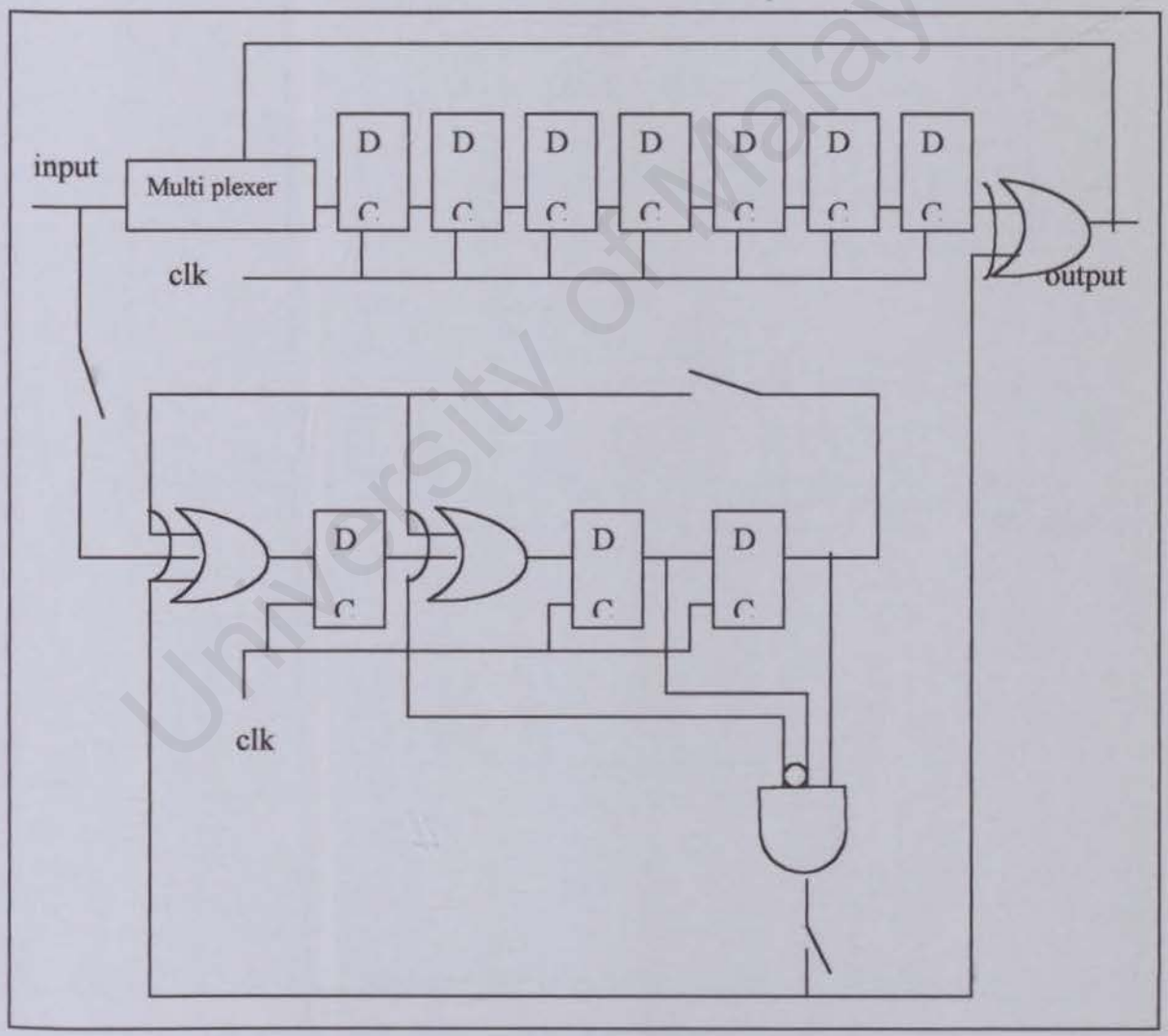
Litar pengesan corak ralat

Dalam litar ini, decoder akan menyemak samada sindrom yang dihasilkan sama atau tidak dengan corak ralat pada lokasi terjadinya ralat. Jika sindrom mempunyai corak ralat yang sama. maka terdapatnya ralat dan bit-bit yang diterima perlu di hantar semula kepada penghantar

4.6. Operasi Cyclic Redundency Check

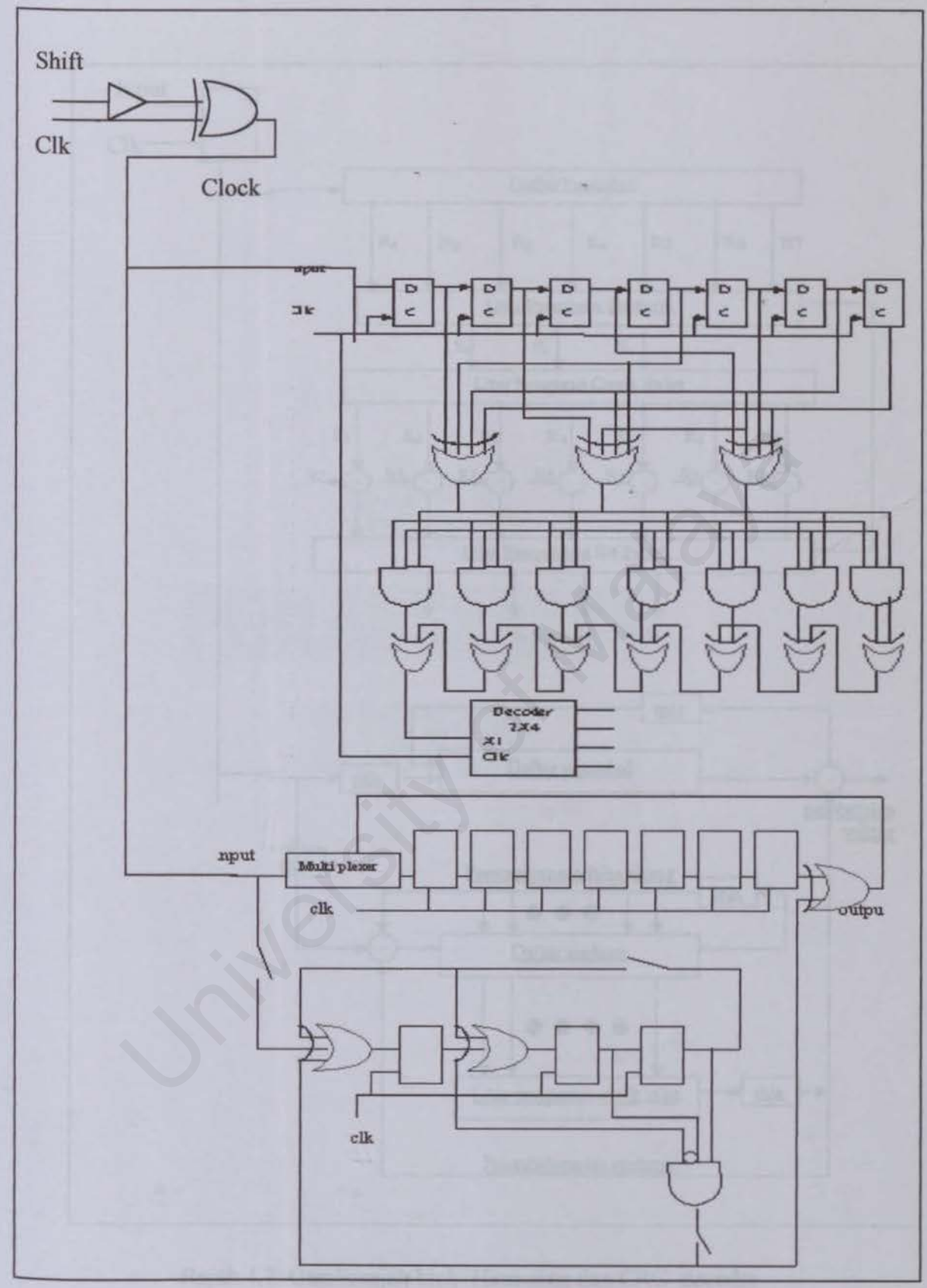
Permulaan kepada decoder adalah menerima 7 bit vektor di mana modul yang menerimanya adalah daftar sindrom dan dalam pada masa yang sama 7 bit vektor akan disimpan dalam daftar penimbal dimana ia akan melalui multiplexer. Multiplexer berfungsi untuk menselarikan nilai input yang akan memasuki daftar penimbal. 7 bit vektor yang dihantar kepada daftar sindrom akan terlebih dahulu di enakan operasi XOR untuk menghasilkan sindrom. Sindrom akan di bentuk apabila vektor dianjak dalam daftar sindrom. Pada bahagian terkanan, output akan dikeluarkan. Jika output adalah 1 maka ralat telah berlaku ,jika 0 tiada ralat dalam aliran data ini.

Bit pertama yang akan diterima akan di baca pada bahagian kanan iaitu sebelah kanan daftar penimbal. Dalam pada masa yang sama nilai bit dalam daftar sindrom akan dianjak. Jika bit pertama yang dikesan mengandungi ralat maka ia akan dibetulkan oleh output decoder. Output decoder juga boleh dihantar semula ke daftar sindrom untuk mengubahsuai sindrom seperti membuang kesan ralat dari sindrom. Nilai baru sindrom akan terhasil di mana ianya sama seperti vektor yang diterima dianjak sebanyak satu kedudukan ke kanan. Sindrom baru yang terhasil akan digunakan untuk mengesan ralat dalam vektor ynag di hantar.

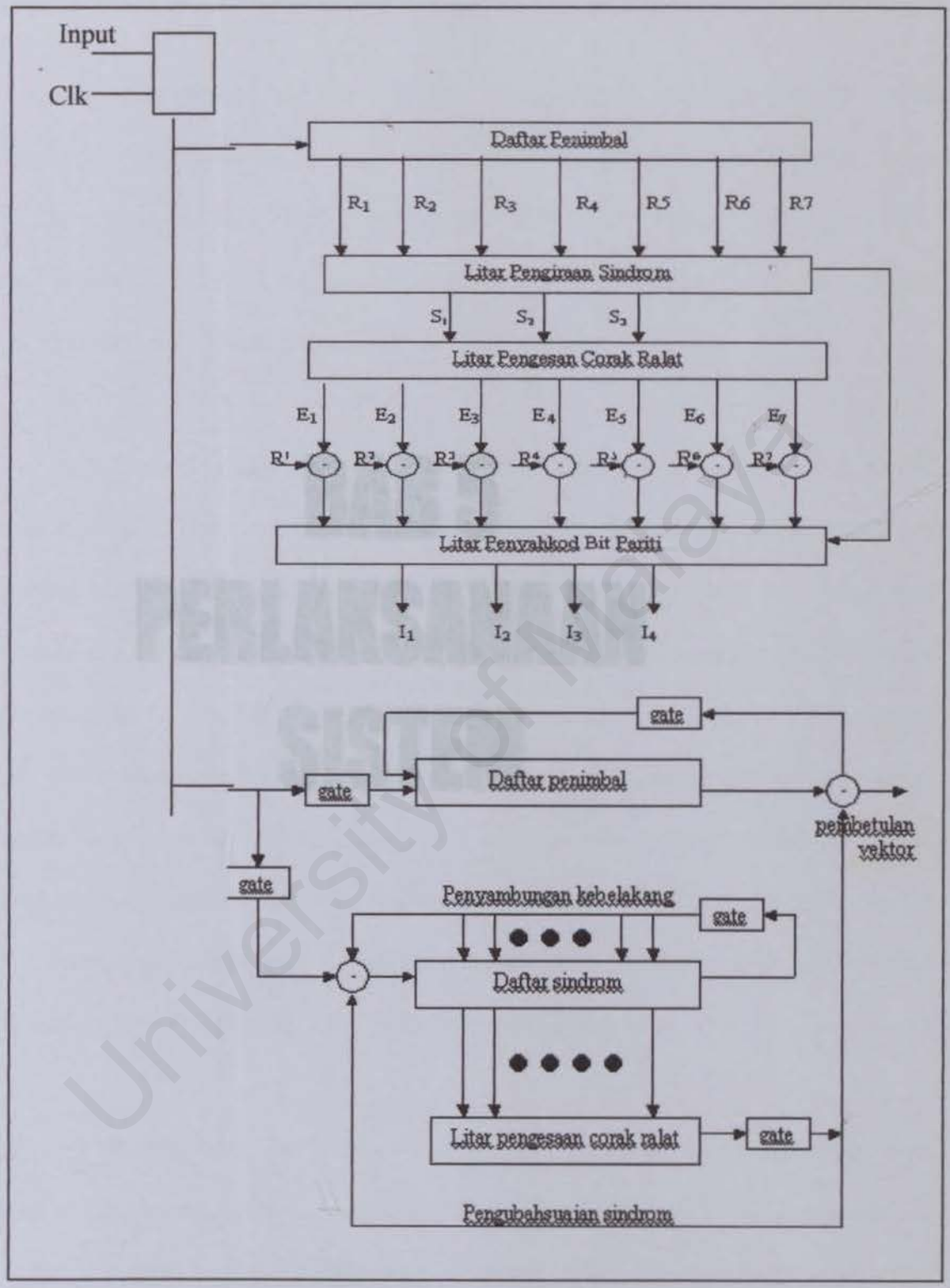


Rajah 4.5: Gambarajah logik bagi *Cyclic Redundency Check*

Pada bahagian kanan daftar penimbal decoder akan ulang langkah-langkah di atas bagi setiap input yang dikeluarkan. Bit kedua yang diterima akan dibetulkan sama seperti bit pertama. Decoder akan menyahkod vektor yang diterima dari satu bit ke satu bit sehingga semua vektor yang diterima keluar dari pada daftar penimbal. Untuk melihat dengan jelas bagaimana decoder berfiungsi dalam CRC maka rujuk kepada Rajah 4.5.



