UNIVERSITI MALAYA

SARJANA MUDA SAINS KOMPUTER DENGAN KEPUJIAN

Rekabentuk Cip Pemampatan Data

Nama : Shah Rizal Bin Ismail

No. Matrik : WEK 98345

Penyelia : Encik Zaidi Razak

Moderator : Puan Sameem Abdul Karem

WXES3182
PROJEK ILMIAH TAHAP AKHIR

FAKULTI SAINS KOMPUTER & TEKNOLOGI MAKLUMAT
UNIVERSITI MALAYA

Abstrak

Dokumen ini telah disiapkan untuk membuat cadangan projek Ilmiah Tahun Akhir yang bertajuk Rekabentuk Perkakasan Cip Pemampatan Data. Bahagian pertama adalah berkenaan pengenalan yang memberi gambaran tentang tujuan, objektif dan kepentingan projek juga keperluan perkakasan dan perisian dalam proses pembangunan. Bahagian kedua pula, tumpuan diberikan kepada pemampatan data iaitu jenis teknik, algoritma Huffman dan contoh penggunaan Huffman. Dalam bahagian ketiga, pengenalan terhadap perisian yang akan digunakan iaitu perisian VHDL.

Bahagian keempat dokumen menyentuh tentang pendekatan pembangunan modul rekabentuk dan rekabentuk akhir perkakasan yang akan dibangunkan. Juga turut dimasukkan ialah gambarajah status bertujuan untuk memudahkan pemahaman. Dalam bahagian lima pula, implementasi projek akan diterangkan dengan perubahan modul yang telah dibuat.

Segala perbincangan dan masalah yang telah berlaku semasa pembangunan projek akan dibincangkan dalam bahagian enam, manakala kesimpulan projek akan dinyatakan dalam bahagian tujuh.

Penghargaan

Assalammualaikum dan salam sejahtera.

Pertama sekali saya bersyukur ke hadirat Allah Yang Maha Esa kerana dengan limpah kurniaNya dapat saya menyiapkan laporan Ilmiah Tahun Akhir ini.

Sekalung tahniah buat diri sendiri kerana akhirnya berjaya menyiapkan laporan pada masanya.

Jutaan terima kasih buat penyelia saya, Encik Zaidi Razak kerana telah memberikan saya peluang, panduan dan bimbingan sepanjang proses menyiapkan laporan. Tidak dilupakan moderator saya, Puan Sameem Abdul Karem.

Terima kasih yang tidak terhingga juga kepada keluarga tercinta kerana telah memberikan kata-kata semangat dan sokongan moral buat saya.

Ribuan terima kasih kepada Encik Yamani, tutor saya sepanjang proses pembangunan kod. Beliau banyak mengajar dan tidak jemu untuk membantu sekiranya diperlukan. Semoga beliau sukses dalam kerjayanya.

Buat rakan-rakan seperjuangan seperti Najmee, Hemley, Bobot, Lalok, J.r, Shah (GTi), Fadhil dan yang lain; terima kasih atas sokongan dan 'kutukan' yang kalian berikan.

Terima kasih semua.

Kandungan

Bab 2 : Kajian Pemampatan Data

Bab 3 :VHDL

Senarai rajah

Senarai jadual

Bab 1

Bab 1 : Pengenalan Projek

1.1 Pengenalan

Proses pemampatan data merupakan suatu proses yang sememangnya diperlukan dalam dunia komputer. Namun, kebanyakan proses pemampatan adalah melalui proses perisian atau program tertentu dan data yang dimampat adalah dari jenis analog ke digital.

Dalam Projek Ilmiah 1 ini, saya ingin menampilkan rekabentuk cip yang seterusnya akan dijadikan satu perkakasan. Perkakasan yang akan dihasilkan ialah cip pemampatan data. Cip pemampatan data adalah merupakan suatu perkakasan yang akan memudahkan proses pemampatan. Proses pemampatan akan terus melalui cip ini dan tidak lagi memerlukan proses pemampatan dari perisian. Namun, cip yang akan dihasilkan cuma akan memampatkan data dalam bentuk jujukan binari '0' dan '1'.

Kenapa kita memerlukan perkakasan yang sedemikian? Satu jawapan yang mudah ialah kita perlukan satu kemudahan dalam proses pemampatan data. Jika sebelum ini proses pemampatan perlu melalui proses perisian, dengan terhasilnya cip ini, perkara sedemikian tidak perlu lagi. Cip ini boleh diimplementasi dalam sistem elektronik seperti telefon bimbit yang menggunakan aplikasi WAP atau mana-mana sistem yang memerlukan proses pemampatan data.

Cip ini akan direkabentuk dengan menggunakan perisian VHDL, iaitu suatu perisian yang khas untuk merekabentuk suatu perkakasan manakala teknik pemampatan yang akan digunakan ialah teknik pemampatan Huffman.

1.2 Objektif Projek

Terdapat dua objektif utama dalam menghasilkan projek ini:

1.2.1 Menghasilkan cip pemampatan data sebagai suatu perkakasan.

Tujuan projek ini dilakukan adalah untuk saya merekabentuk suatu cip. Ini merangkumi kajian dari segi rekabentuk, keperluan perisian, penyelidikan dan bagaimana proses akhir untuk menghasilkan cip tersebut.

Cip yang akan dihasilkan akan menjadi salah satu komponen atau perkakasan dalam suatu sistem komputer.

1.2.2 Memberikan kemudahan dalam proses pemampatan data.

Pemampatan data sekarang kebanyakannya menggunakan proses dari perisian. Jadi projek ini bertujuan untuk memberikan alternatif lain iaitu dengan menggunakan cip sebagai medium untuk pemampatan data. Segala data yang hendak dihantar akan dimampat dalam cip ini dan akan dihantar ke destinasi. Dengan cara ini, proses pemampatan akan menjadi lebih mudah kerana tidak perlu proses dari perisian.

1.3 Skop Projek

Untuk menghasilkan suatu projek, kita memerlukan suatu julat tertentu untuk memudahkan kita melakukan penyelidikan mengenai projek yang ingin kita lakukan. Penyelidikan mengenai cip pemampatan data adalah merangkumi perkara-perkara berikut:

1.3.1 Rekabentuk perkakasan yang akan dihasilkan.

Perkara ini merangkumi jenis rekabentuk yang paling optima, jenis perkakasan yang diperlukan, keperluan perisian yang sesuai dan kemudahan untuk mengimplementasikan projek.

1.3.2 Penentuan jenis data yang akan dimampat.

Pembangunan projek hanya dalam skop selepas data diterima dalam bentuk digital. Ia tidak melibatkan penukaran data dari analog ke digital atau digital ke analog. Jujukan data yang ingin dimampat akan dianggap dalam bentuk digital iaitu perwakilan binari "0" atau "1".

1.3.3 1 dimensi data.

Data-data dalam komputer boleh terdiri dari 1, 2 atau 3 dimensi. Untuk projek ini, kajian pemampatan akan dilakukan terhadap data 1 dimensi. Data 1 dimensi adalah data yang mempunyai 1 permukaan sahaja. Data jenis ini kebanyakannya digunakan dalam penulisan teks dalam komputer.

1.4 Kepentingan Projek

- Memenuhi keperluan bagi penghasilan komputer bersaiz lebih kecil.

Teknologi yang berkembang memberi penumpuan terhadap penghasilan komputer bersaiz kecil. Perkakasan yang akan dihasilkan akan dapat memenuhi keperluan tersebut memandangkan saiz perkakasan adalah kecil.

- Menyokong pemampatan data yang mudah.

  Dengan adanya perkakasan ini, proses pemampatan manjadi semakin mudah dan lebih pantas.

- Menambah kebolehpercayaan peranti.

  Fungsi peranti akan semakin bertambah dengan penghasilan perkakasan ini. Ini bermakna peranan yang dimainkan oleh perkakasan akan semakin bertambah dan seterusnya menambah kebolehpercayaan peranti.

1.5 Perkakasan dan perisian

Projek ini akan dibangunkan dengan menggunakan komputer peribadi di mana sistem pengendalian yang digunakan ialah Windows 98. Kapasiti penyimpanan pula adalah dalam lingkungan 4.0 sehingga 20.4 GB.

Perisian utama yang akan digunakan ialah Xilinx di mana ia menggunakan bahasa pengaturcaraan **V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage (VHDL). Xilinx ini akan digunakan dalam fasa pengkodan dan pengujian.

Selain itu terdapat beberapa perisian lain yang turut digunakan dalam pembangunan projek ini. Senarai perisian dan aplikasinya adalah seperti berikut:

- Microsoft Word 2000 dan Word XP

  Mendokumentasi projek.

- Microsoft Excel

  Membina jadual carta Granttt

- Microsoft Power Point

  Pembentangan kertas projek

- Turbo C++ dan Matlab

  Program sokongan

1.6 Penjadualan projek

Penjadualan adalah perlu dalam memastikan projek dapat disiapkan dalam masa yang ditetapkan. Untuk memastikan perkara ini, satu jadual ringkas yang menyatakan tempoh masa dan aktiviti-aktiviti sepanjang proses pembangunan dibuat.Berikut merupakan jadual bagi projek ini :

TARIKH / AKTIVITI	JUN 2001				JULAI 2001				OGOS 2001				SEPT 2001			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Pemilihan Tajuk																
Kajian Literasi																
Perancangan																
Analisa Sistem																
Rekabentuk																
Ulasan																
Dokumentasi																

Jadual 1.1: Penjadualan Tugasan

Bab 2

Bab 2 : Kajian Pemampatan Data

2.1 Pengenalan

Pemampatan data merujuk kepada proses pertukaran "jasad" data kepada bentuk yang lebih kecil, yang mana ia boleh dijanakan ke saiz yang asal semula. Pemampatan data adalah perlu untuk tujuan penjimatan ruang simpanan dan kecepatan penghantaran data melalui rangkaian komputer.

Perkara penting dalam pemampatan data ialah "ratio pemampatan"; iaitu saiz ratio fail yang telah dimampatkan kepada yang belum dimampatkan. Contohnya, katakan fail 100kb dimampatkan kepada 50kb. Kita boleh katakan pemampatan saiz adalah pada faktor dua, iaitu ratio pemampatan 2:1. Terdapat 2 teknik atau bentuk pemampatan data :

- 'lossless'.
- 'lossy'.

2.1.1 'Lossless'

Pemampatan data secara 'lossless' digunakan apabila data perlu dinyahmampat seperti keadaan sebelum pemampatan berlaku. Fail teks disimpan dengan menggunakan teknik 'lossless', memandangkan kehilangan satu karektor

akan mencacatkan teks yang hendak disampaikan. Secara umumnya juga data-data lain seperti imej, grafik, audio dan video juga menggunakan teknik ini.

Namun, teknik ini mempunyai limit ratio pemampatan iaitu antara 2:1 hingga 8:1. Antara algoritma yang popular dalam teknik ini ialah Huffman, Adptive Huffman dan LZW (Lempel-Ziv-Welch).

2.1.2 'Lossy'

Pemampatan cara ini berfungsi dengan tanggapan bahawa data tidak perlu disimpan dengan sempurna. Kebanyakan maklumat secara mudah boleh dibuang daripada imej, video dan audio; dan apabila dinyahmampat data masih mempunyai kualiti. Ratio pemampatan dengan cara ini lebih besar daripada 'lossless'.

2.1.3 Kegunaan Pemampatan Data

Kemajuan teknologi pekomputeran telah menyebabkan penggunaan data semakin besar dan satu isu yang timbul ialah penyimpanan data yang bersaiz besar. Untuk itu, proses pemampatan adalah diperlukan memandangkan kaedah pemampatan boleh menjimatkan penggunaan ruang simpanan.

Selain itu, proses pemampatan juga akan memudahkan proses penghantaran melalui rangkaian. Ini kerana pemampatan akan mengecilkan saiz fail atau data dan seterusnya akan memudahkan data atau fail dihantar menerusi rangkaian. Kajian

mendapati data yang telah dimampat adalah lebih laju penghantarannya berbanding data yang tidak dimampat.

Kesimpulannya, proses pemampatan adalah suatu proses yang penting dalam suasana persekitaran pekomputeran masa kini.

2.1.4 Kajian pemampatan data

Dalam bab ini, kajian terhadap algoritma-algoritma terpilih akan dilakukan. Dua jenis algoritma pemampatan yang dipilih ialah algoritma Huffman dan juga algoritma LZW iaitu "Lemple-Ziv-Welch algorithm".

Dalam bab ini juga akan disenaraikan kelebihan dan kekurangan algoritma pemampatan masing-masing.

2.2 Pengkodan Huffman

2.2.1 Sejarah ringkas

Algoritma Huffman kebiasaannya digunakan dalam program pemampatan komersial, mesin fax dan sebagai sebahagian daripada algoritma pemampatan JPEG. Ia dinamakan sempena nama D.A Huffman, seorang yang membentangkan kertas kerja pada tahun 1952 bertajuk "A Method for the construction of Minimum Redundancy Codes" yang menerangkan algoritma beliau.

2.2.2 Pengenalan

Huffman kod merupakan salah satu teknik pemampatan secara 'lossless', yang mana karektor dalam fail data akan ditukar kepada kod binari. Ia merupakan kod yang menjadi asas atau ilham kepada kod pemampatan yang lain.

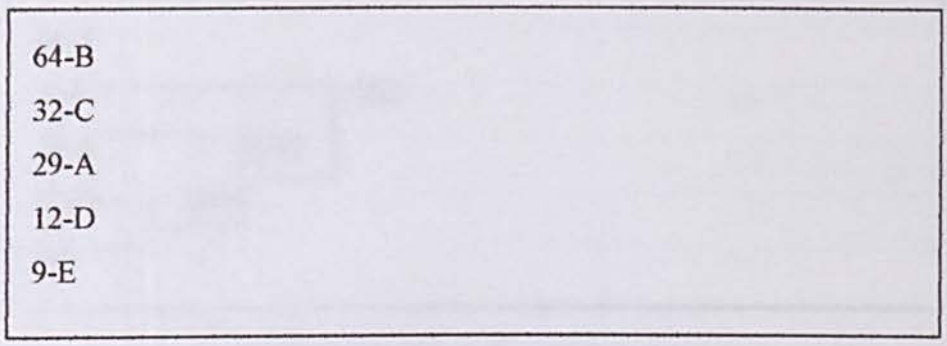
Konsep Huffman adalah penambahan dua frekuensi yang rendah jumlahnya. Kemudian, ia akan terus menambah dengan jumlah tambahan merupakan nilai seminimum yang mungkin. Selepas menambah kesemua huruf, pengkodan binari akan dilakukan. Huruf yang paling kerap akan mempunyai nilai binari yang rendah. Untuk melihat bagaimana kod Huffman berfungsi, kita lihat contoh yang diberikan.