Rajah 4. 6: Gabungan gambarajah logik hamming dan CRC decoder



Rajah 4.7: Gambarajah blok Hamming dan CRC decoder

BAB 5
PERLAKSANAAN SISTEM

Bab 5: Perlaksanaan / Pembangunan Sistem

Perlaksanaan/pembangunan 'Sistem Pengesanan dan Pembetulan ralat menggunakan teknik Hamming Codes dan Cyclic Redundency Checks / Decoder ' akan dihuraikan dalam bab ini. Sebelum penerangan lebih lanjut tentang pergabungan sistem antara Hamming dan CRC menggunakan *controller* akan diterangkan terlebih dahulu tentang perlaksanaan Hamming decoder dan CRC decoder.

Bagi Hamming decoder, sedikit perubahan telah dibuat keatas modul. Untuk mendapatkan nilai Hamming decoder, teknik pengiraan *conventional* bagi bit maklumat digunakan dan bukan menggunakan teknik penjanaan matriks. Oleh itu, dalam litar pengiraan sindrom, nilai H matriks tidak digunakan untuk mendapatkan sindrom tetapi sebaliknya nilai ini diperolehi dengan menggunakan pengoperasian get XOR. Masukkan input untuk Hamming adalah secara sesiri dan bukan secara selari, jadi disini penggunaan *clock* telah dibuang. Bagi CRC decoder pula, terdapat sedikit perubahan dilakukan keatas rekabentuk sistem dimana output terakhir CRC decoder telah ditukar dari 4 bit kepada 7 bit ouput. Ini adalah kerana CRC decoder yang dihasilkan akan memaparkan 7 bit output yang telah mengalami pembetulan ralat.

Seperti yang telah dicadangkan sebelum ini, sebanyak 2 modul digunakan iaitu Hamming decoder dan CRC decoder dan dikawal oleh c*ontroller*, Kedua-dua modul ini telah diterjemahkan kedalam bahasa pengaturcaraan VHDL Berikut adalah penerangan tentang *source codes* bagi Hamming decoder dan CRC decoder yang telah direkabentuk ke dalam arahan-arahan menggunakan bahasa pengaturcaraan VHDL.

5.1. Perlaksanaan Hamming Decoder

5.1.1. *Source codes* bagi Hamming Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity HAMDEC is
    port (
            DI      : in std_ulogic_vector (6 downto 0);
            SY      : out std_logic_vector (2 downto 0);
            DO      : out std_ulogic_vector (6 downto 0);
            DOU     : out std_logic_vector (6 downto 0);
            DOUT    : out std_logic_vector (3 downto 0)
        );
end HAMDEC;

architecture VER1 of HAMDEC is
    signal  ein0,ein1,ein2,ein3,ein4,ein5,ein6 : std_ulogic;
    signal e4,e5,e6 : std_ulogic;

  begin
            e4 <= DI(0) xor DI(3) xor DI(5) xor DI(6);
            e5 <= DI(1) xor DI(3) xor DI(4) xor DI(5);
```

```
e6 <= DI(2) xor DI(4) xor DI(5) xor DI(6);

SY (0) <= e4;   -- keluaran 3 bit sindrom
SY (1) <= e5;
SY (2) <= e6;


ein0 <= (((not e5) and (not e6)) and e4); -- pengiraan corak ralat
ein1 <= (((not e4) and (not e6)) and e5);
ein2 <= (((not e4) and (not e5)) and e6);
ein3 <= (e4 and (e5 and (not e6)));
ein4 <= (((not e4) and e5) and e6);
ein5 <= ((e6 and e5) and e4);
ein6 <= (((not e5) and e4) and e6);


DO (0) <= ein0;
DO (1) <= ein1;
DO (2) <= ein2;
DO (3) <= ein3;
DO (4) <= ein4;
DO (5) <= ein5;
DO (6) <= ein6;

DOU (0) <= ein0 xor DI (0);      -- output yang telah dibetulkan
DOU (1) <= ein1 xor DI (1);
```

```
DOU (2) <= ein2 xor DI (2);
DOU (3) <= ein3 xor DI (3);
DOU (4) <= ein4 xor DI (4);
DOU (5) <= ein5 xor DI (5);
DOU (6) <= ein6 xor DI (6);

DOUT (0) <= ein3 xor DI (3); --kodkan output kepada (7,4)
DOUT (1) <= ein4 xor DI (4);
DOUT (2) <= ein5 xor DI (5);
DOUT (3) <= ein6 xor DI (6);
end VER1;
```

5.1.2. Penerangan *Source codes* bagi Hamming Decoder

Bit maklumat yang dimasukkan adalah sebanyak 7 bit iaitu dengan mewakilkannya dengan pembolehubah DI. Ketujuh-tujuh bit ini dikenakan operasi XOR untuk menghasilkan sindrom dan diumpukkan kedalam signal e4, e5 dan e6 yang mana akan menentukan kedudukan ralat. 3 bit sindrom akan terhasil setelah pengoperasian XOR dilakukan dan 3 bit sindrom ini dikeluarkan sebagai output melalui SY (0), SY (1) dan SY (2) seperti di bawah:

```
e4 <= DI(0) xor DI(3) xor DI(5) xor DI(6
e5 <= DI(1) xor DI(3) xor DI(4) xor DI(5);
e6 <= DI(2) xor DI(4) xor DI(5) xor DI(6);
```

```
SY (0) <= e4;    -- 3 bit sindrom
SY (1) <= e5;
SY (2) <= e6;
```

Apabila sindrom diketahui maka, corak ralat terhadap bit dapat ditentukan dimana pengoperasian menggunakan get NOT dan get DAN dilakukan. Bit-bit ralat yang dihasilkan diumpukkan ke atas pembolehubah ein0, ein1, ein2 dan seterunya. Corak ralat akan dikeluarkan sebagai DO (0), DO (1)..... dan seterusnya.

```
ein0 <= ((( not e5 ) and ( not e6 )) and e4 );  -- 7 bit corak ralat
ein1 <= (((not e4) and (not e6)) and e5);
ein2 <= (((not e4) and (not e5)) and e6);
ein3 <= (e4 and (e5 and (not e6)));
ein4 <= (((not e4) and e5) and e6);
ein5 <= ((e6 and e5) and e4);
ein6 <= (((not e5) and e4) and e6);


DO (0) <= ein0;     -- output ralat
DO (1) <= ein1;
DO (2) <= ein2;
DO (3) <= ein3;
DO (4) <= ein4;
DO (5) <= ein5;
DO (6) <= ein6;
```

Kedudukan corak ralat adalah berdasarkan kepada jadual corak ralat yang terdapat pada jadual 5.2. Disini, bit-bit yang mengalami ralat dapat dibetulkan melalui melakukan operasi XOR ke atas corak bit yang ralat dengan bit maklumat yang dihantar seperti di bawah:

```
DOU (0) <= ein0 xor DI (0);        -- output yang telah dibetulkan
DOU (1) <= ein1 xor DI (1);
DOU (2) <= ein2 xor DI (2);
DOU (3) <= ein3 xor DI (3);
DOU (4) <= ein4 xor DI (4);
DOU (5) <= ein5 xor DI (5);
DOU (6) <= ein6 xor DI (6);
```

4 bit akan terhasil apabila apabila pengekodan dilakukan ke atas 7 bit maklumat yang telah dibetulkan. Hasil akhir akan dipaparkan melalui pembolehubah dibawah:

```
DOUT (0) <= ein3 xor DI (3);  --pengekodan output (7,4)
DOUT (1) <= ein4 xor DI (4);
DOUT (2) <= ein5 xor DI (5);
DOUT (3) <= ein6 xor DI (6);
```

5.2. Perlaksanaan CRC Decoder

5.2.1. *Source codes* bagi CRC Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity crc_decode is
        port (
                Clk, Rst, load  : in std_ulogic;
                Din             : in std_ulogic_vector (6 downto 0);
                syndrome        : out std_ulogic;
                shift2          : out std_ulogic;
                shift1          : out std_ulogic;
                shift0          : out std_ulogic;
                 reg            : out std_ulogic;
                XCRC            : out std_ulogic
                );
end crc_decode;
architecture behavior of crc_decode is
        signal X                        : std_ulogic_vector(6 downto 0);
        signal s_d                      : std_ulogic_vector(6 downto 0);
        signal s_out                    : std_ulogic;
        signal syn, s_syn1, s_reg       : std_ulogic;
```

```
begin
  s_d<= Din;

  process (Clk,Rst)
  begin
    if Rst = '1' then          -- reset (active high) = 0
        X <= (others => '0');
        reg  <= '0';
        XCRC <= '0';
    elsif rising_edge(Clk) then
      if (load = '1') then                    -- mula  load data
        s_d<= '0' & s_d (6 downto 1);
            X (6 downto 3) <= (others => '0');
            X (2) <= X (0) xor s_d (0);
            X (1) <= X (2) xor X (0);
            X (0) <= X (1);
            reg <= s_d(0);                    -- aliran input ke register
            s_out<= syn xor s_d (0);
        s_reg <= s_d (0);
      end if;
    end if;
  end process;
```

```
        syn <= (X(2) and (not X(1))) and X(0);
        s_syn1 <= (X (2) and (not X (1))) and X (0);
        syndrome <= syn;
        XCRC <= s_syn1 xor s_reg;
        shift2 <= X (2);
        shift1 <= X (1);
        shift0 <= X (0);
end behavior;
```

5.2.2. Penerangan Source *codes* bagi CRC Decoder

Dalam perlaksanaan CRC decoder, 7 bit input dimasukkan secara selari. Di sini clock (clk) berfungsi untuk memasukkan bit-bit permasa.

```
if Rst = '1' then          -- reset (active high) = 1
        X <= (others => '0');
        reg  <= '0';
        XCRC <= '0';
```

Dalam *source codes* di atas, reset (Rst) berada dalam *active high* iaitu bila rst=1 semua keluaran akan diset kepada 0 dimana tiada nilai baru akan dimasukkan kedalam penimbal ataupun kedalam pendaftar anjakan. Ini menyebabkan semua keluaran adalah sifar.

```
if (load = '1') then                    -- mula  load data
        s_d<= '0' & s_d (6 downto 1);
```

```
X (6 downto 3) <= (others => '0');
X (2) <= X (0) xor s_d (0);
X (1) <= X (2) xor X (0);
X (0) <= X (1);
reg <= s_d(0);                    -- aliran input ke register
s_out<= syn xor s_d (0);
s_reg <= s_d (0);
```

Pada load =1, jujukan bit akan dimasukkan kedalam penimbal dan juga kedalam 3 pendaftar anjakan yang diwakili dengan X (0), X(1), X(2). Di sini 7 bit input akan dimasukkan secara serentak ke dalam penimbal yang diwakili dengan s_d<= '0' & s_d (6 downto 1) dan juga kedalam pendaftar anjakan yang diwakili dengan pembolehubah X secara bit demi bit. Pada awalnya nilai untuk X (0), X(1) dan X(2) adalah berada dalam keadaan sifar. Bit pertama yang di wakili dengan signal s_d (0) akan memasuki pendaftar anjakan dan akan di kenakan operasi XOR dengan nilai yang terdapat pada pendaftar anjakan yang ketiga iaitu X (2) bagi menghasilkan nilai sementara untuk X (0). Nilai sebelum bagi X (0) akan dikenakan operasi XOR dengan X (2) untuk menghasilkan nilai bagi X (1). Sementara nilai bagi X (2) pula adalah hasil dari anjakan bit pada kedudukan X (1). Nilai keluaran bagi X(0), X(1) dan X(2) di wakilkan dengan shift2, shift1 dan shift0 seperti berikut:

```
shift2 <= X (0);
shift1 <= X (1);
shift0 <= X (2);
```

- Setiap kali nilai pada X (0), X(1) dan X(2) terhasil, 3 bit akan dkeluarkan dan dikenakan operasi logik DAN dan NOT bagi menghasilkan corak ralat. Di sni 7 bit corak ralat akan terhasil dimana jika corak ralat yang dihasilkan adalah bernilai sifar maka tiada ralat yang dikesan tetapi jika terdapat nilai selain daripada sifar, maka ralat telah dikesan dalam jujukan bit. Corak ralat akan menentukan kedudukan bit mana yang mengalami ralat. Corak ralat yang terhasil akan dikeluarkan seperti berikut:

```
syn <= (X(2) and (not X(1))) and X(0);
```

Untuk membetulkan ralat, maka nilai bagi corak ralat yang terhasil akan di XOR dengan nilai masukan pertama penimbal iaitu s_d (0) seperti di bawah:

```
s_out<= syn xor s_d (0);
```

Nilai s_out di umpukkan kepada XCRC bagi memaparkan output terakhir.
Proses akan berulang sehinggalah ketujuh-tujuh bit dikeluarkan sebagai output.

Disini skop perlaksanaan telah diperluaskan dengan:

i. Pergabungan antara top modul Hamming dengan top modul CRC menggunakan *controller*.
ii. Pergabungan antara Hamming encoder dengan Hamming decoder menggunakan *controller*

iii. Menggabungkan CRC encoder dan CRC decoder menggunakan *controller*.

Pada mulanya, modul encoder dan modul decoder dibina secara berasingan tetapi satu alternatif baru dibuat di mana Hamming encoder telah digabungkan dengan Hamming decoder menggunakan *controller* dan CRC decoder pula digabungkan dengan CRC encoder menggunakan *controller*. Disini *controller* bertindak sebagai pemilih mod iaitu samada pengguna ingin menggunakan Hamming encoder, Hamming Decoder, CRC encoder atau CRC decoder

5.3. Gabungan Hamming encoder dan Hamming Decoder

5.3.1. *Source codes* bagi Gabungan Hamming encoder dan Hamming Decoder

5.3.1.1. *Source Codes* Top Level Hamming

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.all;

entity toplevel is
  port (
        operator    : in std_ulogic_vector(1 downto 0);
        datain      : in  std_ulogic_vector(3 downto 0);
        DI          : in std_ulogic_vector (6 downto 0);
```

```
        SY          : out std_ulogic_vector (2 downto 0);
        DO          : out std_ulogic_vector (6 downto 0);
        DOU         : out std_ulogic_vector (6 downto 0);
        result      : out std_ulogic_vector(6 downto 0)
        );
end toplevel;

architecture toplevel_arch of toplevel is

component hamenc    -- encoder untuk hamming
  port (
        datain  : in  std_ulogic_vector(3 downto 0);
        hamout : out std_ulogic_vector(6 downto 0)
        );
end component;

component HAMDEC  --decoder untuk hamming
  port (
        DI      : in std_ulogic_vector (6 downto 0);
        SY      : out std_ulogic_vector (2 downto 0);
        DO      : out std_ulogic_vector (6 downto 0);
        DOU     : out std_ulogic_vector (6 downto 0);
        DOUT    : out std_ulogic_vector (3 downto 0)
  );
```

```
end component;

component mux
  port (
       operator       : in  std_ulogic_vector(1 downto 0);
      host_hamout  : in  std_ulogic_vector(6 downto 0);
       host_DOUT   : in std_ulogic_vector (3 downto 0);
       result         : out std_ulogic_vector(6 downto 0)
  );
end component;

   signal s_result1 : std_ulogic_vector(6 downto 0);
   signal s_result2 : std_ulogic_vector(3 downto 0);

begin   -- portmap untuk input,output, clk utk setiap modul

   U1:  hamenc port map (  datain => datain,
                           hamout => s_result1);

   U2:  HAMDEC port map (  DI   => DI
                           SY => SY,
                           DO => DO,
                           DOU => DOU,
                           DOUT => s_result2);
```

```
    U3: mux  port map (   operator => operator,
                          host_hamout => s_result1,
                          host_DOUT  => s_result2,
                          result => result );
end toplevel_arch;
```

5.3.1.2. *Source Codes* Mux *(controller)* Hamming

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;

entity mux is
    port (
          operator      : in  std_ulogic_vector(1 downto 0;
          host_hamout   : in  std_ulogic_vector(6 downto 0);
          host_DOUT     : in std_ulogic_vector (3 downto 0);
          result        : out std_ulogic_vector(6 downto 0)
         );
end mux;

architecture mux_arch of mux is
    signal result1      : std_ulogic_vector(6 downto 0);
```

```
    signal result2      : std_ulogic_vector(3 downto 0);
    signal res          : std_ulogic_vector(6 downto 0);
    constant c_enc : std_ulogic_vector(1 downto 0) := "01";
    constant c_dec : std_ulogic_vector(1 downto 0) := "10";

begin
    result1 <= host_hamout;
    result2 <= host_DOUT;

    with operator select
        res <=  result1  when c_enc,
                result2 & "ZZZ" when c_dec,
                "0000000" when others;
        result<=res;

end mux_arch;
```

5.3.1.3. *Source Codes* Hamming Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity HAMDEC is
    port (
```

```
        DI    : in std_ulogic_vector (6 downto 0);
        SY    : out std_ulogic_vector (2 downto 0);
        DO    : out std_ulogic_vector (6 downto 0);
        DOU   : out std_ulogic_vector (6 downto 0);
        DOUT : out std_ulogic_vector (3 downto 0)
    );

end HAMDEC;

architecture VER1 of HAMDEC is
    signal ein0,ein1,ein2,ein3,ein4,ein5,ein6 : std_ulogic;
    signal e4,e5,e6 : std_ulogic;

  begin

        e4 <= DI (0) xor DI (3) xor DI (5) xor DI (6);
        e5 <= DI (1) xor DI (3) xor DI (4) xor DI (5);
        e6 <= DI (2) xor DI (4) xor DI (5) xor DI (6);

        SY (0) <= e4;
        SY (1) <= e5;
        SY (2) <= e6;

        ein0 <= (((not e5) and (not e6)) and e4
```

```
ein1 <= (((not e4) and (not e6)) and e5);
ein2 <= (((not e4) and (not e5)) and e6);
ein3 <= (e4 and (e5 and (not e6)));
ein4 <= (((not e4) and e5) and e6);
ein5 <= ((e6 and e5) and e4);
ein6 <= (((not e5) and e4) and e6);

DO (0) <= ein0;
DO (1) <= ein1;
DO (2) <= ein2;
DO (3) <= ein3;
DO (4) <= ein4;
DO (5) <= ein5;
DO (6) <= ein6;

DOU (0) <= ein0 xor DI (0);
DOU (1) <= ein1 xor DI (1);
DOU (2) <= ein2 xor DI (2);
DOU (3) <= ein3 xor DI (3);
DOU (4) <= ein4 xor DI (4);
DOU (5) <= ein5 xor DI (5);
DOU (6) <= ein6 xor DI (6);
```

```
        DOUT (0) <= ein3 xor DI (3);
        DOUT (1) <= ein4 xor DI (4);
        DOUT (2) <= ein5 xor DI (5);
        DOUT (3) <= ein6 xor DI (6);
  end VER1;
```

5.3.1.4. *Source Codes* Hamming Encoder

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.all;
entity  hamenc is
    port(
        datain : in std_ulogic_vector(3 downto 0);
        hamout : out std_ulogic_vector(6 downto 0)
      );
end hamenc;

architecture behavior of hamenc is
  signal p0,p1,p2  : std_ulogic;
begin
  p0 <= (datain (0) xor datain (1)) xor datain (3);
```

```
p1 <= (datain (3) xor datain (2)) xor datain (1);
p2 <= (datain (2) xor datain (1)) xor datain (0);

hamout (6 downto 4) <= ( p0,p1,p2);
hamout (3 downto 0) <= datain(3 downto 0) ;
end behavior;
```

5.3.2. Penerangan *Source codes* bagi Gabungan Hamming encoder dan Hamming Decoder

Bagi membolehkan pemilihan dilakukan antara Hamming encoder dengan Hamming Decoder, pemilihan mod perlu dibuat. Disini, modul mux bertindak sebagai pengawal ke atas pemilihan mod yang dilakukan .Dalam komponen mux, 2 pembolehubah tetap digunakan iaitu:

```
constant c_enc : std_ulogic_vector(1 downto 0) := "01";
constant c_dec : std_ulogic_vector(1 downto 0) := "10";
```

Jika pengguna memilih Hamming encoder sebagai data input maka '01' akan dipilih sebagai *controller,* sebaliknya Hamming decoder yang dipilih maka nilai '10' akan di pilih. Output akhir akan diumpukkan keatas result dimana ia bergantung kepada pemilihan *controller.* Disini, panjang untuk encoder adalah berbeza dengan panjang decoder iaitu encoder mengeluarkan 7 bit output manakala decoder pula mengeluarkan 4 bit ouput. Oleh itu, keputusan akhir yang dikeluarkan untuk Hamming decoder akan

digabungkan dengan nilai "ZZZ" seperti *source codes* di bawah. Nilai ZZZ merujuk kepda nilai ampungan iaitu sebarang nombor boleh diwakilkan. Nilai akhir diumpukkan kedalam result.

```
with operator select
    res <=  result1  when c_enc,
            result2 & "ZZZ" when c_dec,
            "0000000" when others;
    result<=res;
```

Dalam gabungan antara Hamming encoder dengan Hamming decoder, satu toplevel telah dibina dimana ianya mengandungi pengisytiharaan terhadap modul Hamming encoder, hamming decoder dan multiplex yang bertindak sebagai *controller*. Dalam *source codes* top level, semua komponen iaitu component hamenc yang mewakili Hamming encoder, component HAMDEC yang mewakili Hamming Decoder dan component mux yang mewakili *controller* dipanggil semula bagi membolehkan operasi setiap modul dapat dijalankan dalam bahagian top level. Dalam bahagian top level terdapat 2 pengisytiharaan terhadap signal iaitu:

```
signal s_result1 : std_ulogic_vector(6 downto 0);
signal s_result2 : std_ulogic_vector(3 downto 0);
```

Signal ini digunakan untuk menghubungkan ketiga-tiga komponen yang tedapat dalam top level. Setiap port yang terdapat dalam komponen dipetakan melalui port map agar

ianya dapat berfungsi di bahagian top level dan mengeluarkan output gabungan bagi encoder dan decoder.