2.2.3 Algoritma pemampatan

Katakan suatu fail teks yang hendak dimampatkan mengandungi karektor-karektor dan bilangannya seperti berikut:

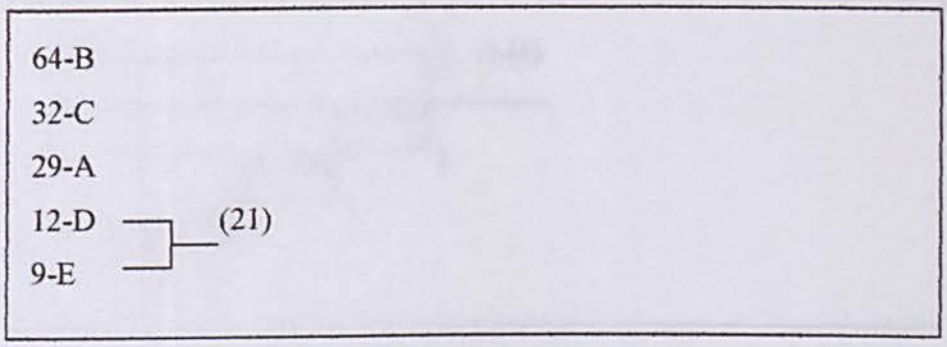
A-29, B-64, C-32, D-12, E-9.

Langkah yang pertama ialah menyusun huruf yang paling tinggi frekuensinya (bilangannya) di bahagian atas hingga ke frekuensi yang terendah di bahagian bawah.



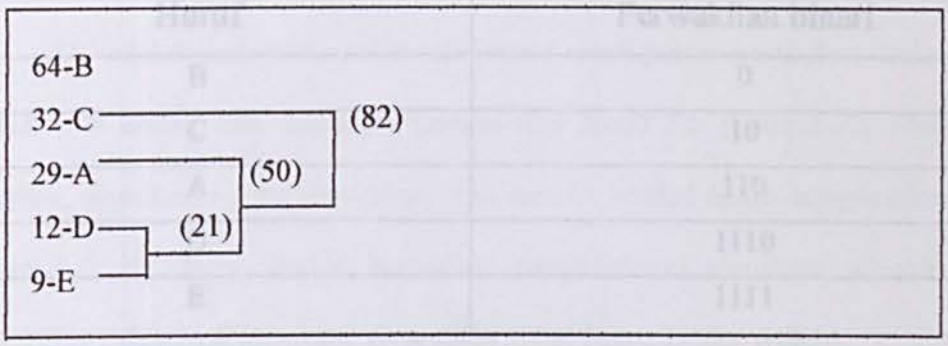
Rajah 2.1 : Penyusunan huruf mengikut frekuensi

Kemudian, dua frekuensi yang terendah akan ditambah bilangannya.



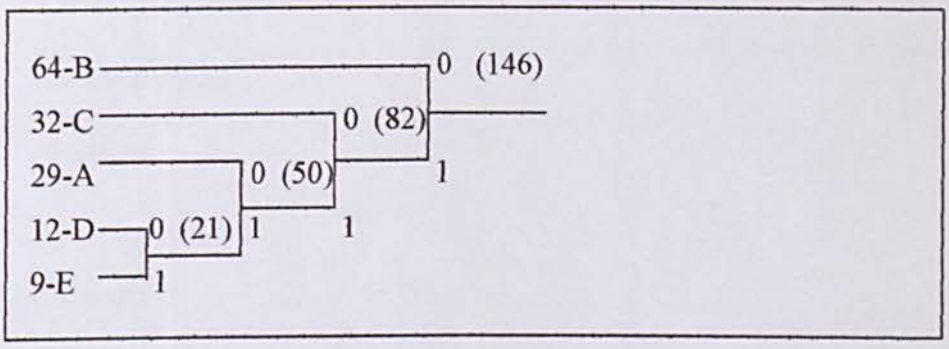
Rajah 2.2 : Penambahan frekuensi yang benilai paling rendah

Selepas itu, tambahkan dua frekuensi yang mempunyai jumlah yang minima sehingga lengkap.



Rajah 2.3 : Penambahan frekuensi berterusan

Akhir sekali, pepohon Huffman yang lengkap dihasilkan.



Rajah 2.4 : Penambahan yang lengkap

Berikut pula adalah jadual keputusan atau output dalam bentuk binari bagi setiap huruf.

Huruf	Perwakilan binari
B	0
C	10
A	110
D	1110
E	1111

Jadual 2.1 : Perwakilan pemampatan data

Jadi, output yang akan dihantar keluar ialah 01011011101111 (iaitu jujukan dari yang paling kerap kepada yang paling kurang kekerapannya).

2.3 Pengkodan LZW (Lempel-Ziv-Welch)

2.3.1 Sejarah ringkas

LZW adalah merupakan salah satu metod untuk pemampatan data dari teknik "lossless". Ia direka oleh Abraham Lempel dan Jacob Ziv (Lempel-Ziv Method). Kemudian, algoritma LZ ini dimajukan oleh mereka berdua untuk menghasilkan LZ 77 dan LZ 78. Terry Welch kemudian mengubahsuai algoritma mereka dan menghasilkan algortima baru yang dinamakan LZW (Lempel-Ziv-Welch).

2.3.2 Konsep

Pemampatan bagi metod ini memerlukan komponen-komponen berikut:

- Kamus (dictionary)
- Penimbal (buffer)
- Algoritma

Kamus (dictionary)

Untuk proses mampat dan nyahmampat, LZW menggunakan kamus yang mempunyai 2 barisan. Baris pertama mewakili kod, manakala baris kedua mewakili 'string' yang berkenaan.

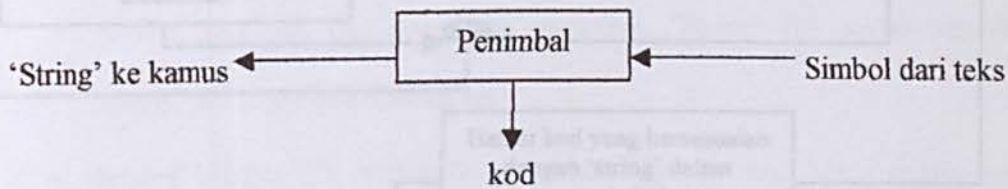
Sebelum pemampatan data, kamus akan di'initialize'kan dengan set karektor dalam teks. Jika teks mengandungi 3 simbol; A, B, dan C, kamus akan mempunyai 3 baris. Ia akan menjadi banyak apabila teks semakin banyak dimampatkan. Berikut adalah rajah bagi kamus.

1	2	3	
A	B	C	

Rajah 2.5 : Kamus untuk tiga simbol

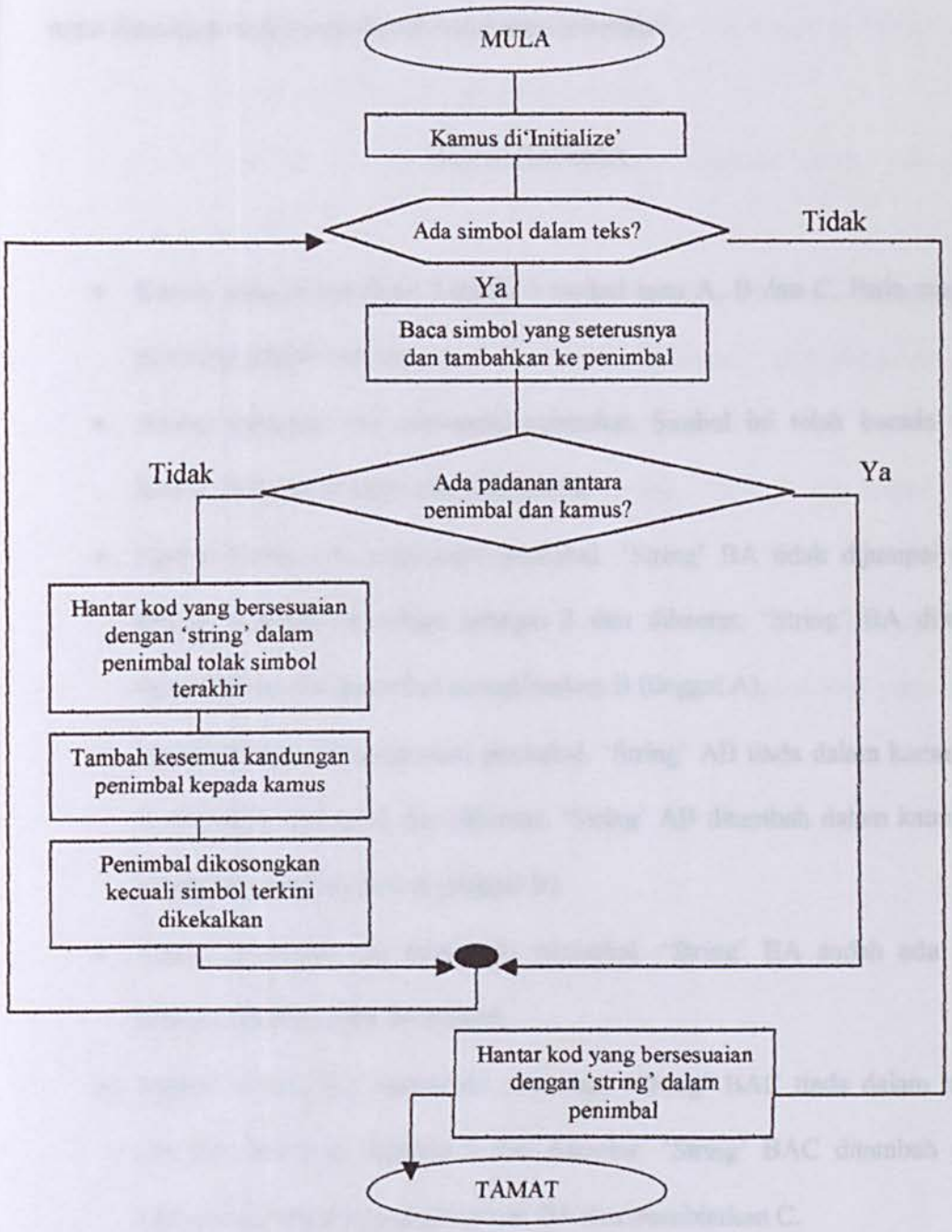
Penimbal (buffer)

LZW menggunakan penimbal. Simbol-simbol memasuki penimbal satu demi satu. 'String' dipindahkan ke kamus dan kod yang berkenaan dikeluarkan berdasarkan algoritma. Rajah berikut merupakan penimbal.



Rajah 2.6 : Penimbal dalam LZW

2.3.3 Algoritma pemampatan



Rajah 2.7 : Rajah menunjukkan proses pemampatan LZW

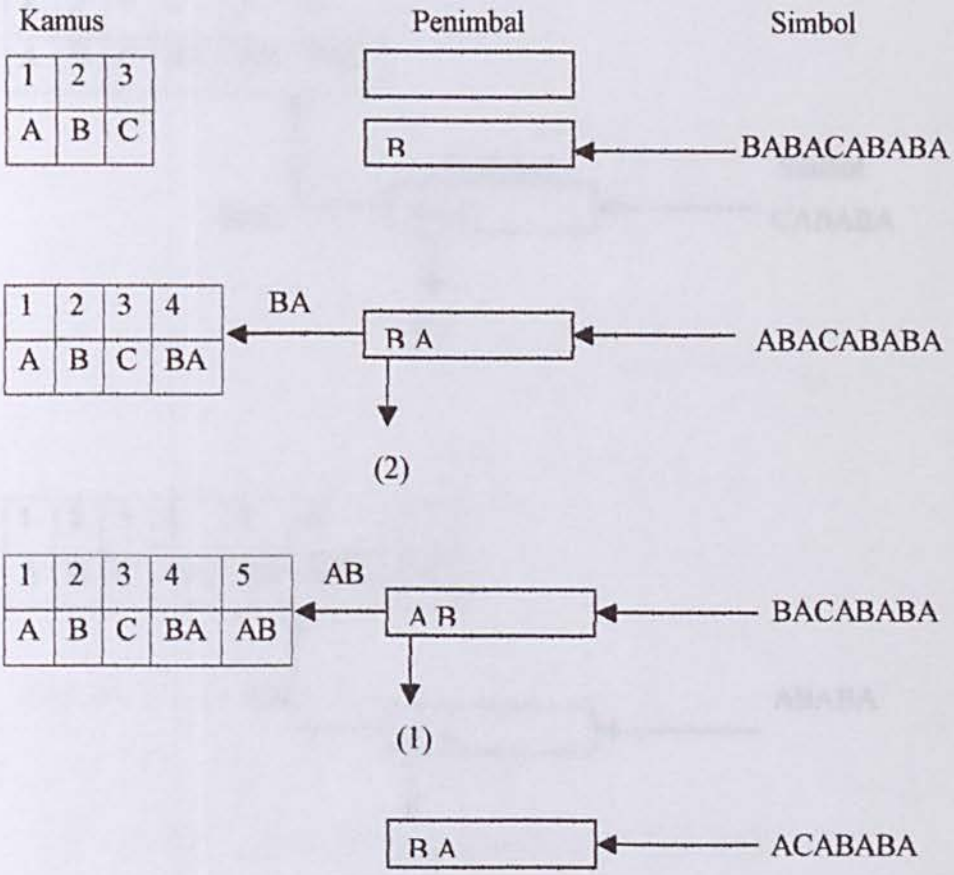
Kita akan melihat bagaimana algoritma LZW berfungsi. Katakan contoh teks yang ingin dimampat mempunyai huruf-huruf seperti berikut:

BABACABABA

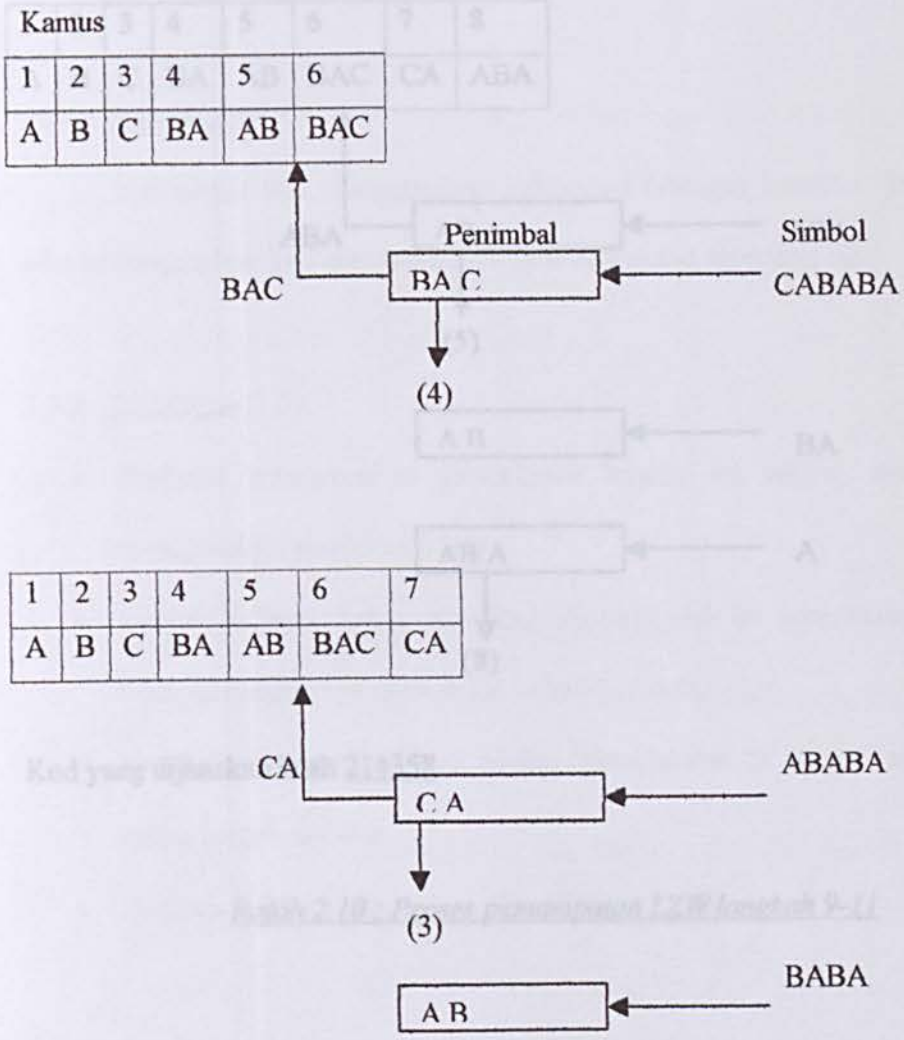
- Kamus akan di'initialize' kepada 3 simbol iaitu A, B dan C. Pada masa ini, penimbal adalah kosong.
- Simbol pertama, (B) memasuki penimbal. Simbol ini telah berada dalam kamus, jadi proses algoritma diteruskan.
- Simbol kedua, (A) memasuki penimbal. 'String' BA tidak dijumpai dalam kamus, jadi (B) dikodkan sebagai 2 dan dihantar. 'String' BA ditambah dalam kamus dan penimbal mengeluarkan B (tinggal A).
- Simbol ketiga, (B) memasuki penimbal. 'String' AB tiada dalam kamus, jadi A dikodkan sebagai 1 dan dihantar. 'String' AB ditambah dalam kamus dan penimbal mengeluarkan A (tinggal B).
- Simbol keempat, (A) memasuki penimbal. 'String' BA sudah ada dalam kamus, jadi algoritma diteruskan.
- Simbol kelima, (C) memasuki penimbal. 'String' BAC tiada dalam kamus, jadi BA dikodkan sebagai 4 dan dihantar. 'String' BAC ditambah dalam kamus dan penimbal mengeluarkan BA dan membiarkan C.

- Simbol keenam, (A) memasuki penimbal. 'String' CA tiada dalam kamus, jadi C dikodkan sebagai 3 dan dihantar. 'String' CA ditambah dalam kamus. Penimbal menyingkirkan C dan membiarkan A.
- Simbol ketujuh, (B) memasuki penimbal dan didapati 'string' AB wujud. Algortima diteruskan.
- Simbol kelapan, (A) masuk dan didapati 'string' ABA tiada dalam kamus. Jadi, Ab dikodkan sebagai 5 dan dihantar. 'String' ABA ditambah dalam kamus dan penimbal mengeluarkan AB dan tinggalkan A.
- Simbol kesembilan, (B) memasuki penimbal. 'String' AB wujud dalam kamus, jadi algortima diteruskan.
- Simbol kesepuluh, (A) memasuki penimbal. 'String' ABA didapati wujud dalam kamus dan algoritma ditamatkan kerana tiada lagi simbol yang tinggal. ABA dikodkan sebagai 8.
- Output yang dihasilkan ialah 214358.

Berikut merupakan rajah yang merujuk kepada contoh yang telah diberikan.

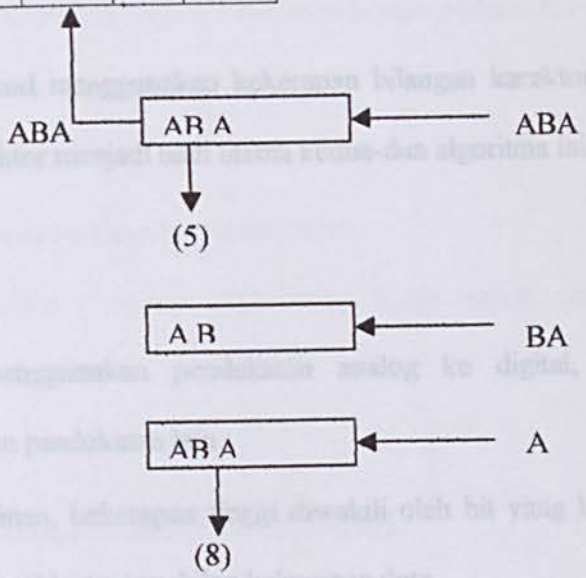


Rajah 2.8 : Proses pemampatan LZW langkah 1-5



Rajah 2.9 :Proses pemampatan LZW langkah 6-8

1	2	3	4	5	6	7	8
A	B	C	BA	AB	BAC	CA	ABA



Kod yang dijanakan ialah 214358

Rajah 2.10 : Proses pemampatan LZW langkah 9-11

2.4 Perbandingan antara Huffman dan LZW

2.4.1 Persamaan

Kedua-dua kod menggunakan kekerapan bilangan karektor. Ini bermaksud, nilai kekerapan karektor menjadi nadi utama kedua-dua algoritma ini.

2.4.2 Perbezaan

- Huffman menggunakan pendekatan analog ke digital, manakala LZW menggunakan pendekatan lain.
- Dalam Huffman, kekerapan tinggi diwakili oleh bit yang kurang. LZW pula tidak menampakkan perwakilan kekerapan data.
- Kod keluaran bagi Huffman adalah dalam bentuk '0' dan '1' manakala LZW dalam bentuk nombor.
- Huffman tidak menggunakan penimbal dan kamus.

2.5 Pemilihan algortima

Pemilihan algoritma dilakukan setelah meneliti beberapa perkara berikut:

- Pendekatan yang sesuai.
- Kepantasan proses.
- Boleh diimplementasikan ke dalam projek.

Setelah penelitian dibuat, algoritma Huffman dipilih untuk dilaksanakan dalam projek ini.

2.6 Kebaikan menggunakan kod Huffman

Terdapat beberapa sebab kenapa saya menggunakan kod Huffman:

2.6.1 Merupakan kod pemampatan yang mudah difahami dengan konsep yang ringkas.

Huffman menyediakan algoritma yang senang difahami. Konsep yang diperkenalkan adalah suatu konsep yang amat ringkas dan perlaksanaan algoritma adalah mudah.

2.6.2 Senang untuk diimplementasikan ke dalam aturcara.

Oleh kerana ia merupakan algoritma yang mudah, maka Huffman telah digunakan dalam banyak aturcara seperti C, C++, Visual Basic dan Visual C++. Dalam projek ini, saya akan cuba untuk mengimplementasikan Huffman dalam VHDL.

2.6.3 Sesuai untuk data jenis statik.

Data statik adalah merupakan data yang tidak bergerak atau beranimasi. Antara contoh data statik ialah teks, imej dan grafik. Untuk memampatkan data jenis ini, algortima Huffman adalah antara algoritma yang sesuai.

2.6.4 Perlaksanaan yang pantas.

Oleh kerana algoritma Huffman merupakan kaedah yang mudah, maka algoritmanya adalah mudah dan ringkas. Algoritma yang ringkas akan memberikan kepantasan dalam perlaksanaannya.

Bab 3

Bab 3 : VHDL Secara Ringkas

3.1 Pengenalan

VHDL adalah singkatan daripada ayat "Very high speed integrated circuit Hardware Description Language". Ia merupakan suatu bahasa pengaturcaraan yang boleh digunakan untuk cadangan simulasi , permodelan, percubaan, sintesis, rekabentuk dan mendokumentasikan sistem digital. VHDL memberikan format yang sesuai untuk gambaran hierarki suatu fungsi dan pendawaian sistem digital.

3.2 Unit rekabentuk VHDL

Terdapat 3 komponen asas:

- Entiti.
- Senibina.
- Konfigurasi.

2 komponen lagi yang boleh didapati dari program VHDL yang kompleks ialah :

- Pakej.
- Badan pakej.

3.2.1 Entiti

Entiti merupakan unit rekabentuk asas dan membina pengisytiharan 'port' yang bersambung (interface) kepada sistem. Ia boleh mewakili keseluruhan sistem, papan (board), cip, sel (cell) atau get.

Format entiti adalah :

```
ENTITI entity_name is
    PORT (port list);
END entity_name;
```

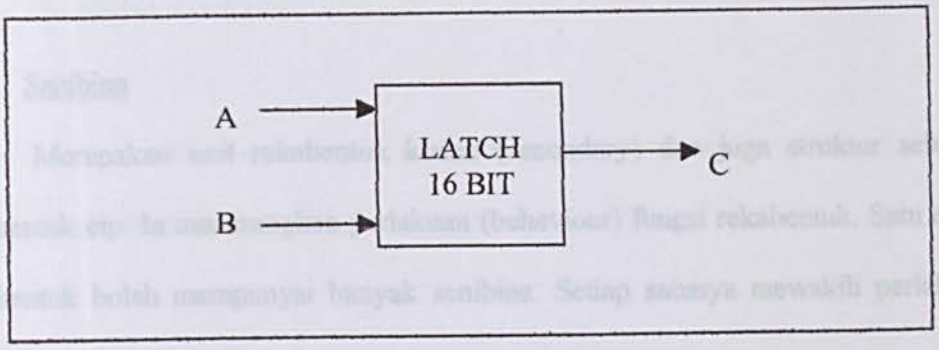
PORT- menyediakan saluran untuk komunikasi antara entiti dengan persekitarannya. Setiap 'port' mesti mempunyai nama , mod dan jenis.

Format PORT adalah :

```
PORT (port names : mode type_indication [bus])
```

Pengisytiharan 'BUS' dibuat jika 'PORT' disambung lebih dari satu keluaran dari modul atau sistem lain.

Gambarajah berikut menunjukkan diagram 16 bit LATCH.



Rajah 3.1: Diagram blok LATCH

Contoh pengisytiharan LATCH adalah seperti berikut:

```
Entity LATCH is
        PORT (A : in bit_vector (15 downto 0);
              B : in bit;
              C : out bit_vector (15 downto 0)
              );
End LATCH;
```

Berikut adalah jenis mod yang boleh diisytihar dalam pernyataan 'PORT':

- IN masukkan ke modul.
- OUT keluaran dari modul.
- INOUT isyarat dua arah (bi-directional signal).

- BUFFER pendaftar yang dilampirkan (attached) pada keluaran.

3.2.2 Senibina

Merupakan unit rekabentuk kedua (secondary) dan juga struktur sebenar rekabentuk cip. Ia menerangkan perlakuan (behaviour) fungsi rekabentuk. Satu entiti rekabentuk boleh mempunyai banyak senibina. Setiap satunya mewakili perlakuan rekabentuk yang berbeza.

Format :

```
Architecture architecture_name of entity_name is
        --kenyataan pengisytiharan
begin
        --kenyataan kerjasama (concurrent statement)
end architecture_name;
```

Berikut adalah contoh pengisytiharan senibina LATCH:

```
architecture BEHAV_LATCH of LATCH is
begin
if B='1' then
        C <= A;
```

```
end if;
end BEHAV_LATCH;
```

Nama 'port' entiti dalam senibina memberikan sambungan (linkage) antara pengisytiharan entiti dan senibina.

3.2.3 Konfigurasi

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

Format :

```
CONFIGURATION configuration_name of entity_nameis
    for architecture_name
    end for;
END configuration_name;
```

Berikut adalah contoh konfigurasi LATCH :

```
Configuration CONFIG_LATCH of LATCH is
    for BEHAV_LATCH
    end for;
```

```
END CONFIG_LATCH;
```

3.2.4 Pakej dan Badan Pakej

Pakej adalah merupakan suatu unit perpustakaan yang terdiri daripada pengisytiharan yang boleh digunakan dalam unit rekabentuk yang lain. Unit pakej terbahagi kepada 2 bahagian iaitu :

- Pengisytiharan.
- Badan.

Pengisytiharan suatu pakej mungkin terdiri daripada nilai tetap, jenis, fungsi dan pengisytiharan fungsi dan prosedur. Format ini adalah sama seperti pengisytiharan suatu entity. Kelakuan fungsi dan subaturcara sebenar pula dimasukkan ke dalam badan pakej.

Format untuk pengisytiharan pakej dan badan pakej adalah:

```
package nama_pakej is
        .....pengisytiharan
end nama_pakej ;
```

```
package body nama_pakej is
        ...fungsi atau subaturcara sebenar
end nama_pakej;
```

Berikut adalah contoh pakej dan badan pakej:

```
package LATCH is
begin
        subtype bit_16 is bit_vector (15 downto 0);
        type bit_16_array is array (integer range <>) of bit_16;
        function resolve_bit_16 (driver : in bit_16_array) return bit_16;
        subtype bus_bit_16 is resolve_bit_16 bit_16;
end LATCH;
package body LATCH is
        function resolve_16_bit (driver : in bit_16_array) return bit_16 is
        variable result : bit_16;
        begin
                result :=X "0000";
                for i in driver'range loop
                        result:=result or driver (i);
```

```
        end loop;
        return result;
        end resolve_bit_16;
end LATCH;
```

3.3 Simulasi VHDL

'Test bench' perlu dibuat sebelum proses simulasi dilakukan. Setiap modul yang dihasilkan mesti mempunyai satu 'test bench' untuk menguji operasinya.

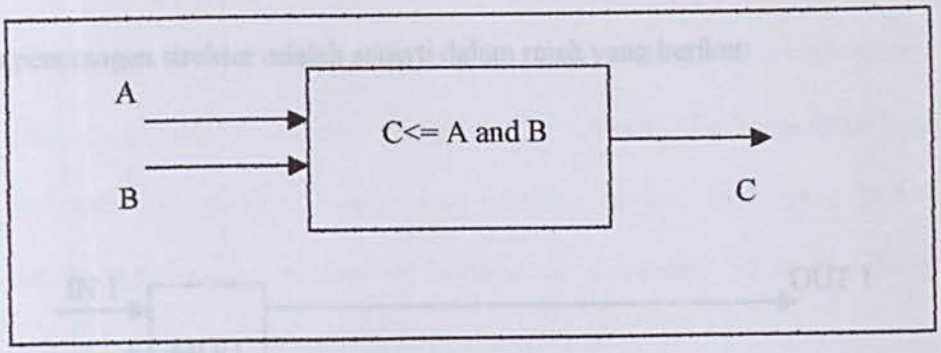
Format :

```
ENTITY test_bench_name IS
END test_bench_name;
    ARCHITECTURE architecture_name of test_bench_name IS
     COMPONENT entity_name
     END COMPONENT;
     --signal declaration
     BEGIN
     --concurrent statement
     END architecture_name;
```

3.4 Penerangan kelakuan

Penerangan kelakuan memodelkan perkakasan dengan menerangkan apa atau bagaimana sistem berkelakuan. Ia amat berguna bagi suatu sistem litar yang kecil dan ringkas. Walaubagaimanapun, penerangan kelakuan bagi litar yang lebih besar dan kompleks adalah sukar walaupun ianya tidak mustahil untuk dilakukan.

Contoh suatu penerangan kelakuan adalah seperti berikut:

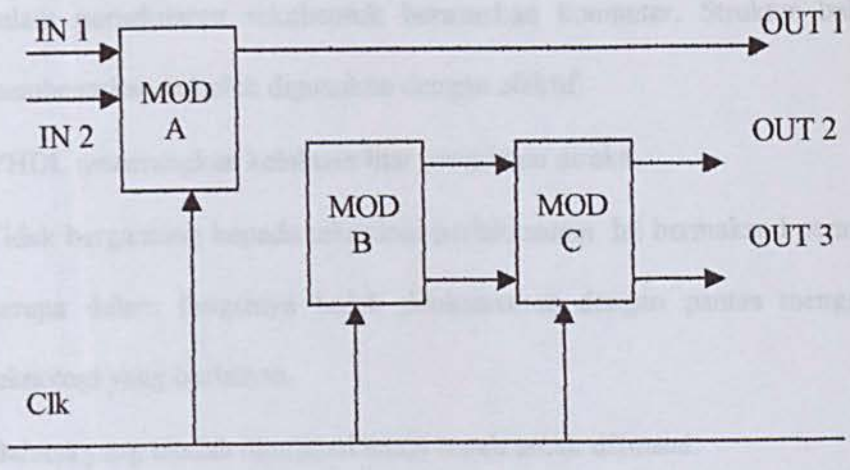


Rajah 3.2 : Model Kelakuan untuk Pembandingan

3.5 Penerangan Struktur

Penerangan struktur menerangkan entiti dalam konteks struktur yang tertakrif. Kita boleh mengambil suatu entiti kecil yang ditakrifkan menggunakan penerangan kelakuan untuk menghubungkan mereka. Penerangan struktur adalah suatu tugas yang besar dan perlu dipecahkan kepada tugasan yang lebih kecil bagi memenuhi keperluan pembangunan kod atau bagi meningkatkan kepantasan perlaksanaan.

Contoh penerangan struktur adalah seperti dalam rajah yang berikut:



Rajah 3.3 : Rajah Model Struktur

3.6 Kebaikan menggunakan VHDL

Tidak seperti bahasa pengaturacaraan perkakasan yang lain, VHDL menyediakan beberapa kelebihan seperti:

- VHDL mempunyai ciri-ciri yang menarik yang membenarkan aspek-aspek elektrikal litar dijelaskan. Contoh aspek litar ialah lengahan masa dan operasi fungsi.
- VHDL boleh 'capture' prestasi litar dalam bentuk 'test bench'. 'Test bench' adalah deskripsi VHDL terhadap stimulus litar dan output yang berkenaan.
- Merupakan suatu bahasa pengaturcaraan yang "berkomunikasi" pada peringkat rendah (low level) yang berguna, antara 'tool' yang berlainan dalam persekitaran rekabentuk berasaskan komputer. Struktur bahasanya membenarkan ia boleh digunakan dengan efektif.
- VHDL menerangkan kelakuan litar yang ingin direka.
- Tidak bergantung kepada teknologi perlaksanaan. Ini bermaksud peranti yang serupa dalam fungsinya boleh dilaksanakan dengan pantas menggunakan teknologi yang berlainan.
- Bahasa yang mudah dipelajari tetapi susah untuk dikuasai.

Bab 4

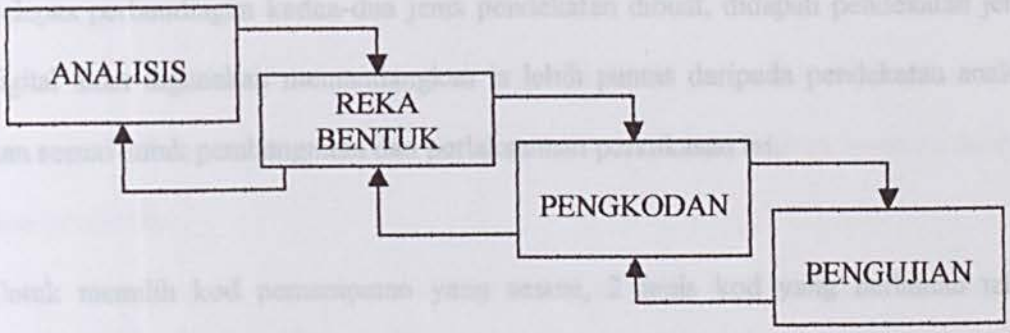
Bab 4 : Metodologi dan Rekabentuk

4.1 Metodologi

Metodologi adalah kaedah yang digunakan untuk membangunkan suatu sistem. Ia penting kerana melibatkan kos, masa dan tenaga. Pengambilan masa yang terlalu lama menyebabkab projek yang dijalankan tidak dapat disiapkan dalam masa yang diperuntukkan.

4.1.1 Model pembangunan perkakasan

Untuk membangunkan model perkakasan ini, terdapat 4 fasa pembangunan iaitu fasa analisis, rekabentuk, pembangunan kod dan pengujian. Kesemua fasa ini adalah mengikut pendekatan Kitaran Hayat Pembangunan Sistem (SDLC) dan dihubungkan secara dua hala seperti rajah berikut :



Rajah 4.1 : Rajah model pembangunan perkakasan

4.1.2 Penerangan setiap fasa

Fasa Analisis

Dalam fasa ini, analisa dijalankan ke atas perkara-perkara yang berikut:

- Jenis pendekatan pembangunan
- Jenis kod pemampatan data yang ingin digunakan
- Rekabentuk modul
- Risiko-risiko yang mungkin berlaku

Jenis pendekatan yang pembangunan yang boleh dilakukan terbahagi kepada dua:

- Pendekatan analog
- Pendekatan digital

Selepas perbandingan kedua-dua jenis pendekatan dibuat, didapati pendekatan jenis digital akan digunakan memandangkan ia lebih pantas daripada pendekatan analog dan sesuai untuk pembangunan dan perlaksanaan perkakasan ini.

Untuk memilih kod pemampatan yang sesuai, 2 jenis kod yang berlainan telah dianalisa dengan menyenaraikan kebaikan dan kelemahan setiapnya untuk perbandingan dan ia telah diterangkan dalam Bab 2.

Fasa Rekabentuk

Dalam fasa ini, rekabentuk modul dibuat berdasarkan keperluan perkakasan dan rekabentuk yang dihasilkan adalah dianggap yang optima iaitu mudah diimplementasikan dan boleh melaksanakan apa yang dikehendaki.

Fasa Pengkodan

Peringkat ini pula, proses pengkodan akan dilakukan berdasarkan rekabentuk modul yang telah dicadangkan. Proses pengkodan akan dilakukan dalam bahasa pengaturcaraan perkakasan VHDL.

Fasa Pengujian

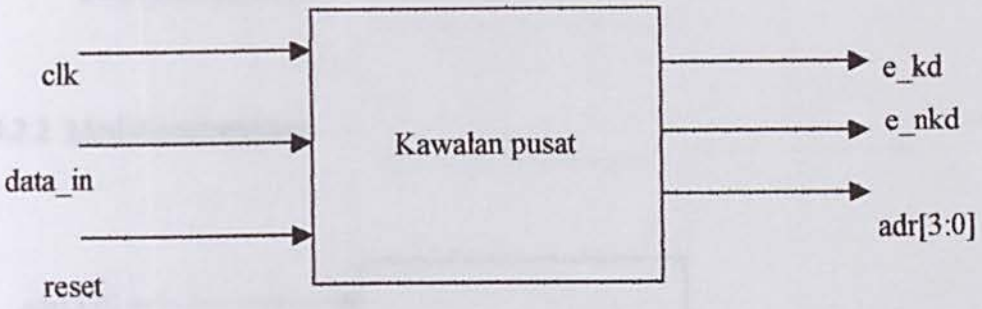
Fasa terakhir ini adalah fasa di mana kod-kod yang telah dibuat akan diuji sama ada terdapat ralat atau tidak. Kebiasaannya fasa ini dijalankan bersama dengan fasa pengkodan.

Setiap modul juga akan diuji sama ada ia boleh menjalankan fungsi seperti yang dinyatakan dalam rekabentuknya dan menghasilkan output yang dikehendaki.

4.2 Rekabentuk perkakasan yang dicadangkan

Setelah membuat analisis dan kajian, rekabentuk perkakasan yang akan dihasilkan adalah terdiri dari modul kawalan pusat, modul pemampatan dan modul penyahmampatan. Penerangan modul adalah seperti berikut:

4.2.1 Modul kawalan pusat



Rajah 4.2 : Modul kawalan pusat

Modul kawalan pusat adalah modul yang mengawal modul-modul lain dalam rekabentuk perkakasan. Ia menyediakan alamat bagi modul-modul lain dan mod pemampat atau penyahmampat.

Port input :

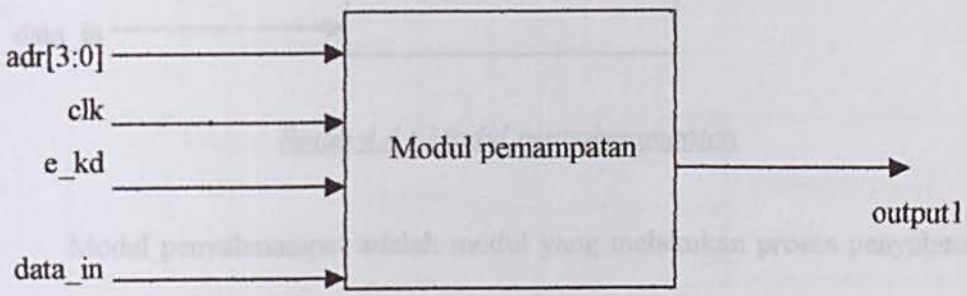
- 1 bit jam (clk) : menyediakan modul dengan jam sistem.
- data_in : data masukkan.
- reset : menyediakan sistem kembali ke asal.

Port output :

- 4 bit alamat (adr[3:0]) : memperuntukkan alamat untuk modul-modul lain.
- 1 bit isyarat e_kd : menentukan mod pemampatan untuk modul pemampatan.

- 1 bit isyarat e_nkd : menentukan mod penyahmampat untuk modul penyahmampatan.

4.2.2 Modul pemampatan



Rajah 4.3 : Modul pemampatan

Modul pemampatan adalah modul yang menjalankan proses pemampatan. Berikut adalah port input dan output:

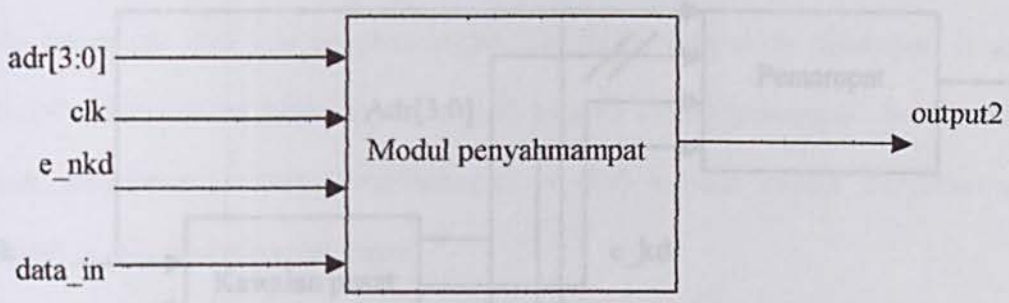
Port input :

- adr[3:0] : mengawal jujukan pengulangan semula.
- clk : menyediakan modul dengan jam sistem.
- e_kd : mengaktifkan modul pemampatan.
- data_in : jujukan data yang ingin dimampat.