5.4. Gabungan CRC Encoder dan CRC Decoder

5.4.1. *Source codes* bagi Gabungan CRC Encoder dan CRC Decoder

5.4.1.1. *Source Codes* Top Level CRC

```
library IEEE;
use IEEE.std_logic_1164.all;

entity toplevel is
   port (
        operator        : in std_ulogic_vector(1 downto 0);
        clk,rst,load    : in std_ulogic;
        Din             : in std_ulogic_vector (3 downto 0);
        Datain          : in std_ulogic_vector (6 downto 0);
        syndrome,shift0,shift1,shift2, reg  : out std_ulogic;
        result          : out std_ulogic_vector(6 downto 0)
        );

end toplevel;
```

```
architecture toplevel_arch of toplevel is

    component crc_encod
        port (
            Clk, Rst, load : in std_ulogic;
            Din            : in std_ulogic_vector (3 downto 0);
            CRC_Sum        : out std_ulogic_vector (6 downto 0)
            );
    end component;

    component crc_decode
        port (
            Clk, Rst, load : in std_ulogic;
            Datain         : in std_ulogic_vector (6 downto 0);
            syndrome       : out std_ulogic;
            shift0         : out std_ulogic;
            shift1         : out std_ulogic;
            shift2         : out std_ulogic;
            reg            : out std_ulogic;
            XCRC           : out std_ulogic
            );
        end component;

    component mux
```

```
        port (
                operator        : in  std_ulogic_vector(1 downto 0);
                CRC_en          : in std_ulogic_vector (6 downto 0);
                CRC_de          : in std_ulogic;
                result          : out std_ulogic_vector(6 downto 0)
            );
    end component;

    signal s_CRC_SUM  :  std_ulogic_vector(6 downto 0);
    signal s_OutCRC   :  std_ulogic;

begin

    U1: crc_encod  port map (  Clk=> clk,
                               Rst=>Rst,
                               load => load,
                               Din => din,
                               CRC_SUM=>s_CRC_SUM);

    U2: crc_decode port map (  Clk => clk,
                               Rst => Rst,
                               load => load,
                               Datain => Datain,
                               syndrome =>syndrome,
```

```
                                shift0=>shift0,
                                shift1 =>shift1,
                                shift2=>shift2,
                                reg =>reg,
                                XCRC=> s_OutCRC);

    U3: mux port map    (       operator=> operator,
                                CRC_en =>s_CRC_SUM,
                                CRC_de =>s_OutCRC,
                                result=> result);

end toplevel_arch;
```

5.4.1.2. *Source Codes* Mux (*controller)* CRC

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;

entity mux is
    port (
          operator      : in  std_ulogic_vector(1 downto 0);
          CRC_en        : in std_ulogic_vector (6 downto 0);
          CRC_de        : in std_ulogic;
```

```
        result          : out std_ulogic_vector(6 downto 0)
    );

end mux;

architecture host_arch of mux is
    signal CRCdecod    : std_ulogic_vector(6 downto 0);
    signal CRCencod    : std_ulogic_vector(6 downto 0);
    signal last        : std_ulogic_vector(6 downto 0);
    constant c_enc : std_ulogic_vector(1 downto 0) := "01";
    constant c_dec : std_ulogic_vector(1 downto 0) := "10";

begin
 CRCencod <= CRC_en;
 CRCdecod <= CRC_de & "000000";

 with operator select
  last <=   CRCencod when c_enc,
            CRCdecod when c_dec,
            "0000000" when others;
  result <= last;

end host_arch;
```

5.4.1.3. *Source Codes* CRC Encode

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;

entity crc_encod is
  port (
        Clk, Rst, load    : in std_ulogic;
        Din               : in std_ulogic_vector (3 downto 0);
        CRC_Sum           : out std_ulogic_vector (6 downto 0)
        );
end crc_encod;

architecture behavior of crc_encod  is
   signal X               : std_ulogic_vector(6 downto 0);
   signal s_d             : std_ulogic_vector(3 downto 0);

begin
    s_d <= Din;

   process (Clk,Rst)
   begin
```

```
    if Rst = '1' then
      X <= (others=> '0'); -- initalize values
      CRC_Sum <="0000000";
    elsif rising_edge(Clk) then

      if (load= '1') then
          s_d<= '0'& s_d (3 downto 1);
          X (6 downto 3) <= (others=>'0');
          X (2) <= s_d (0) xor X (0);
          X (1) <= s_d (0) xor X (2) xor X (0);
          X (0) <= X (1);

       end if;
    end if;
  end process;

CRC_Sum<= Din & X (2 downto 0);

end behavior;
```

5.4.1.4. *Source Codes* CRC Decoder

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity crc_decode is
    port (
            Clk, Rst, load   : in std_ulogic;
            Datain           : in std_ulogic_vector (6 downto 0);
            syndrome         : out std_ulogic;
            shift2           : out std_ulogic;
            shift1           : out std_ulogic;
            shift0           : out std_ulogic;
            reg              : out std_ulogic;
            XCRC             : out std_ulogic

        );

    end crc_decode;

    architecture behavior of crc_decode is

    signal X      : std_ulogic_vector(6 downto 0);
    signal s_d    : std_ulogic_vector(6 downto 0);
    signal s_out  : std_ulogic;
    signal syn, s_syn1, s_reg : std_ulogic;

    begin
```

```
 s_d<=Datain;
 process (Clk,Rst)
begin

  if Rst = '1' then
        X <= (others=> '0');
        reg  <= '0';
        XCRC <= '0';
  elsif rising_edge(Clk) then

     if (load = '1') then
        s_d<= '0' & s_d(6 downto 1);
        X (6 downto 3) <= (others=>'0');
        X (2) <= X (0) xor s_d (0);
        X (1) <= X (2) xor X (0);
        X (0) <= X (1);
        reg <= s_d(0);
        s_out<=  syn xor s_d(0);
        s_reg <= s_d(0);

     end if;
  end if;

end process;
```

```
        syn <= (X(2) and (not X(1))) and X(0);
        s_syn1 <= (X (2) and (not X (1))) and X (0);
        syndrome <= syn;
      XCRC <= s_syn1 xor s_reg;
        shift2 <= X (2);
        shift1 <= X (1);
        shift0 <= X (0);

end behavior;
```

5.4.2. Penerangan *Source codes* bagi Gabungan CRC Encoder dan CRC Decoder

Gabungan antara CRC encoder dan CRC decoder juga menggunakan teknik yang sama seperti gabungan untuk Hamming. Disini controller juga digunakan untuk membuat pemilihan samada pengguna ingin menggunakan teknik CRC encoder atau teknik CRC decoder. Dua pembolehubah yang bertindak sebagai *controller* digunakan iaitu:

```
constant c_enc : std_ulogic_vector(1 downto 0) := "01";
constant c_dec : std_ulogic_vector(1 downto 0) := "10";
```

Disini, pembolehubah di atas berfungsi mengawal pemilihan mod keatas CRC. Constant c_enc: std_ulogic_vector (1 downto 0):= "01" merupakan pemboleh ubah yang

mengawal pemilihan mod keatas CRC encoder manakala constant c_dec: std_ulogic_vector (1 downto 0):= "10" adalah untuk pemilihan mod keatas CRC decoder. Di akhir keluaran, 7 bit ouput akan dikeluarkan apabila pemilihan mod dibuat.

Bagi top level CRC pula, 3 komponen akan di panggil dari 3 modul iaitu component crc_encod, component crc_decod dan component mux. Ketiga-tiga komponen ini akan memanggil kembali komponen-komponen yang dipelukan apabila pengujian dibuat berdasarkan kepada proses yang berlaku dalam setiap modul. Port map digunakan untuk memetakan setiap port yang terdapat dalam komponen.

BAB 6
PENGUJIAN SISTEM

Bab 6: Pengujian Sistem

Bab 6 ini akan memaparkan hasil-hasil dari pengujian yang telah dilakukan keatas sistem ini. Sebelum melangkah ke bahagian pengujian sistem bagi top level akan diterangkan terlebih dahulu output bagi hamming decoder dan CRC decoder.

6.1. Pengujian sistem bagi Hamming Decoder

Test bench digunakan untuk memasukkan nilai output kedalam atucara bagi memastikan samada sistem berfungsi seperti spesifikasi yang dikehendaki ataupun tidak. Berikut adalah test bench bagi Hamming decoder beserta hasil pengujiannya.

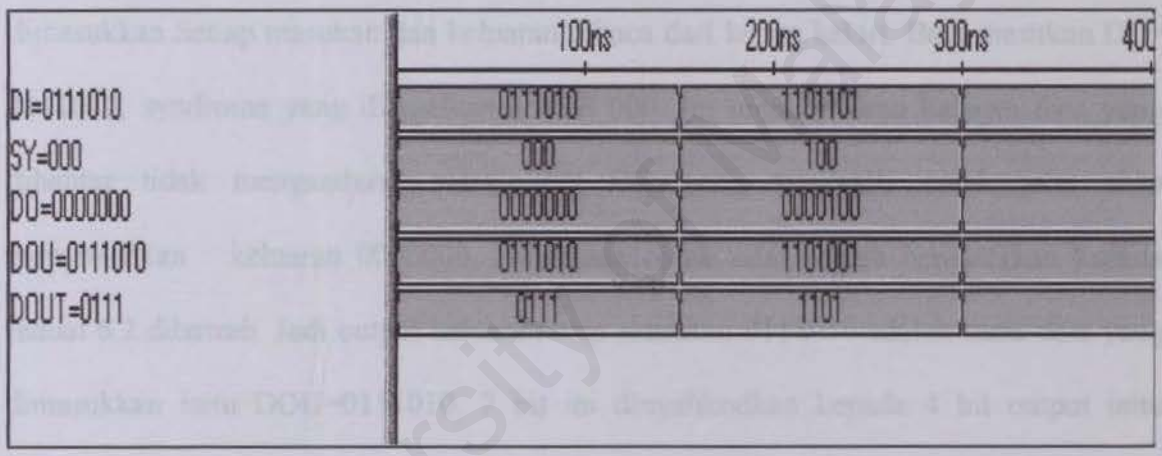
6.1.1. Test Bench bagi Hamming Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity testbench is
end testbench;
use work.all;

architecture stimulus of testbench is

component HAMDEC
```

```
port (
        DI          : in std_ulogic_vector(6 downto 0);
        SY          : out std_logic_vector (2 downto 0);
        DO          : out std_ulogic_vector(6 downto 0);
        DOU         : out std_logic_vector (6 downto 0);
        DOUT        : out std_logic_vector (3 downto 0)
        );
end component;

signal DI       : std_ulogic_vector(6 downto 0);
signal SY       : std_logic_vector(2 downto 0);
signal DO       : std_ulogic_vector(6 downto 0);
signal DOU      : std_logic_vector(6 downto 0);
signal DOUT     : std_logic_vector (3 downto 0);

begin

DUT : HAMDEC port map (DI,SY,DO,DOU,DOUT);
 stimulus1: process
 begin

    DI <= "0111010";   --tiada ralat
    wait for 150 ns;
```

```
        DI <= "1101101";  --ralat
        wait for 150 ns;
        wait;

    end process;
end stimulus;
```

6.1.2. Hasil Pengujian Hamming Decoder



Rajah 6.1: Output bagi simulasi Hamming Decoder

Bagi tujuan menguji samaada sistem berfungsi dengan betul atau tidak, test bench dilakukan dengan memasukkan input-input ke dalam *source codes* test bench. Komponen HAMDEC dipanggil semula didalam test bench bagi membolehkan bit-bit yang dimasukkan dilaksanakan mengikut source codes yang telah dijanakan. Disini 2 contoh input dimasukkan iaitu 0101110 dan 1011011 sebagai DI seperti dibawah:

```
DI <= "0111010";  --tiada ralat
wait for 150 ns;

DI <= "1101101";  --ralat
wait for 150 ns;
```

Setiap masukan akan mengalami proses sehingga 150 ns sebelum masukan input seterusnya. Dalam rajah 6.1 diatas dipaparkan hasil keluaran bagi setiap input yang dimasukkan.Setiap masukan dan keluaran dibaca dari kanan kekiri. Bagi masukan DI = 0111010, syndrome yang dihasilkan adalah 000. Ini menunjukkan bahawa data yang dihantar tidak mengandungi ralat. Jadi DO yang mewakili corak ralat akan menghasilkan keluaran 0000000. Penentuan corak ralat adalah berdasarkan kepada jadual 6.2 dibawah. Jadi output keluaran bagi masukan 0111010 adalah sama data yang dimasukkan iaitu DOU=0111010. 7 bit ini dinyahkodkan kepada 4 bit output iaitu DOUT= 0111. Bagi masukan yang kedua DI = 1101101 pula, ralat telah di kesan berlaku keatas data apabila sindrom yang terhasil adalah SY=100. Jika dibandingkan dengan jadual corak ralat didapati, ralat telah berlaku pada kedudukan kedua dimana bit yang dihasilkan adalah 0000100. Data yang ralat akan di betulkan dengan melakukan operasi XOR keatas data masukan, DI dengan corak ralat DO. Data yang telah dibetulkan dikeluarkan sebagai DOU=1101001 dan kemudiannya dikodkan kepada 4 bit untuk menghasilkan 1101. Sebarang masukan dan keluaran dapat dibandingkan melalui jadual Kebenaran bagi Hamming seperti yang terdapat dalam jadual 6.1.