Port output:

- output1 : jujukan data yang dimampat.

4.2.3 Modul penyahmampat



Rajah 4.4 : Modul penyahmampatan

Modul penyahmampat adalah modul yang melakukan proses penyahmampat. Berikut adalah port input dan output:

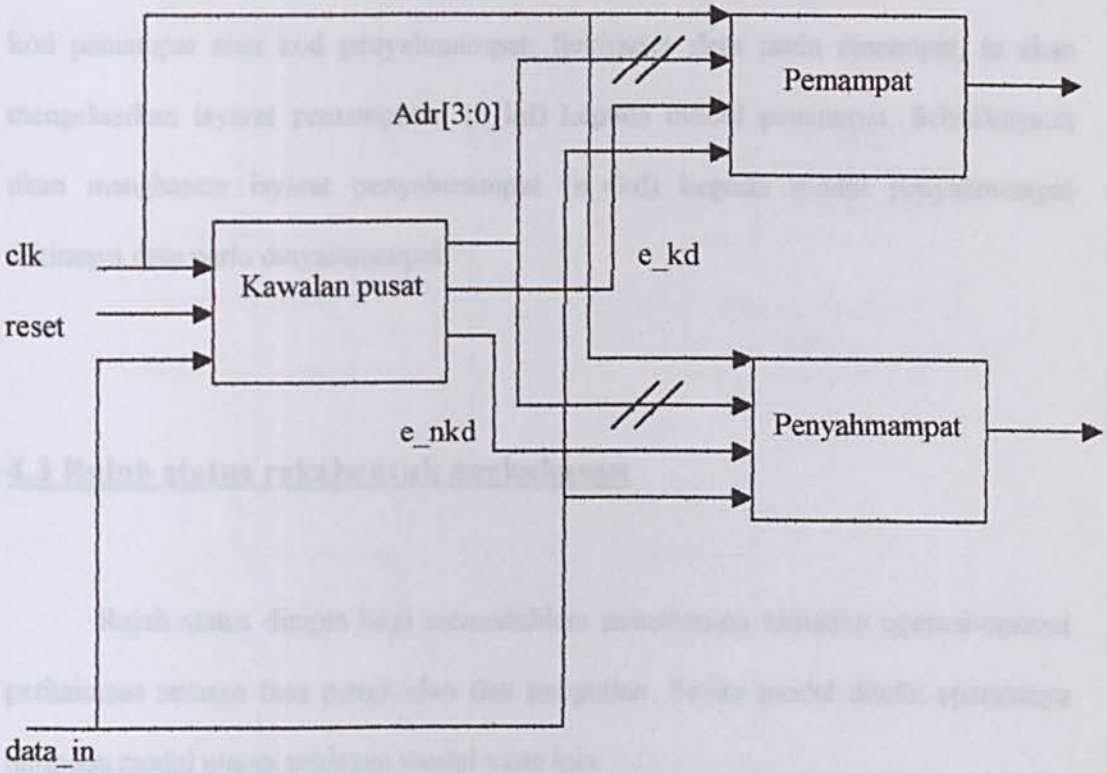
Port input :

- adr[3:0] : mengawal jujukan pengulangan semula.
- clk : menyediakan modul dengan jam sistem.
- e_nkd : mengaktifkan modul penyahmampatan.
- data_in : jujukan data yang ingin dinyahmampat.

Port output :

- output2 : kod yang telah dinyahmampat.

4.2.4 Gabungan modul-modul yang dicadangkan



Rajah 4.5 : Gabungan modul-modul untuk perkakasan

Rekabentuk di atas adalah terdiri daripada modul kawalan pusat, pemampatan dan penyahmampatan.Bagi mengawal kedua-dua modul pemampat dan penyahmampat, modul kawalan diperlukan dan ia akan memastikan kedua-dua modul ini dapat disinkronisasikan.

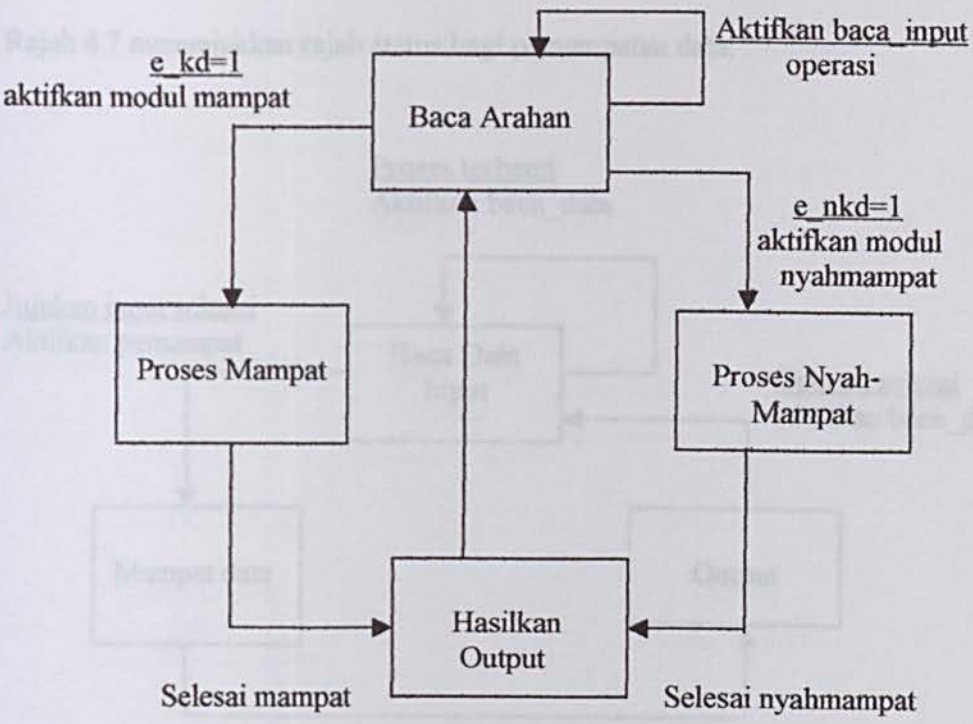
Apabila data memasuki perkakasan, kawalan pusat akan memastikan apakah jenis operasi yang harus dilakukan. Kemudian, ia akan menghantar isyarat sama ada kod pemampat atau kod penyahmampat. Sekiranya data perlu dimampat, ia akan mengeluarkan isyarat pemampatan (e_kd) kepada modul pemampat. Sebaliknya,ia akan menghantar isyarat penyahmampat (e_nkd) kepada modul penyahmampat sekiranya data perlu dinyahmampat.

4.3 Rajah status rekabentuk perkakasan

Rajah status dicipta bagi memudahkan pemahaman terhadap operasi-operasi perkakasan semasa fasa pengkodan dan pengujian. Setiap modul diteliti operasinya daripada modul utama sehingga modul yang lain.

4.3.1 Operasi keseluruhan

Rajah 4.6 menunjukkan rajah status bagi keseluruhan operasi perkakasan.

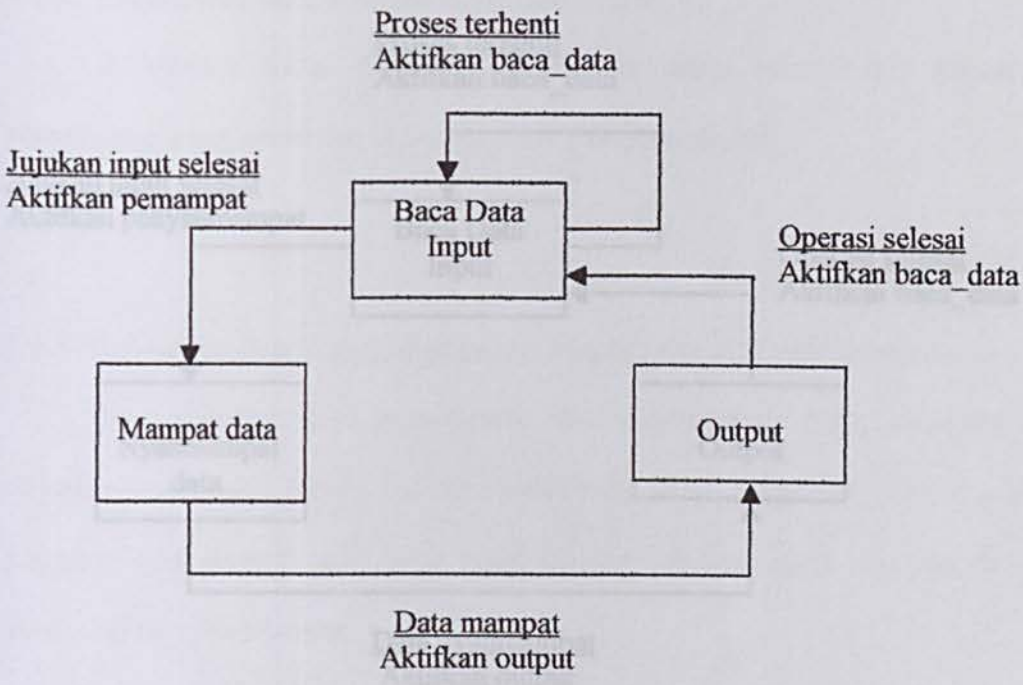


Rajah 4.6 : Rajah status keseluruhan perkakasan

Daripada rajah di atas, operasi modul dapat dilihat dengan jelas. Apabila suatu data memasuki perkakas, modul kawalan pusat akan mengaktifkan operasi baca_input. Sekiranya data perlu dimampat, arahan e_kd dikeluarkan untuk mengaktifkan modul pemampatan. Jika data perlu dinyahmampat, kawalan pusat akan mengeluarkan arahan e_nkd untuk mengaktifkan modul penyahmampatan.

4.3.2 Pemampatan data

Rajah 4.7 menunjukkan rajah status bagi pemampatan data.

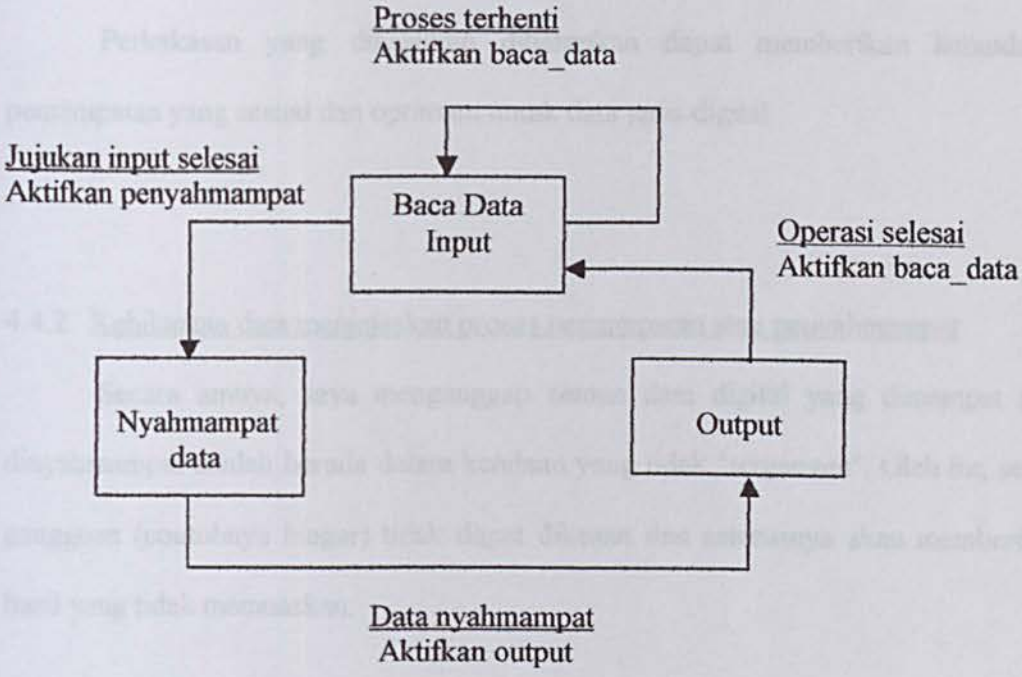


Rajah 4.7 : Rajah status pemampatan

Setelah modul kawalan pusat menghantar isyarat e_kd, modul pemampat akan bersedia untuk memampatkan data yang masuk. Arahan baca_data akan diaktifkan dan setelah itu, jujukan input akan dibaca dan dimampatkan. Kemudian, hasil mampatan akan dikeluarkan sebagai output. Baca_data akan diaktifkan semula untuk data selanjutnya.

4.3.3 Penyahmampatan data

Rajah berikut menunjukkan rajah status bagi penyahmampatan data.



Rajah 4.8 : Rajah status penyahmampatan

Setelah menerima arahan e_nkd dari modul kawalan pusat, arahan baca_data akan diaktifkan dan jujukan data yang telah dibaca akan bersedia untuk dinyahmampatkan. Setelah proses penyahmampatan selesai, output akan dihasilkan dan baca_data akan diaktifkan semula untuk data berikutnya.

4.4 Hasil yang dijangka

4.4.1 Pemampatan yang optimum untuk data digital

Perkakasan yang dihasilkan diharapkan dapat memberikan kemudahan pemampatan yang sesuai dan optimum untuk data jenis digital.

4.4.2 Kehilangan data menjejaskan proses pemampatan atau penyahmampat

Secara amnya, saya menganggap semua data digital yang dimampat atau dinyahmampat adalah berada dalam keadaan yang tidak 'terganggu'. Oleh itu, setiap gangguan (contohnya hingar) tidak dapat dikesan dan seterusnya akan memberikan hasil yang tidak memuaskan.