Jadual 6.1: Jadual Kebenaran bagi Hamming

Mesej	"Codewords"
0 0 0 0	0 0 0 0 0 0 0
1 0 0 0	1 1 0 1 0 0 0
0 1 0 0	0 1 1 0 1 0 0
1 1 0 0	1 0 1 1 1 0 0
0 0 1 0	1 1 1 0 0 1 0
1 0 1 0	0 0 1 1 0 1 0
1 1 1 0	0 1 0 1 1 1 0
0 0 0 1	1 0 1 0 0 0 1
1 0 0 1	0 1 1 1 0 0 1
0 1 0 1	1 1 0 0 1 0 1
1 1 0 1	0 0 0 1 1 0 1
0 0 1 1	0 1 0 0 0 1 1
1 0 1 1	1 0 0 1 0 1 1
0 1 1 1	0 0 1 0 1 1 1
1 1 1 1	1 1 1 1 1 1 1

Jadual 6.2: Jadual Corak Ralat

SINDROM	CORAK RALAT
000	0000000
100	1000000
010	0100000
001	0010000
100	0001000
011	0000100
111	0000010
101	0000001

6.2. Pengujian sistem bagi CRC decoder

6.2.1. Test Bench bagi CRC decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity CRC is
end CRC;
use work.all;

Architecture behavior of hamming is
        component crc_decode
```

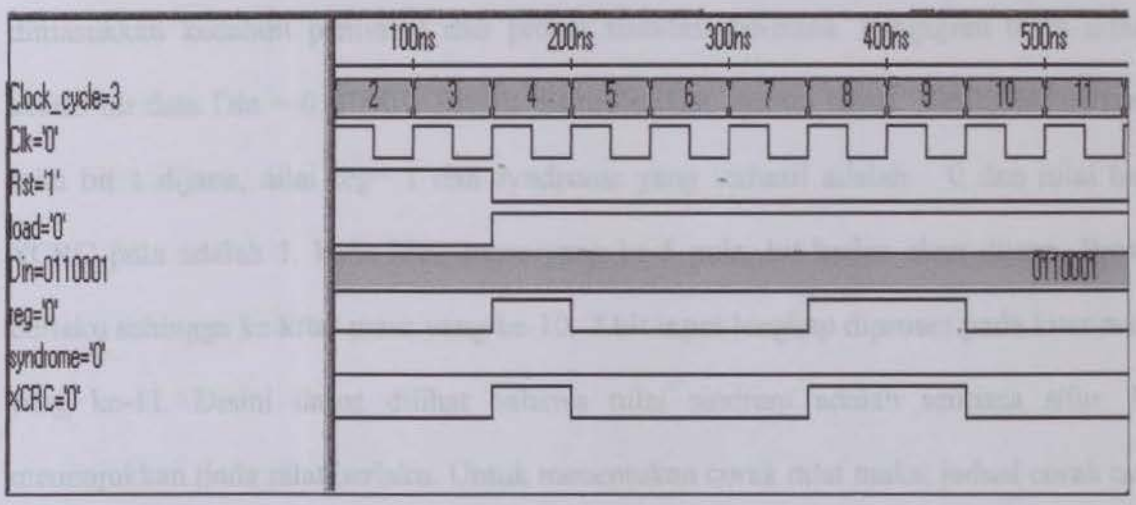
```
        port (
                Clk,Rst,load    : in std_ulogic;
                Din             : in std_ulogic_vector (6 downto 0);
                syndrome        : out std_ulogic;
                shift2          : out std_ulogic;
                shift1          : out std_ulogic;
                shift0          : out std_ulogic;
                reg             : out std_ulogic;
                XCRC            : out std_ulogic
                );

end component;

signal Clk,Rst,load             : std_ulogic;
signal Din                      : std_ulogic_vector(6 downto 0);
signal syndrome                 : std_ulogic;
signal shift2,shift1,shift0     : std_ulogic;
signal reg                      : std_ulogic;
signal XCRC                     : std_ulogic;
signal Clock_cycle              : natural := 0;

begin

DUT: crc_decode port map (Clk, Rst, load, Din, syndrome, shift2,
```

```
                                    shift1, shift0, reg, XCRC);

    CLOCK: process

    Begin
        Clock_cycle <= Clock_cycle + 1;
        Clk <= '1';
        wait for 25 ns;
        Clk <= '0';
        wait for 25 ns;
    end process;

Stimulus1: process
    begin
    -- tiada ralat
    load <= '0';
    Rst <= '1';
    Din <= "0110001";
    wait for 150 ns;
    load <= '1';
    Rst  <= '0';
    Din <= "0110001";
    wait for 150 ns;
```

```
        -- ralat
        load <= '0';
        Rst  <= '1';
        Din <= "0111001";
        wait for 150 ns;
        load <= '1';
        Rst <= '0';
        Din <= "0111001";
        wait for 150 ns;


        wait;

    end Process;

end behavior;
```

6.2.2. Hasil Pengujian CRC Decoder

Untuk menguji perlaksanaan CRC decoder ini, jujukan bit akan dimasukkan dengan mengumpukkannya kepada DIN, 2 pengujian telah di buat iaitu untuk masukan bit ketika load=0 dan load =1 dan juga ketika rst=0 dan rst=1.



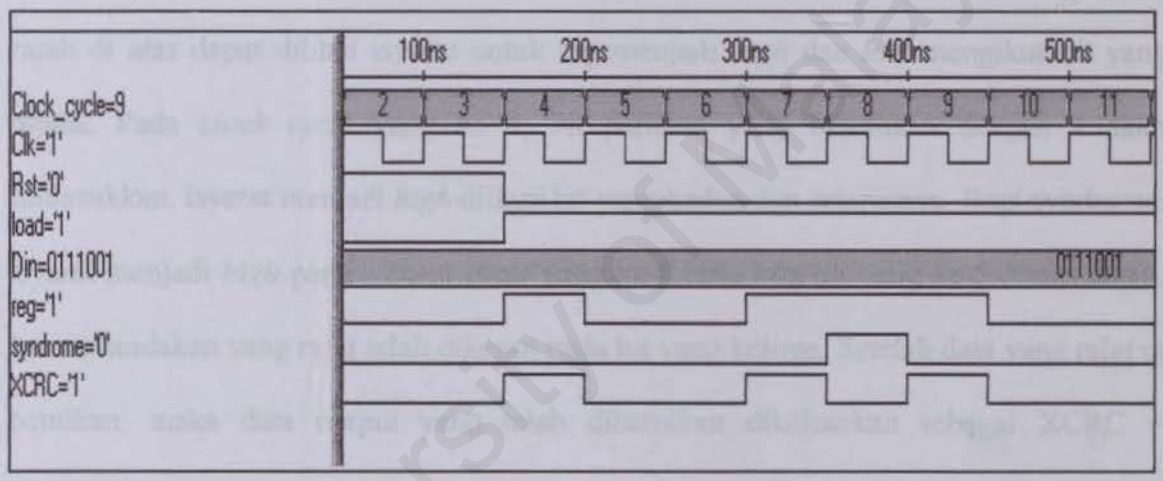
Rajah 6.2: Output bagi simulasi CRC decoder tanpa ralat

Dalam rajah 6.2 ditunjukkan output bagi simulasi CRC decoder tanpa ralat. Berdasarkan kepada test bench, input yang dimasukkan adalah seperti dibawah:

```
load <= '0';
Rst <= '1';
Din <= "0110001";
wait for 150 ns;
load <= '1';
Rst  <= '0';
Din <= "0110001";
wait for 150 ns;
```

Semasa load=0 dan Rst=1, tiada sebarang simulasi dibuat sehinggalah kitar masa ke-4 iaitu selepas 150ns. Selepas kitar masa ke-4, load =1, Rst=0 dan clk=1, 7 bit input akan

dimasukkan kedalam penimbal dan proses simulasi bermula. Pengujian telah dibuat keatas bit data Din = 0110001. Bit-bit disimulasikan secara selari. Ketika bit pertama iaitu bit 1 dijana, nilai reg= 1 dan syndrome yang terhasil adalah 0 dan nilai bagi XCRC pula adalah 1. Pada kitar masa yang ke-5 pula, bit kedua akan dijana. Proses berlaku sehingga ke kitar masa yang ke-10. 7 bit input lengkap diproses pada kitar masa yang ke-11. Disini dapat dilihat bahawa nilai sindrom adalah sentiasa sifar. Ini menunjukkan tiada ralat berlaku. Untuk menentukan corak ralat maka, jadual corak ralat dirujuk seperti yang terdapat pada jadual 6.2.



Rajah 6.3: Output bagi simulasi CRC decoder mengandungi ralat

Dalam rajah 6.3 pula ditunjukkan output bagi simulasi CRC decoder yang mengandungi ralat. Berdasarkan kepada test bench, input yang di masukkan adalah seperti dibawah:

```
load <= '0';
Rst <= '1';
```

```
Din <= "0111001";
wait for 150 ns;
load <= '1';
Rst <= '0';
Din <= "0111001";
wait for 150 ns;
```

Ketika load = '0', Rst = '1', tiada output yang dikeluarkan. Bila load=1, clk=1 dan Rst=0 maka, 7 bit codeword Din = "0111001" akan memasuki penimbal. Dalam rajah di atas dapat dilihat isyarat untuk reg menjadi *high* dan *low* mengikut bit yang dijana. Pada *clock cycle* yang ke 4, bit pertama yang besamaan dengan 1 akan dimasukkan. Isyarat menjadi *high* diikuti bit yang kedua dan seterusnya. Bagi syndrome, isyarat menjadi *high* pada *clock cycle* yang ke-8 iaitu bila bit yang ke-5 dimasukkan. Ini menandakan yang ralat telah dikesan pada bit yang kelima. Setelah data yang ralat di betulkan, maka data output yang telah dibetulkan dikeluarkan sebagai XCRC = "0101001.

6.3. Pengujian sistem gabungan antara Hamming Decoder dan Hamming encoder

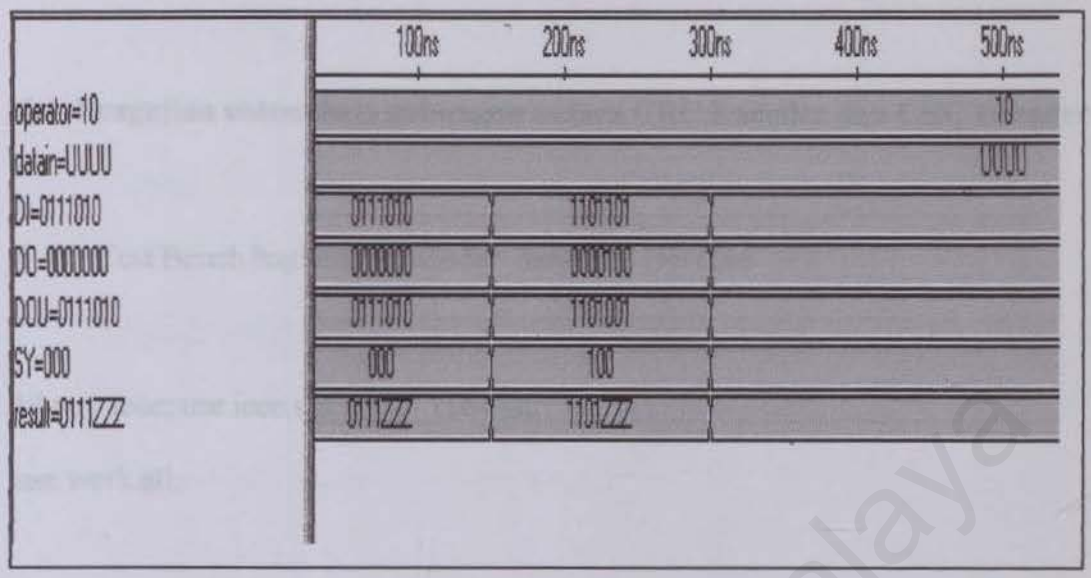
6.3.1. Test Bench bagi Hamming Decoder dan Hamming encoder

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity testbench is
end testbench;
use work.all;

architecture stimulus1 of testbench is

    component toplevel
    port (
                operator     : in std_ulogic_vector(1 downto 0);
                datain   : in  std_ulogic_vector(3 downto 0);

                DI       : in std_ulogic_vector (6 downto 0);
                SY       : out std_ulogic_vector (2 downto 0);
                DO       : out std_ulogic_vector (6 downto 0);
                DOU      : out std_ulogic_vector (6 downto 0);
                result   : out std_ulogic_vector(6 downto 0)
            );
    end component;

        constant PERIOD : time := 150 ns;
        signal operator : std_ulogic_vector(1 downto 0);
        signal datain   : std_ulogic_vector(3 downto 0);
        signal DI       : std_ulogic_vector(6 downto 0);
        signal SY       : std_ulogic_vector(2 downto 0);
```

```
        signal DO     : std_ulogic_vector(6 downto 0);
        signal DOU    : std_ulogic_vector(6 downto 0);
        signal result : std_ulogic_vector(6 downto 0);

begin
 DUT: toplevel port map (operator, datain, DI, SY, DO,DOU, result);

   input: process
  begin
    -- input decoder
              operator<= "10";  -- tiada ralat
              DI<= "0111010";
              wait for PERIOD;

              operator<= "10"; -- ralat
              DI<= "1101101";
              wait for PERIOD;
     wait;
   end process;
end stimulus1;
```

6.3.2.. Hasil Pengujian Hamming Decoder dan Hamming encoder



Rajah 6.4: Output bagi simulasi Hamming Decoder dan Hamming encoder

Pergabungan antara Hamming encoder dengan Hamming Decoder membolehkan pengguna cip memilih samada ingin memasukkan input encoder atau decoder. Pemilihan modul adalah bergantung kepada pemilihan operator yang bertindak sebagai controller. Disini, 2 operator telah ditetapkan dimana operator 01 adalah diperuntukkan untuk Hamming encoder manakala operator 10 pula adalah diperuntukkan untuk Hamming Decoder. Dalam pengujian diatas, operator bagi hamming decoder di pilih. Pada baris pertama dapat dilihat bahawa operator=10 maka nilai input dan output yang terpapar adalah hasil dari test bench untuk decoder. Terdapat 2 pengujian dikenakan keatas sistem iaitu untuk input yang tidak mengandungi ralat dan input yang kedua adalah input yang menghasilkan ralat. Keluaran output adalah sama seperti yang diterangkan sebelum ini. Tetapi yang membezakannya cuma keluran output terakhir dimana 4 bit output

digabungkan dengan nilai "ZZZ". Nilai ZZZ ini merupakan nilai ampungan iaitu sebarang nilai boleh mewakilinya.

6.4. Pengujian sistem bagi gabungan antara CRC Encoder dan CRC Decoder

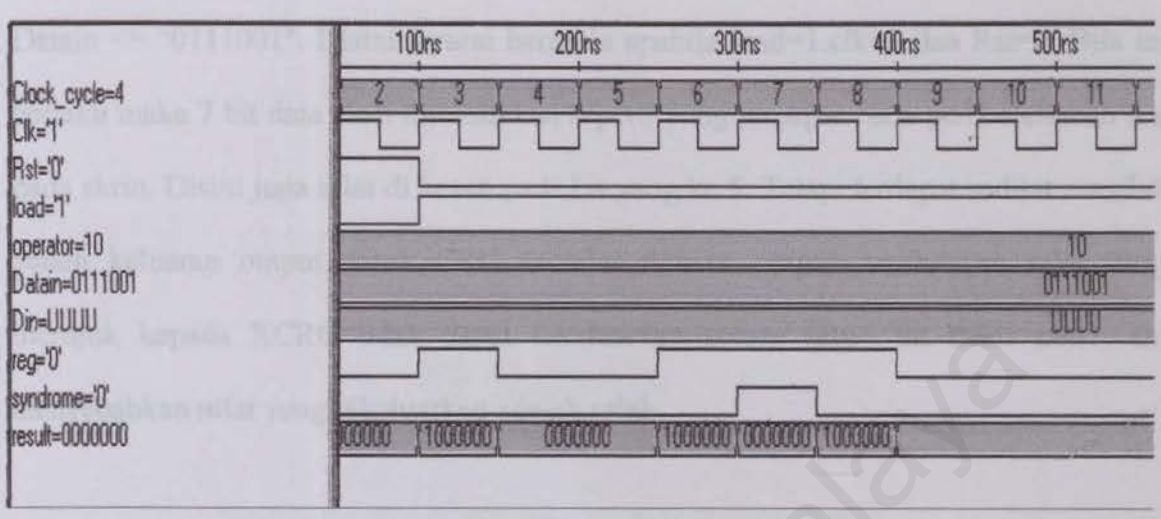
6.4.1. Test Bench bagi CRC Encoder dan CRC Decoder

```
library ieee; use ieee.std_logic_1164.all;
use work.all;

entity testbench is
end testbench;

architecture stimulus of testbench is
    component toplevel
        port (
            operator      : in std_ulogic_vector(1 downto 0);
            clk,rst,load: in std_ulogic;
            Din           : in std_ulogic_vector (3 downto 0);
            Datain        : in std_ulogic_vector (6 downto 0);
             syndrome,shift0,shift1,shift2,reg  : out std_ulogic;
            result        : out std_ulogic_vector(6 downto 0)
            );
    end component;
```

```
    constant PERIOD         : time := 100 ns;
    signal operator         : std_ulogic_vector(1 downto 0);
    signal clk,rst,load     : std_ulogic;
    signal Din              : std_ulogic_vector(3 downto 0);
    signal Datain           : std_ulogic_vector(6 downto 0);
    signal syndrome,shift0,shift1,shift2,reg :std_ulogic;
    signal result           : std_ulogic_vector(6 downto 0);
    signal Clock_cycle      : natural := 0;

begin

  DUT: toplevel portmap (operator, clk, rst, load, Din,
                         syndrome, shift0, shift1, shift2, reg, result);

   CLOCK: process
  begin
    Clock_cycle <= Clock_cycle + 1;
    Clk <= '1';
    wait for 25 ns;
    Clk <= '0';
    wait for 25 ns;
  end process;
```

```
INPUTS: process

begin

    -- input CRC decoder
            rst<='1';
            load<='0';
            operator <= "10";
            Datain <= "0111001";
            wait for PERIOD;
            rst<='0';
            load<='1';
            operator <= "10";
            Datain <= "0111001";
            wait for PERIOD;
    wait;
  end process;
end stimulus;
```

6.4.2.. Hasil Pengujian CRC Encoder dan CRC Decoder



Rajah 6.5: Output bagi simulasi CRC encoder dan CRC Decoder

Dalam test bench diatas, input yang masukkan adalah

```
rst<='1';

load<='0';

operator <= "10";

Datain <= "0111001";

wait for PERIOD;

rst<='0';

load<='1';

operator <= "10";

Datain <= "0111001";

wait for PERIOD;
```

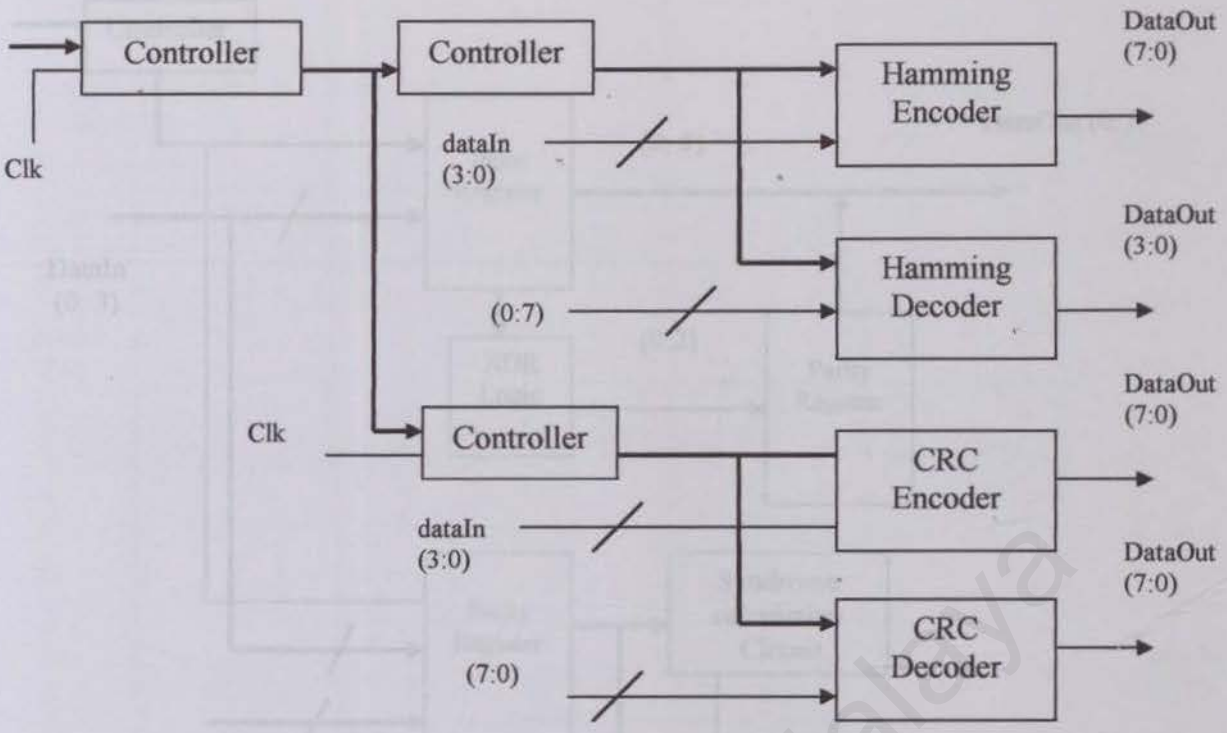
Seperti yang dilihat, operator yang dipilih adalah operator=10. nilai 10 merujuk kepada mod pemilihan untuk CRC decoder. Maka input yang dimasukkan adalah 7 bit iaitu Datain <= "0111001". Disini isyarat bermula apabila load=1.clk=1 dan Rst=0. Bila ini berlaku maka 7 bit data akan dimasukkan seperti yang terpapar pada pembolehubah reg pada skrin. Disini juga ralat di kesan pada bit yang ke 5. Tetapi terdapat sedikit masalah dalam keluaran output untuk CRC decoder dimana output pembetulan ralat yang merujuk kepada XCRC tidak dapat dikeluarkan secara satu bit demi satu . Ini menyebabkan nilai yang dikeluarkan adalah salah.

BAB 7
PERBINCANGAN

Bab 7: Perbincangan

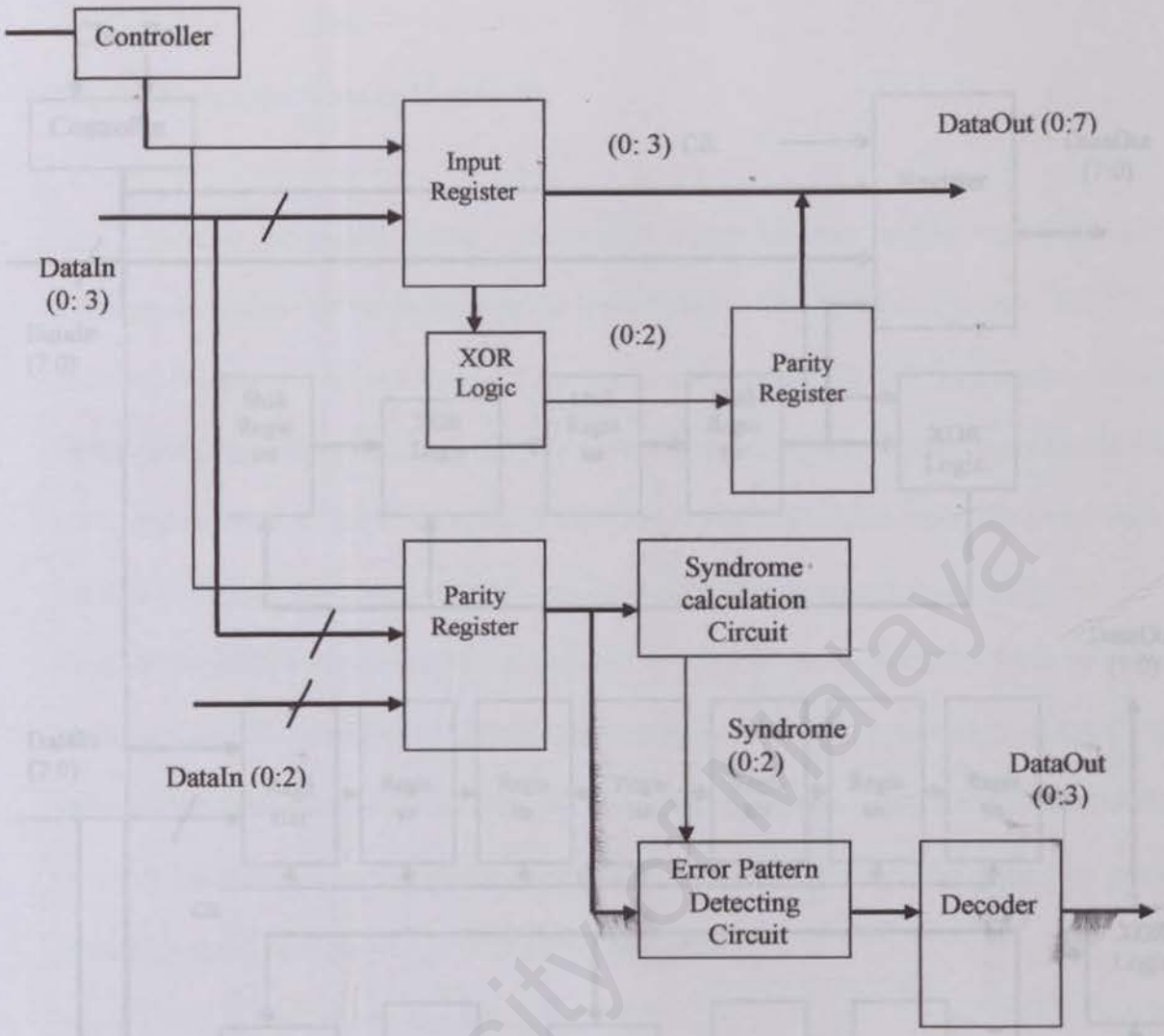
Bab 7 akan membincangkan tentang sebarang keputusan yng diperolihi, hasil dari pengujian. Selain itu, akan dimuatkan juga masalah yang dihadapi dan penyelesaian yang diambil, kelebihan dan kelemahan sistem yang dibangunkan dan juga cadangan serta kesimpulan bagi projek yang dijalankan.