Bab 5

Bab 5 : Implementasi Projek

(Pembangunan dan Pengujian)

Fasa ini menunjukkan implementasi ke atas kod VHDL yang telah dibangunkan menggunakan perisian XILINX. Kod yang telah dibangunkan akan dikompil untuk mengesan sebarang ralat. Sekiranya tiada apa-apa ralat, maka fasa berikutnya adalah proses sintesis. Fasa ini adalah berkenaan pemetaan kod kepada bentuk get litar. Sekiranya terdapat ralat pada fasa ini, kod akan dibangunkan semula.

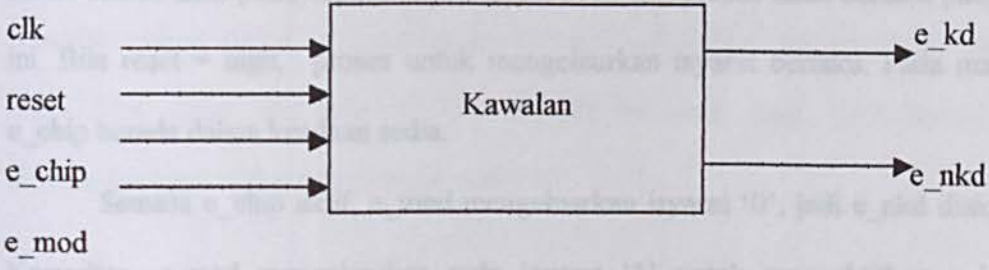
Proses simulasi, suatu pendekatan saling tak bersandar bagi setiap entiti telah dijalankan di mana setiap entiti yang wujud disimulasikan untuk mengetahui tahap ketepatan pemprosesan yang dijalankan berbanding dengan kitaran pengaturcara sendiri. Setiap keputusan simulasi akan dipaparkan.

Terdapat beberapa perubahan modul yang telah dilakukan berbanding cadangan asal. Berikut merupakan penerangan bagi semua modul yang telah dibangunkan.

5.1 Modul kawalan.vhd

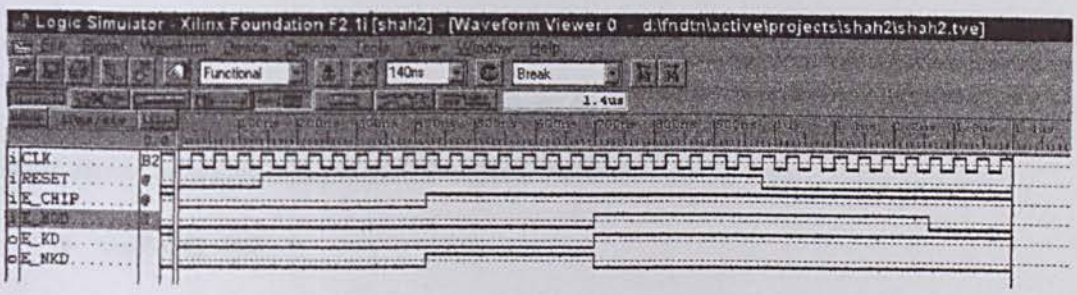
Modul ini merupakan modul yang memberikan kawalan ke atas operasi pemampatan dan penyahmampatan. Di sini, pemampatan akan diwakili oleh pengkodan (e_kd) dan penyahmampatan diwakili oleh penyahkodan (e_nkd). Modul ini akan mengeluarkan isyarat sekiranya ia berada dalam keadaan sedia, e_chip = high. Dalam modul ini, reset adalah aktif pada keadaan "low".

Sekiranya isyarat yang dihasilkan oleh e_kd ini adalah '1', maka mod pengkodan akan diaktifkan manakala jika isyarat '1' dihasilkan dari e_nkd, maka mod penyahkodan pula akan diaktifkan.



Rajah 5.1 : Modul kawalan

Berikut merupakan simulasi bagi modul kawalan2.vhd :



Rajah 5.2 : Simulasi modul kawalan

Penerangan simulasi (pengujian) :

Reset adalah aktif pada keadaan "low", jadi sebarang proses tidak berlaku pada masa ini. Bila reset = high, proses untuk mengeluarkan isyarat berlaku. Pada masa ini, e_chip berada dalam keadaan sedia.

Semasa e_chip aktif, e_mod mengeluarkan isyarat '0', jadi e_nkd diaktifkan. Kemudian, e_mod mengeluarkan pula isyarat '1' untuk mengaktifkan e_kd. Ini membuktikan bahawa modul ini berfungsi dengan baik. Diperhatikan bahawa apabila reset kembali aktif, sebarang perubahan pada e_mod tidak menghasilkan apa-apa isyarat.

5.2 Modul Pemampatan Huffman

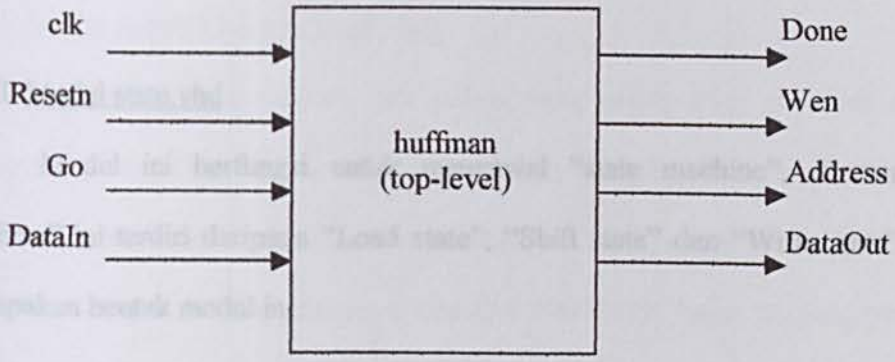
Modul vhdl huffman.vhd perihalkan algoritma untuk menukarkan data (setiap satu adalah 8-bit panjang) dari modul memori, dan mengkod data kepada jujukan 'variable-length huffman coded' yang mana ia akan disimpan ke dalam memori lain. Kod ini mengandungi empat modul seperti berikut :

- huffman.vhd
- control.vhd
- state.vhd
- huff.vhd

Keempat-empat modul ini bekerjasama untuk menghasilkan pemampatan bagi data-data 8-bit yang dimasukkan. Gambarajah simulasi bagi pemampatan akan ditunjukkan pada akhir penerangan modul ini.

5.2.1 Modul huffman.vhd

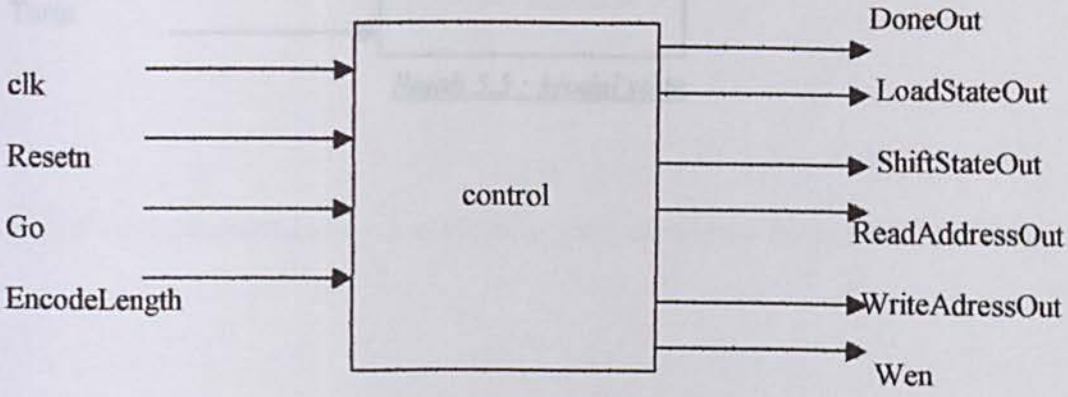
Merupakan modul 'top-level'. Mengandungi kod untuk berinteraksi (pemetaan) dengan modul 'low-level' seperti control.vhd, state.vhd dan huff.vhd. Berikut menunjukkan gambarajah untuk modul ini :



Rajah 5.3 : Modul "top-level" Huffman

5.2.2 Modul control.vhd

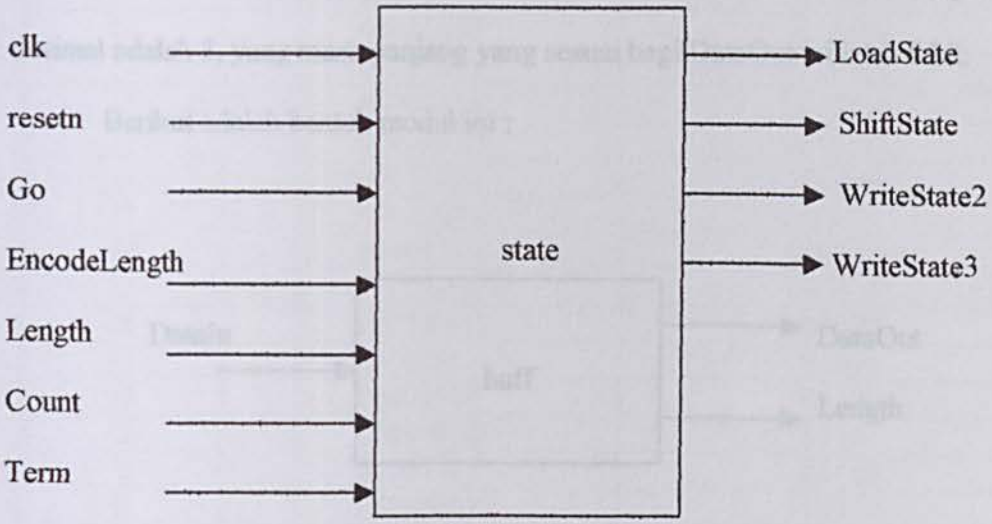
Berfungsi menjejaki alamat memori semasa membaca dan menulis data ke memori. Di bawah merupakan bentuk modul control ini :



Rajah 5.4 : Modul control

5.2.3 Modul state.vhd

Modul ini berfungsi untuk mengawal "state machine", di mana "state machine" ini terdiri daripada "Load state", "Shift state" dan "Write state". Berikut merupakan bentuk modul ini :

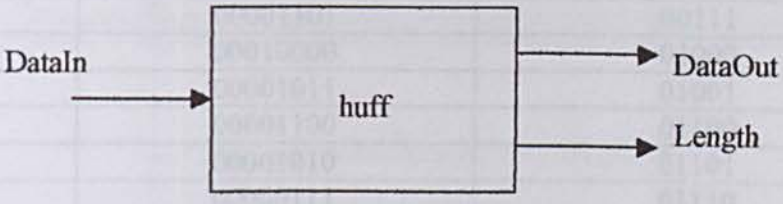


Rajah 5.5 : Modul state

5.2.4 Modul Jadual Huffman (huff.vhd)

Modul ini menyediakan satu jadual yang mempunyai masukan data 8-bit sebanyak 32 masukan. Ia terdiri dari masukan (DataIn), keluaran (DataOut) dan Length. Fungsi Length adalah untuk memberikan nilai panjang yang sesuai kepada DataOut. Sebagai contoh, sekiranya DataOut 00**011110,** maka panjang yang sesuai adalah 101, di mana data yang dikeluarkan adalah 011110. Nilai Length dalam desimal adalah 5, yang mana panjang yang sesuai bagi DataOut ini ialah 6 bit.

Berikut adalah bentuk modul ini :



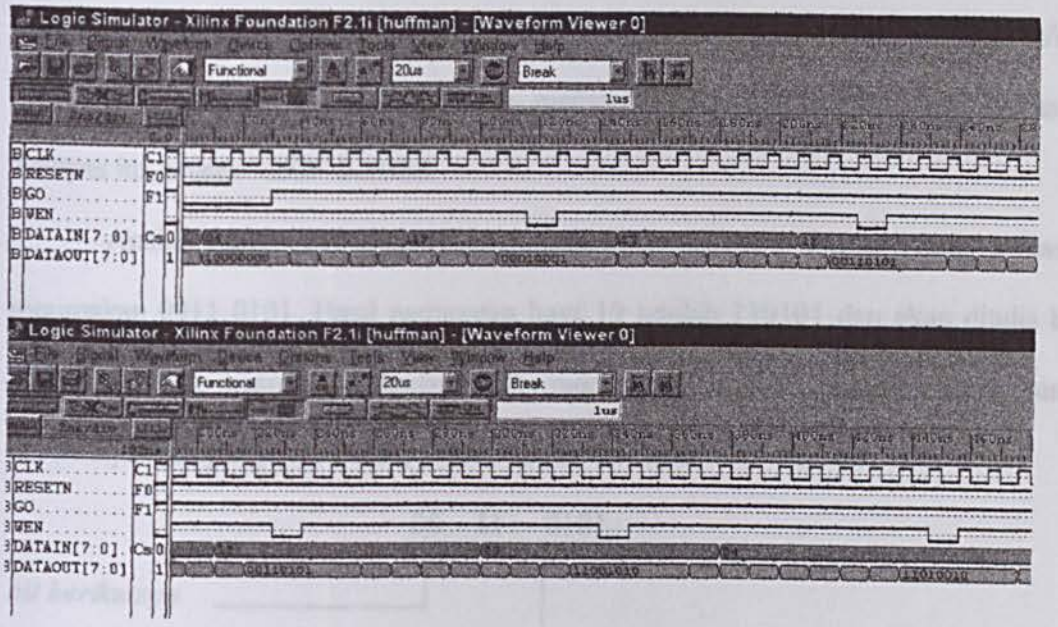
Rajah 5.6 : Modul huff

Jadual berikut menunjukkan jadual Huffman yang digunakan dalam program VHDL.