Dalam melaksanakaan projek ini, terdapat beberapa perubahan di buat keatas rekabentuk sistem dimana rekabentuk sistem yang sediaada telah diubah suai dengan menukar top level asal kepada top level seperti dalam rajah 6.1. Dalam rajah ini, terdapat 2 modul iaitu modul Hamming dan modul untuk CRC. *Controller* bertidak sebagai pengawal untuk mengawal pemilihan mana-mana antara 2 modul ini. Jika mod yang dipilih adalah modul Hamming maka pengguna sekali memilih samada ingin menggunakan encoder atau decoder. Bagi encoder 4 bit data input diperlukan dan kemudiannya mengeluarkan 7 bit output yang telah dinyahkodkan. Tetapi jika pengguna memilih decoder, 7 bit akan di masukkan ke dalam decoder dan 4 bit output akan dikeluarkan di bahagian ini. Cara yang sama juga berlaku paada bahagian CRC dimana, 4 bit input dimasukkan pada bahagian encoder dan 7 bit input untuk bahagian decoder. Perbezaan antara 2 modul ini ialah pada bahagian decoder Hamming, 4 bit output dikeluarkan manakala pada bahagian decoder CRC pula, 7 bit input dikeluarkan.

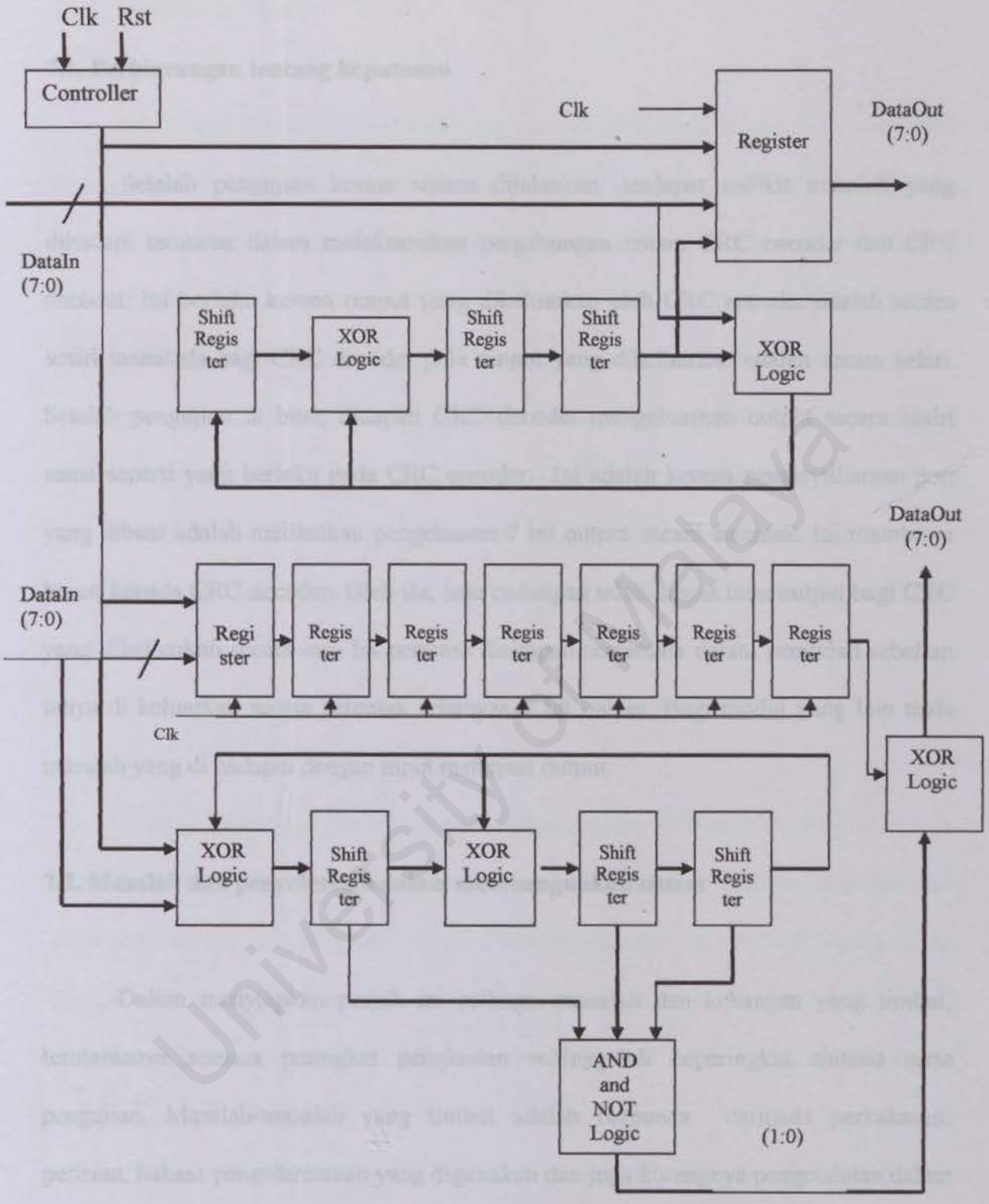


Rajah 7.1: Top Level Diagram

Rajah 7.2 dan rajah 7.3 pula menunjukkan blok diagram gabungan bagi Hamming dan CRC. Bagi blok diagram untuk Hamming, gabungan telah dibuat antara Hamming encoder dengan Hamming Decoder dan untuk CRC pula gabungan telah dilakukan keatas CRC encoder dan CRC decoder. Aliran perjalanan bit adalah seperti yang di terangkan dalam bab sebelum ini.



Rajah 7.2: Blok Diagram Hamming Codes (7, 4)



Rajah 7.3: Blok Diagram Cyclic Redundancy Check Codes

7.1. Perbincangan tentang keputusan

Setelah pengujian keatas sistem dijalankan ,terdapat sedikit masalah yang dihadapi terutama dalam melaksanakan pergabungan antara CRC encoder dan CRC decoder. Ini berlaku kerana output yang dikeluarkan oleh CRC encoder adalah secara sesiri manakala bagi CRC decoder pula output yang dikeluarkan adalah secara selari. Setelah pengujian di buat, didapati CRC decoder mengeluarkan output secara sesiri sama seperti yang berlaku pada CRC encoder. Ini adalah kerana pengisytiharaan port yang dibuat adalah melibatkan pengeluaran 7 bit output secara serentak. Ini membawa kesan kepada CRC decoder. Oleh itu, satu cadangan telah dibuat iaitu output bagi CRC yang dikeluarkan secara satu bit permasa disimpan sementara dalam penimbal sebelum ianya di keluarkan secara serentak sebanyak 7 bit output. Bagi modul yang lain tiada masalah yang di hadapai dengan input mahupun output.

7.2. Masalah dan penyelesaian dalam membangunkan sistem

Dalam menyiapkan projek ini pelbagai masalah dan kekangan yang timbul, terutamanya semasa peringkat pengkodan sehinggalah keperingkat sintesis serta pengujian. Masalah-masalah yang timbul adalah berpunca daripada perkakasan, perisian, bahasa pengaturcaraan yang digunakan dan juga kurangnya pengetahuan dalam bahasa pengturcaraan yang digunakan ketika membangunkan projek ini.

7.2.1. Masalah peralatan

Dari segi peralatan, masalah yang timbul adalah dari segi komputer yang digunakan dimana kelajuan pemproses agak lambat. Ini menyebabkan banyak masa perlu diperuntukkan setiap kali menggunakan perisian peak FPGA. Masalah menjadi lebih ketara apabila berlakunya kerosakan hard disk yang menyebabkan kerja tertangguh. Ini menyebabkan hard disk baru terpaksa dibeli. Pemasangan semula hard disk juga mengambil masa kerana ia melibatkan mendapatkan semula hard disk baru dan *install* semula perisian Peak FPGA yang digunakan. Selain daripada kerosakan hard disk, drive A untuk PC yang digunakan juga mengalami kerosakan menyebabkan fail-fail penting seperti source code dan lain-lain yang terdapat dalam disket tidak dapat dibuka dan dibaca.

Penyelesaian

Dalam menyelesaikan masalah ini, beberapa pendekatan telah diambil dimana dalam masalah hard disk, hard disk yang rosak telah ditukar kepada yang baru dan perisian PeakFGA telah di install semula. Perkara yang sama juga telah dilakukan keatas drive A agar capaian keatasnya boleh dibuat. Selain itu, satu alternatif lain telah diambil iuitu menggunakan komputer dalam makmal VLSI di FSKTM untuk melaksanakan projek yang agak tertangguh.

7.2.2. Masalah Perisian

Pada peringkat permulaan, cadangan perisian yang ingin digunakan dalam membangunkan projek adalah webpack tetapi perisian ini tidak dapat menyokong atau mensintesiskan kod yang dijanakan.

Penyelesaian

Untuk mengatasi masalah ini, satu perisian baru telah digunakan iaitu Peak FPGA yang mana perisian ini dapat menyokong keperluan projek.

7.2.3. Masalah Bahasa Pengaturcaraan

Pengatahuan yang kurang dalam penggunaan bahasa pengaturcaraan VHDL menjadi satu masalah dimana masa yang ada untuk mempelajari bahasa pengatrcaraan ini adalah terhad kerana sibuk dengan tugas sebagai belajar. VHDL juga merupakan perisian baru dalam diri pembangun, jadi pembangun memerlukan masa untuk mendalami bahasa pengaturcaraan ini.

Penyelesaian

Satu jadual telah dibuat untuk membahagikan masa antara belajar dan juga masa membangunkan projek. Ini kerana selain sibuk dengan projek, pembangun juga terpaksa melakukan penyelidikan keatas kursus-kursus pengajian yang lain dimana kebanyakan kursus-kursus ini juga adalah dalam bentuk projek. Ulangkaji telah dibuat melalui buku,

sumber dari internet dan juga bertanya kepada penyelia dan kawan-kawan tentang VHDL

7.2.4. Masalah Sumber rujukan

Sumber rujukan terutamanya buku-buku yang berkaitan dengan Hamming dan CRC adalah kurang, begitu juga dengan contoh-contoh *source codes* yang boleh dijadikan panduan untuk membangunkan projek adalah terhad. Ini terjadi kerana berlakunya sedikit masalah dalam mendapatkan sumber di perpustakaan utama kerana pembaikpulihan perpustakaan telah dijalankan, maka kebanyakan buku-buku yang dikehendaki dihalang daripda meminjamnya. Selain itu juga, maklumat yang di perolihi daripada rujukan keatas buku dan beberapa sumber lain tidak memenuhi kriteria-kriteria yang di kehendaki.