Nilai desimal	DataIn	DataOut
31	00011111	011110
30	00011110	011111
26	00011010	100110
29	00011101	100111
27	00011011	101000
23	00010111	101001
21	00010101	101010
28	00011100	101011
25	00011001	110000
20	00010100	110001
24	00011000	110100
19	00010011	110101
15	00001111	110110
22	00010110	110111
18	00010010	00000
17	00010001	00001
14	00001110	00110
13	00001101	00111
16	00010000	01000
11	00001011	01001
12	00001100	01100
10	00001010	01101
7	00000111	01110
9	00001001	10000
8	00001000	10001
6	00000110	10010
5	00000101	11001
4	00000100	0001
3	00000011	0010
2	00000010	0101
1	00000001	1011
0	00000000	111

Jadual 5.1 : Jadual kod Huffman

Berikut pula merupakan simulasi yang terhasil daripada gabungan keempat-empat modul yang membentuk modul pemampatan Huffman.



Rajah 5.7 : Simulasi pemampatan Huffman

Penerangan simulasi (pengujian) :

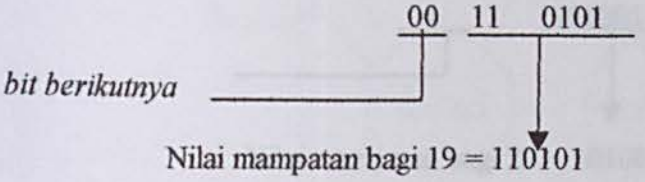
Jadual berikut merupakan nilai-nilai data yang digunakan sebagai contoh :

Data	Nilai (binari)	Nilai mampatan	Length
4	00000100	0001	011
19	00010011	110101	101
27	00011011	101000	101
12	00001100	01100	100
23	00010111	101001	101

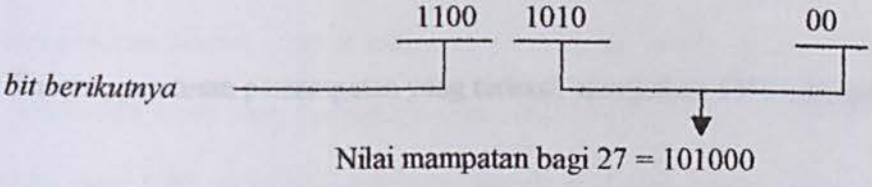
Jadual 5.2 : Jadual nilai data

Diperhatikan dari simulasi bahawa apabila nilai 4 dimasukkan, maka hasil output adalah 0001 0001. Hasil ini sepatutnya memaparkan 0000 0001 sahaja. Walau bagaimanapun, hasil mampatan bagi 4 tetap dipaparkan iaitu 0001. Di sini, Wen (write enable) akan mula menulis ke memori untuk nilai output yang terhasil. Selepas itu, proses "shift" berlaku.

Apabila nilai 19 dimasukkan, perhatikan bahawa output yang terhasil merupakan 0011 0101. Hasil mampatan bagi 19 adalah 110101 dan akan ditulis ke memori. 2 bit kosong di hadapannya merupakan 2 bit dari masukkan yang berikutnya. Selepas proses Wen, proses "shift" berlaku.

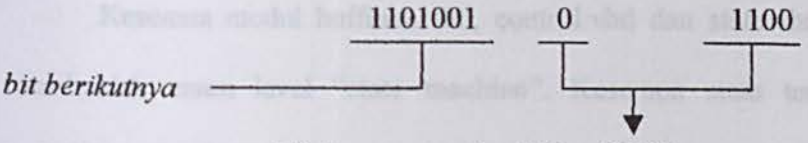


Apabila nilai 27 masuk, output yang terhasil adalah 1100 1010. Nilai mampatan bagi 27 adalah 101000, tapi di sini dipaparkan 1100 1010 kerana 2 bit 0 sebelum ini telah pun dipaparkan.



Proses "shift" berlaku setelah data ditulis ke memori.

Apabila nilai 12 dimasukkan, output yang terhasil adalah 1101 0010. Nilai mampatan bagi 12 adalah 01100, yang mana 1100 telah pun dipaparkan sebelum ini.



Nilai mampatan bagi 12 = 01100

Perhatikan bahawa nilai mampatan bagi 23 iaitu 101001 telah pun dipaparkan. Ketika ini, Wen akan berfungsi untuk menulis output ke modul memori. Jadi, yang tinggal adalah bit 1.

Nilai mampatan bagi 23 = 101001

Secara ringkasnya, apa yang terhasil adalah seperti berikut :

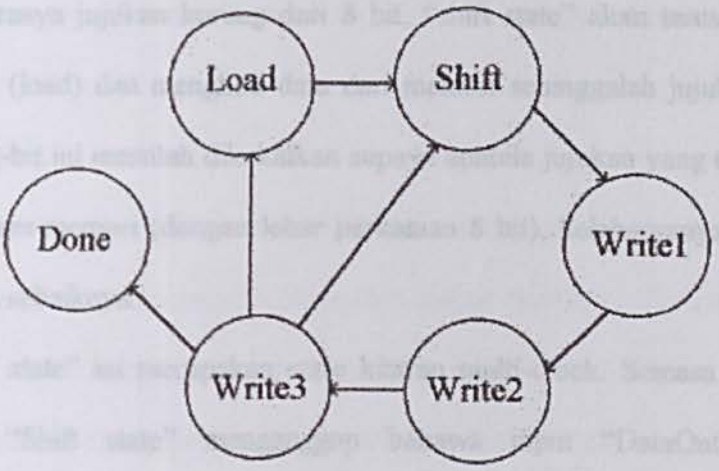
Data input = 00000100**00010011**0001101**10000110**0000010111

Hasil output = 0001**11010**1101000**1100**101001

di mana, peratusan pemampatan yang terhasil merupakan 55% sehingga 65%.

5.2.5 Penerangan mengenai operasi modul

Kesemua modul huffman.vhd, control.vhd dan state.vhd digabungkan untuk membentuk enam level "state machine". Kesemua state tersebut diillustrasikan seperti yang berikut :



Rajah 5.8 : "State machine".

"Load state"

"Load state" merupakan state yang pertama yang bermula selepas reset. Ia menjanakan alamat (output address) yang mana ia digunakan untuk untuk pindah (load) data input yang berikutnya dari modul memori luar. Semasa "Load state" ini, data input telah dipindahkan dan memasuki huff.vhd, menghasilkan 8-bit "DataOut"

dan 3-bit nilai "EncodeLength". Selepas data memasuki huff.vhd, proses akan memasuki state yang berikutnya.

"Shift state"

Di dalam state ini, sistem akan membina jujukan kod Huffman "variable-length". Sekiranya jujukan kurang dari 8 bit, "shift state" akan terus menyambung untuk pindah (load) dan mengkod data dari memori sehinggalah jujukan menjadi 8 bit. Jujukan 8-bit ini mestilah dikekalkan supaya apabila jujukan yang telah dikodkan ditulis ke dalam memori (dengan lebar perkataan 8 bit), kelebarannya (width) akan padan dengan sebaiknya.

"Shift state" ini merupakan state kitaran multi-clock. Semasa proses multi-kitaran ini, "Shift state" menganggap bahawa input "DataOut" dan input "EncodeLength" dari huff.vhd tidak akan berubah. Sebagai contoh, sekiranya nilai input adalah DataOut = 00000111 dan EncodeLength = 010, "Shift state" ini memerlukan 3 kitaran clock untuk proses data. Semasa ini data mestilah statik sebanyak 3 kitaran clock. Sekiranya keadaan ini dapat dipenuhi, sistem akan beralih kepada "Write1 state".

"Write1 state"

State ini adalah state satu clock (single clock latency state). Tugas state ini ialah menghentikan (pauses) satu kitaran (cycle) untuk membenarkan masa kepada operasi lain untuk siap. Operasi lain ini termasuklah menukar output "address" untuk menunjukkan (display) alamat memori yang betul yang mana data yang telah dikodkan ditulis. Selepas satu kitaran clock, sistem akan beralih ke state "Write2".

"Write2 state"

Dalam state ini, sistem akan "write-enable" (Wen => low) modul memori luaran 8-bit untuk alamat yang diberikan oleh output "address". Data yang ditulis ke memori adalah data jujukan pengkodan 8-bit yang dibina semasa "Shift state". Selepas satu kitaran clock, sistem akan beralih ke state "Write3".

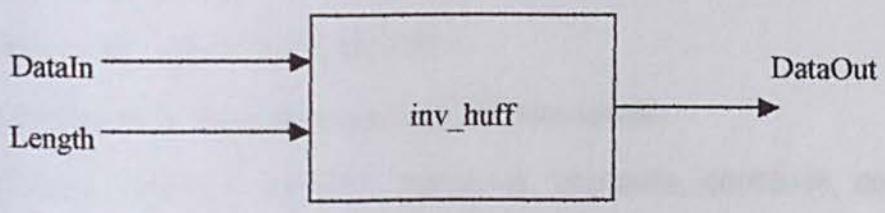
"Write3 state"

State ini membenarkan "single clock latency" supaya alamat memori sekali lagi ditukarkan (switch) kepada alamat memori data input. State ini juga menentukan sama ada sistem perlu balik ke "Load state" atau "Shift state". Jika "Shift state" menjanakan jujukan melebihi 8-bit, semua bit baki (jika 9 bakinya 1 bit) akan mewakili permulaan jujukan baru dan sistem akan pulang (return) ke "Shift state". Tetapi, jika tiada bit baki, sistem akan memerlukan data. Jadi ia akan pulang ke

"Load state". Telah dianggapkan bahawa saiz memori maksimum untuk modul memori "DataIn" adalah 128 perkataan (128 words). Untuk itu, jika alamat memori untuk data berikutnya lebih besar dari 128, maka modul akan serta-merta beralih kepada "Done state", yang mana ia akan kekal "disable" sehingga reset.

5.3 Modul Penyahmampatan

Modul pemampatan adalah sepatutnya terdiri dari empat modul yang sama dengan modul pemampatan Huffman. Cuma bezanya adalah dari segi jadual yang digunakan bagi modul ini. Namun, oleh kerana modul ini berdepan dengan pelbagai masalah ralat dan sintesis, maka apa yang dapat dipaparkan hanyalah modul jadual "inverse" Huffman. Berikut menunjukkan bentuk modul "inverse" Huffman :

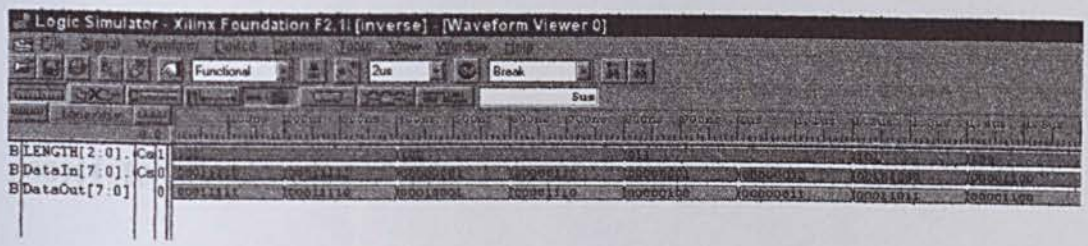


Rajah 5.9 : Modul "inv_huff"

DataIn di sini bukanlah seperti DataIn yang terdapat pada modul pemampatan Huffman kerana data yang masuk merupakan data yang telah dimampat. Input Length adalah bertujuan untuk memberikan nilai anjakan untuk menghasilkan kembali nilai data yang sebenar berdasarkan jadual yang telah dibina.

Jadi, dengan nilai input dan Length yang betul, data yang sepadan dengannya akan dikeluarkan dari modul ini, seterusnya menghasilkan DataOut.

Simulasi bagi modul ini adalah seperti yang berikut :



Rajah 5.10 : Simulasi jadual "inverse" Huffman

Penerangan simulasi (pengujian) :

Dari rajah yang diberikan, contoh data yang diambil adalah seperti berikut :

Length = 101, 100, 011, 101 dan 100

manakala DataIn yang merupakan kod Huffman adalah :

00011110, 00011111, 0000001, 00000110, 00000001, 00000010, 00101000 dan 00001100.

Kesemua input ini menghasilkan semula nilai asal seperti :

00011111, 00011110, 00010001, 00001110, 00000100, 00000011, 00011011 dan 00001100.

DataIn sepatutnya 'dipendekkan' dengan Length yang diberikan untuk menghasilkan DataOut yang mempunyai 8-bit. Jadi, hasil yang sepatutnya adalah :

DataIn :

0111100**1111**10000**1000**1100001**001**0101000**01100**

DataOut :

000111110001111**00001000**1000011**10000001**0000000011**00011011**00001100.

Gambarajah bagi gabungan modul ini boleh dilihat di Lampiran.

Bab 6

Bab 6 : Masalah dan perbincangan

6.1 Masalah-masalah yang wujud

Perancangan rapi yang telah dibuat sebelum ini sepatutnya tidak menghadapi masalah. Namun, sebagai manusia biasa, kita memang tidak dapat lari dari bertemu dengan masalah. Semasa pembangunan projek dilakukan, beberapa masalah telah dikesan. Masalah tersebut adalah termasuk rekabentuk modul yang tidak sesuai dan beberapa lagi masalah lain seperti penggunaan perisian pembangunan. Walaubagaimanapun, sebilangan masalah dapat diatasi. Berikut adalah perbincangan mengenai masalah-masalah yang dihadapi.

6.1.1 Masalah dari segi rekabentuk perkakasan

Perubahan semua modul

Semua modul mengalami perubahan sesuai dengan tahap kesukaran yang dialami semasa proses pembangunan kod. Modul kawalan ditambah dengan beberapa input lagi bagi memastikan ia boleh berjalan lancar.

Bagi modul pemampatan pula, ia terdiri dari 4 modul iaitu huffman, control, state dan huff (jadual kod Huffman).

Penggunaan jadual Huffman.

Mengikut cadangan asal, modul Huffman yang ingin diimplementasi adalah dari jenis yang boleh mengira frekuensi data yang dimasukkan, seterusnya akan menghasilkan satu output data yang termampat. Namun, dengan perisian Xilinx ini, proses pengiraan frekuensi tidak dapat dilakukan kerana ia tidak menyokong proses input-output yang dilakukan secara dinamik. Ini secara tidak langsung menjejaskan langkah pengiraan frekuensi data input, yang mana ia merupakan langkah penting bagi proses pemampatan Huffman.