Penyesaian

Mencari alternatif lain iaitu dengan cara mendapatkan sumber daripada internet serta mendapatkan padangan dan tunjuk ajar daripada penyelia projek dan juga kawan-kawan.

7.3. Kelebihan dan kelemahan sistem

7.3.1. Kelebihan sistem

Kelebihan sistem ini ialah rekabentuknya tidak kompleks kerana ia melibatkan penggunaan get XOR, DAN dan NOT sahaja. Selain itu, pengguna juga boleh membuat pilihan samada untuk menggunakan encoder atau decoder. Jadi ini menjimatkan masa. Selain itu penggunaan get XOR dalam Hamming dapat mengurangkan kos dalam pembinaan cip.

7.3.2. Kelemahan sistem

Sistem ini hanya dapat digunakan untuk masukan 7 bit input sahaja pada satu-satu masa. Ini agak ketinggalan dengan teknologi yang ada pada masa kini yang mana kebanyakannya menggunakan 32 bit sebagai masukan input. Sistem ini juga, banyak menghasilkan *delay k*erana tiada penggunaan multipler dalam menghasilakan atau pengiraan simdrom.

7.4. Cadangan dan kesimpulan

'Pengesanan dan Pembetulan ralat menggunakan teknik Hamming Codes dan Cyclic Redundency' dibangunkan menggunakan bahasa pengaturcaraan VHDL dengan melaksanakannya kedalam Peak FPGA. Disini, perlaksanaan keatas sistem ini hanya melibatkan proses compile, link dan simulate. Jadi dicadangkan agar sistem ini dapat

dilaksanakan sehingga keperingkat sintesis dan seterusnya mengeluarkannya kedalam bentuk cip iaitu dalam bentuk perkakasan untuk diuji dan digunakan.

Sindrom boleh dikira dalam pelbagai cara. Dalam projek ini, dianggap bahawa sindrom yang dihasilkan adalah dalam bentuk S1, S2, dan S3 dimana pengiraan sindrom ini dapat menyelesaikan masalah berkenaan dengan ralat. Dalam tesis ini, anggapan yang dibuat ialah jika 2 atau lebih sindrom dalam bentuk S_i, S_{2i}, S_{4i}... dikira, maka penyelesaian yang paling bagus adalah menggunakan pembahagian litar. Syndrome S_i boleh dikira dengan menggunakan constant multiplier dan kemudiannya dalam bentuk persamaan. Kesimpulannya, kaedah pengiraan sindrom berdasarkan kepada constant multiplier boleh dipertimbangkan.

Rekabentuk sistem menggunakan VHDL adalah agak susah kerana semasa pembangunan rekabentuk dijalankan, pelbagai anggapan telah dibuat. Anggapan ini boleh membawa kekangan terhadap format input/output Hamming dan CRC, kelajuan masa dan lain-lain. Maka anggapan telah dibuat iaitu semasa data di hantar ke encoder ia juga boleh di terima oleh decoder. Ini membuatkan rekabentuk menjadi lebih kompleks.

Dalam praktikal sistem, jumlah ralat tidak semestinya kurang daripada $\leq t$ ralat dan terdapat juga corak ralat yang tidak dapat dibetulkan. Sistem ini ini tidak boleh mengjangka dengan tepat sebarang ralat yang akan berlaku. Oleh itu sistem ini sepatutnya dibaiki supaya decoder dapat mengesan corak ralat yang tidak dapat dibetulkan'dan sebarang isyarat yang masuk kedalam sistem. Dengan cara ini mesej yang ralat dapat dihantar semula dengan menggunakan kaedah automatic-repeat-request

(ARQ) atau dibaca semula daripada memori. Selain itu sistem ini mestilah berkebolehan mengesan ralat yang lebih daripada 2 bit yang berlaku dalam bit data yang dihantar. Seterusnya membetulkan ralat tersebut.

RUJUKAN

Rujukan

Monograf dan Bibliografi :

- Halsall Fred. *Data Communications, Computer Networks and Open System.* (4th Edition).Addison-Wesley. Hallow, England.
- Jr.,Clark George C.,Cain J Bibb. *Error-Correction Coding for Digital Communications.* Plenum Press. New York.
- Kuo F. Franklin. *Error Control Coding: Fundamentals and Application.* Prentice Hall. New Jersey.
- Mano M.M and Kime C.R (1997), *Logic Computer and Design Fundamentals*, pp. 220-231, Prentice Hall, Inc, New Jersey.
- Wickew B. Stephen. *Error Control System for digital Communication and Storage.* (1995) . Prentice Hall. Englewood Cliffs.
- Zaini Md. Jana. *Sistem Komunikasi Data.* Dewan Bahasa dan Pustaka, Kuala Lumpur.

Tesis :

- Mohd Hisyam b. Abd fatah (2000). *Error correcting & detection code*. Bachelor Thesis. Universiti Malaya.
- Ng Choon Boon (*1996/97*). *Error correction coding for digital communications* .session *on 1996/97.* Bachelor Thesis. Universiti Malaya.

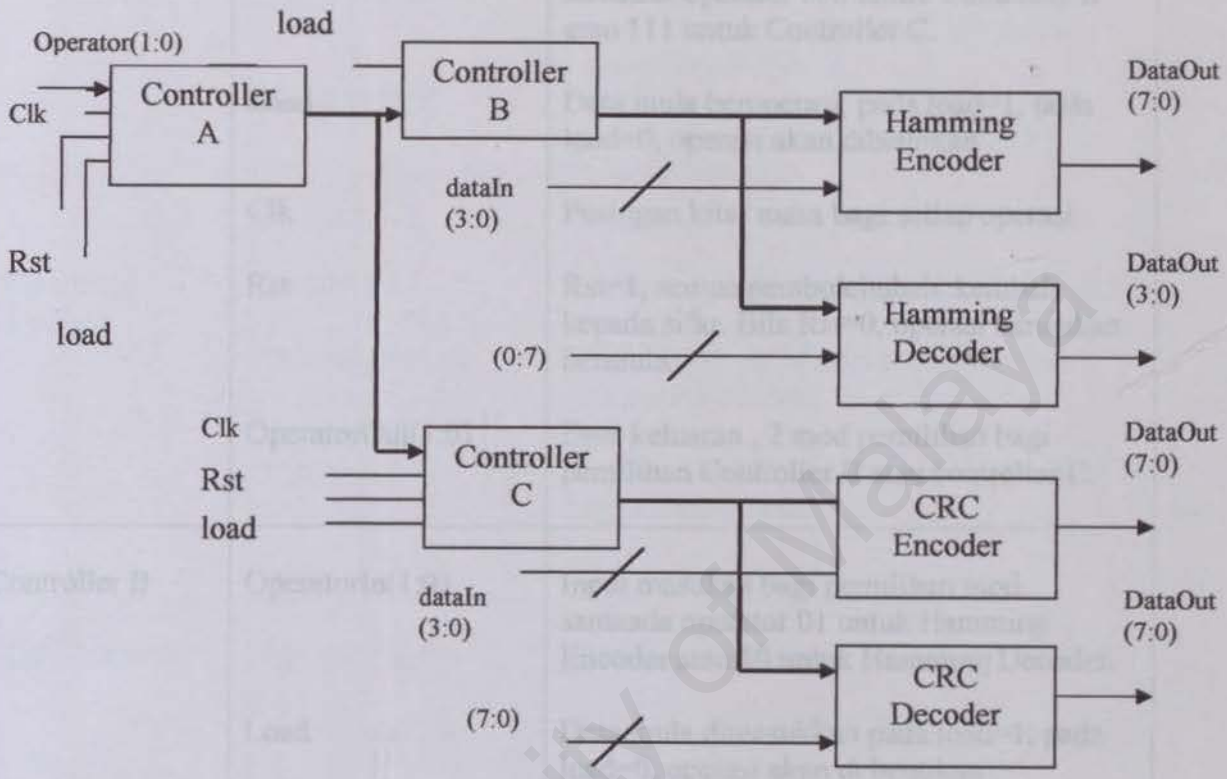
- Nurul Amin Badrul (2000). *Error correcting & detection code–A look at BCH codes*. Bachelor Thesis.Universiti Malaya.

Web:

- http://www.ee.unb.ca/tervo/ee4253/hamming.htm
- http://www.rad.com/networks/1994/err_con/hamming.htm
- http://www-s.ti.com/sc/psheets/spra530/spra530.pdf
- http://www.its.bldrdoc.gov/fs-1037/dir-017/_2528.htm
- http://cegt201.bradley.edu/projects/proj2002/hcedd/seniorprojectpaper.html
- http://pdc.ro.nu/hamming.html
- http://www.xup.msu.edu/labs/lab1/lab1.htm
- http://www.xup.msu.edu/labs/lab2/lab2.htm
- http://www.xup.msu.edu/labs/lab3/lab3.htm
- http://wwbchat.com/webx?23@198.H3oCacocwLo^0@14%40
- http://assembly.nerdworld.com/directory/computers.html
- http://www.cms.dmu.ac.uk/~se00jf/nets_notes/chunk_html/x521.html
- http://www.eecg.toronto.edu/~de/Cn-edc.pdf
- http://www.erg.abdn.ac.uk/users/gorry/course/dlpages/crc.html

LAMPIRAN

Pengenalan



Rajah 1: Gambarajah Top Level Bagi Hamming dan CRC

Pengguna boleh membuat pemilihan samaada ingin menggunakan

i. modul CRC atau Hamming
ii. modul Hamming Encoder atau modul Hamming Decoder
iii. Modul CRC Encoder atau modul CRC Decoder

Disini pengguna perlu memilih mod yang mana ia akan menentukan input masukan dan uga output keluaran . Mod-mod yang perlu dipilih ialah:

i. operator = 01 untuk Hamming Encoder
ii. operator = 10 untuk Hamming Decode
iii. operator = 01 untuk CRC Encoder
iv. operator = 10 untuk CRC Decoder

Penggunaan Komponen

Komponen	Pin	Fungsi
Controller A	OperatorIn(1:0)	Input masukan bagi pemilihan mod, samaada operator 000 untuk Controller B atau 111 untuk Controller C.
	Load	Data mula beroperasi pada load=1, pada load=0, operasi akan dihentikan .
	Clk	Pusingan kitar masa bagi setiap operasi
	Rst	Rst=1, semua pembolehubah kembali kepada sifar. Bila Rst=0, operasi baru akan bermula.
	OperatorOut(1:0)	Data keluaran , 2 mod pemilihan bagi pemilihan Controller B atau controller C.
Controller B	OperatorIn(1:0)	Input masukan bagi pemilihan mod, samaada operator 01 untuk Hamming Encoder atau 10 untuk Hamming Decoder.
	Load	Data mula dimasukkan pada load=1, pada load=0, operasi akan di hentikan .
	OperatorOut(1:0)	Output bagi pemilihan mod.
Controller C	OperatorIn(1:0)	Data input bagi pemilihan mod, samaada operator 01 untuk CRC Encoder atau 10 untuk CRC Decoder.
	Load	Data mula dimasukkan pada load=1, pada load=0, operasi akan dihentikan .
	Clk	Pusingan kitar masa bagi setiap masukan bit.
	Rst	Rst=1, semua pembolehubah kembali kepada sifar. Pada Rst=0, operasi baru akan bermula.

	OperatorOut(1:0)	Output bagi pemilihan mod untuk CRC encoder dan CRC decoder
Hamming Encoder	DataIn(3:0) OperatorIn(1:0) DataOut(7:0)	Data input sebanyak 3 bit dimasukkan secara sesiri Input untuk pemilihan mod, 01 untuk Hamming encoder Data keluaran sebanyak 7 bit selepas di kenakan operasi
Hamming Decoder	DataIn(7:0) OperatorIn(1:0) DataOut(3:0)	Input masukan sebanyak 7 bit secara sesiri Input untuk pemilihan mod, 10 untuk Hamming decoder Output keluaran sebanyak 4 bit setelah 7 bit input dinyahkodkan
CRC Encoder	DataIn(7:0) OperatorIn(1:0) DataOut(7:0)	Data input sebanyak 3 bit secara selari. Input untuk pemilihan mod, 01 untuk CRC encoder Data keluaran sebanyak 7 bit selepas di kenakan operasi
CRC Decoder	DataIn(7:0) OperatorIn(1:0) DataOut(7:0)	Input masukan sebanyak 7 bit secara selari Input untuk pemilihan mod, 10 untuk CRC decoder Output keluaran sebanyak 7 bit setelah pembetulan ralat .