Sebagai penyelesaian bagi masalah ini, saya telah mereka satu jadual data input dan output yang telah dilakukan secara manual. Ini bermakna jadual ini telah menyediakan data termampat melalui proses Huffman yang dilakukan secara manual dan diimplemenkan ke dalam rekabentuk modul pemampatan melalui modul Huff.vhd. Proses untuk mengubah modul ini telah mengambil masa yang agak lama memandangkan perubahan ini memerlukan idea.

Penggunaan jadual telah memberikan kelebihan dari segi kepantasan kerana ia tidak perlu melalui proses pengiraan frekuensi (yang mana ia telah dilakukan secara manual). Namun, kelemahan bagi jadual ini ialah nilainya telah ditetapkan dan ini adalah kurang efisyen untuk sebarang kemasukan data yang baru.

Pembangunan modul penyahmampatan

Modul penyahmampatan merupakan satu-satunya modul yang tidak dapat dibangunkan sepenuhnya. Sebab utamanya ialah modul ini menghadapi masalah untuk disintesiskan. Saya telah menghadapi banyak "error" semasa membuat program untuk modul ini. Disebabkan masa yang terhad, apa yang dapat saya lakukan adalah dengan membuat satu jadual penyahmampatan dan membuat proses input-output dari jadual tersebut. Ini adalah sekadar untuk menerangkan apakah sepatutnya yang berlaku semasa proses penyahmampatan, di mana data input (frekuensi) yang dimasukkan akan menghasilkan data output yang asal (nilai data asal).

6.1.2 Masalah-masalah lain

Ingatan dan kuasa pemproses

Semasa pembangunan projek dilaksanakan, disedari bahawa perlaksanaan perisian Xilinx ini memerlukan saiz ingatan (RAM) yang besar. Selain itu, kuasa pemproses yang ada adalah kurang efisyen untuk kelajuan. Oleh kerana saiz RAM yang kecil dan pemproses yang digunakan adalah Intel Pentium II, ini telah menyebabkan program yang ingin disintesis memakan masa yang agak lama.

Masalah ini diatasi dengan berjumpa dengan pihak yang berkenaan untuk meminta penambahan RAM, memandangkan penukaran kuasa pemprosesan adalah mustahil untuk dilakukan buat masa terdekat ini.

Cadangan saya bagi mengatasi masalah ini untuk jangkamasa panjang adalah pihak fakulti perlulah menukar pemproses yang ada sekarang kepada yang lebih berkuasa dan penambahan RAM juga perlu dilakukan agar sebarang projek yang melibatkan perisian ini dapat dijalankan dengan lancar.

Masalah perisian Xilinx

Perisian Xilinx yang disediakan oleh pihak fakulti ini hanya boleh digunakan pada komputer di makmal VLSI sahaja. Ia merupakan satu perisian yang berdaftar dan tidak boleh di'install'kan pada komputer yang lain. Oleh itu, perlaksanaan projek adalah bergantung penuh terhadap penggunaan komputer di makmal. Dengan penggunaan masa yang terlalu terhad, ditambah lagi dengan masa yang diperuntukkan singkat, adalah sukar bagi saya untuk membangunkan projek terutamanya yang melibatkan teori algoritma.

Kekangan bahan rujukan

Walaupun algortima pemampatan Huffman merupakan algoritma yang popular dikalangan pengaturcara, namun ia tidaklah begitu popular untuk diimplementasikan ke dalam suatu perkakasan. Walaupun konsep algoritmanya agak mudah, namun perlaksanaannya dalam pembangunan perkakasan adalah suatu perkara yang sukar.

Semasa proses pembangunan projek dilaksanakan, tidak banyak bahan rujukan berkaitan perkakasan pemampatan yang boleh dirujuk atau dikaji. Walaupun dalam Internet terdapat banyak kertas kerja berkaitan Huffman boleh dirujuk, tetapi kebanyakannya adalah mengenai pembangunan perisian dan bukannya untuk pembangunan perkakasan. Sedikit sebanyak ia menjejaskan masa yang diambil untuk membangunkan projek ini.

6.2 Ciri-ciri yang boleh diperbaiki

Sekiranya projek ini ingin diteruskan pada masa akan datang, saya mendapati terdapat beberapa ciri yang boleh ditambah untuk memperbaiki perkakasan ini. Ciri-ciri tersebut adalah seperti perbincangan berikut.

6.2.1 Melaksanakan pengiraan frekuensi bagi algoritma Huffman

Dalam pembangunan perkakasan ini, saya tidak dapat melaksanakan pengiraan frekuensi bagi algoritma Huffman. Oleh itu, bagi projek yang akan datang, adalah dicadangkan supaya proses pengiraan frekuensi bagi algoritma Huffman dapat dilakukan di dalam perkakasan. Walaupun perlaksanaan ini akan mengakibatkan proses pemampatan menjadi lambat, namun yang penting ialah ia amat sesuai dan efisyen bagi penggunaan untuk semua nilai data input. Ini seterusnya akan meningkatkan lagi kebolehpercayaan (reliability) terhadap penggunaan perkakasan.

6.2.2 Penambahan modul pengesanan ralat

Oleh kerana perkakasan yang direka ini tidak menyokong proses pengesanan ralat, jadi saya cadangkan agar pada masa akan datang, modul pengesanan ralat bolehlah ditambah kepada modul yang sedia ada. Alasan kenapa ia perlu ditambah adalah untuk menyokong proses penghantaran data dari satu lokasi ke lokasi yang lain (melalui rangkaian).

Seperti yang diketahui umum, penghantaran data antara dua lokasi berlainan akan menghasilkan bit ralat. Walaupun pemampatan telah dilakukan, data yang telah dimampat tidak terelak dari bencana ralat. Oleh itu, sekiranya ia dinyahmampat, ia

akan menghasilkan data yang lari daipada data sebelum dimampat. Dengan adanya modul pengesanan ralat, Insya-Allah masalah ini akan dapat diatasi.

Bab 7

Bab 7 : Kesimpulan

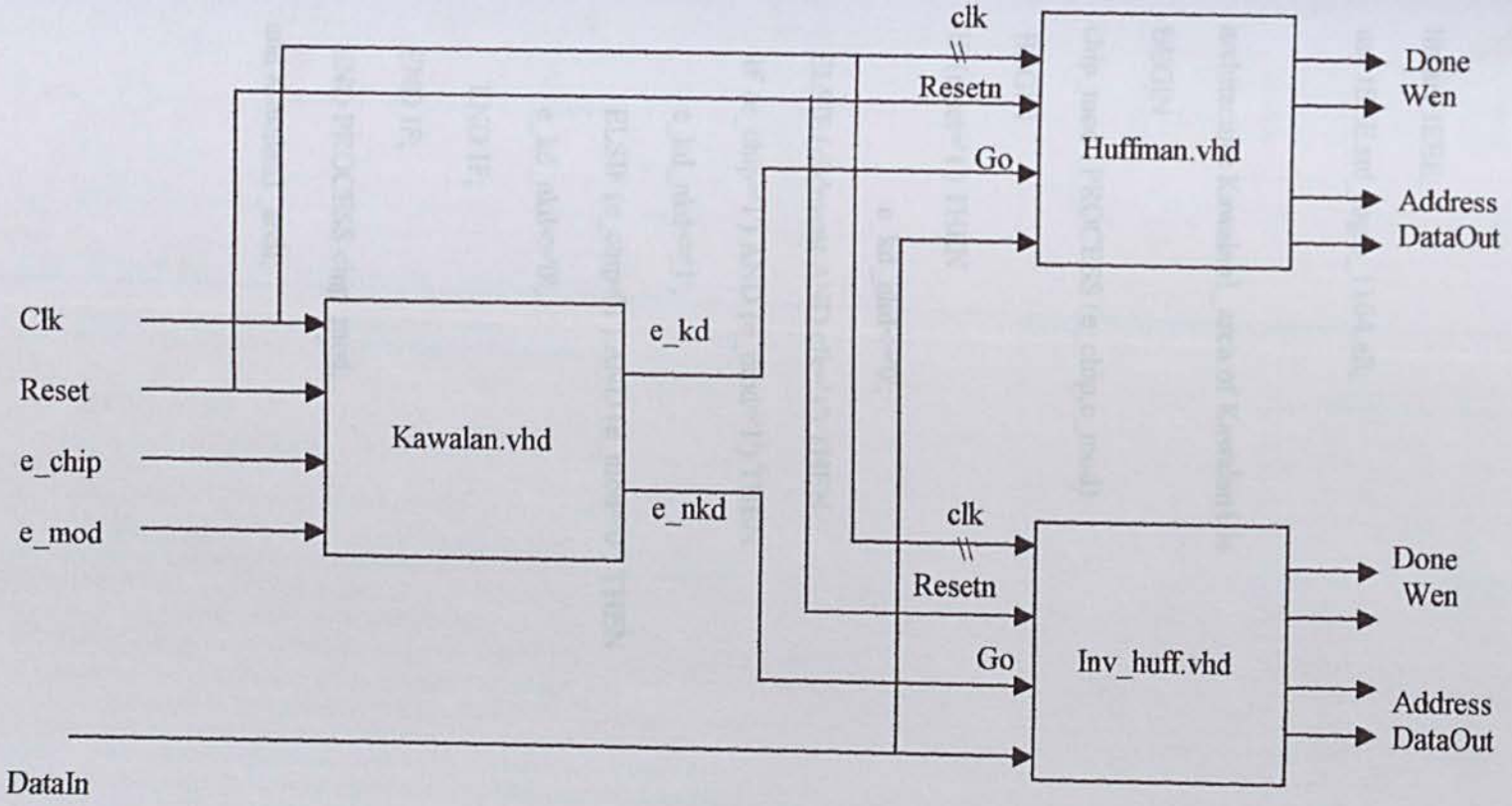
Saya rasa bersyukur kerana walaupun projek ini tidak dapat disiapkan sepenuhnya, namun ia tetap dapat mencapai objektifnya. Meskipun pemampatan hanya dapat dilakukan bagi sebilangan data, namun idea pembangunannya boleh dikembangkan lagi untuk digunakan pada kumpulan data yang lebih besar.

Pembangunan kod sumber bagi perkakasan merupakan suatu perkara yang kompleks berbanding pembangunan kod untuk perisian. Segala perancangan yang pada awalnya dianggap agak 'mudah', namun setelah melalui sendiri proses pembangunannya, ternyata ia amat sukar bagi saya yang tidak berlatarkan pengetahuan tentang pembangunan perkakasan. Walaupun kod sumber yang telah kita bangunkan telah betul dan dapat menghasilkan keluaran yang sepatutnya, namun kadangkala ia tidak berlaku seperti itu. Ini berlaku kerana kadangkala keadaan litar dalam perkakasan boleh memberi kesan kepada data output yang dihasilkan.

Secara keseluruhannya, projek Rekabentuk Cip Pemampatan Data ini telah memberikan saya satu ilmu dan pengalaman yang sangat berguna. Saya telah berpeluang mempelajari fasa pembangunan, pengkodan dan segala selok-belok tentang perkakasan yang saya tidak pernah tahu sebelum ini. Sekurang-kurangnya saya dapat mengaplikasikan pengetahuan pengaturcaraan dan elektronik yang saya pelajari sebelum ini.

Lampiran

Rekabentuk Cip Pemampatan Data

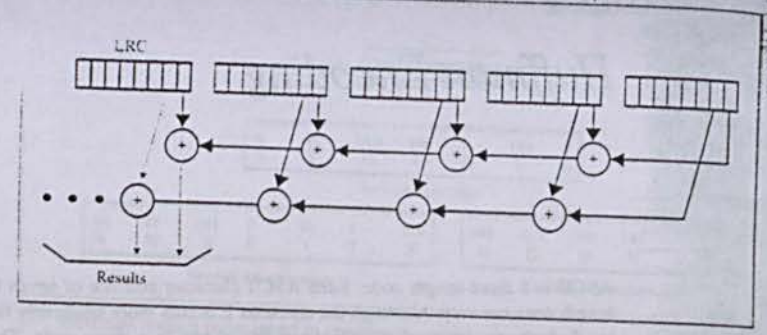


```
-- kod untuk modul kawalan--

library IEEE;
use IEEE.std_logic_1164.all;

architecture Kawalan1_arch of Kawalan1 is
BEGIN
chip_mod : PROCESS (e_chip,e_mod)
 BEGIN
 IF (reset='1') THEN
                    e_kd_nkd<='0';
   ELSIF (clk'event AND clk='1') THEN
    IF (e_chip='1') AND (e_mod='1') THEN
        e_kd_nkd<='1';
        ELSIF (e_chip='1') AND (e_mod='0') THEN
        e_kd_nkd<='0';
      END IF;
  END IF;
  END PROCESS chip_mod;
end Kawalan1_arch;
```

Figure E.7 *LRC checker*



E.4 CYCLIC REDUNDANCY CHECK (CRC)

As we learned in Chapter 9, the cyclic redundancy check (CRC) is more efficient than VRC or LRC. In this section, we show how to make a CRC generator and checker.

The CRC Generator

Designing a CRC generator from a given polynomial is easily done following these steps:

- Change the polynomial to a divisor of size $N + 1$. (N is the order of the polynomial.)
- Make a shift register of size N.
- Align the shift register cells with the divisor so that the cells are located between the bits.
- Put an XOR where there is a 1 in the divisor except for the leftmost bit.
- Make a feedback connection from the leftmost bit to the XORs.
- Add a switch to direct the data and the generated CRC output.

Figure E.8 shows the CRC generator derived from the ITU-T polynomial.

One set of bits moves through the CRC generator to the switch; the other is sent directly to the switch. The switch uses a counter to send the data first and then the remainder (CRC). Figure E.9 shows the generation of the CRC remainder. In each line, the XOR gates add two bits together. After this operation, all bits are then shifted one position to the left. The last line shows the CRC remainder in the shift register.

The CRC Checker

The CRC hardware at the receiving end of the transmission works in precisely the same way except that it tests the whole package, including the data and the CRC, to determine the accuracy of the received data (see Figure E.10).

Figure E.8 *From polynomial to CRC generator*

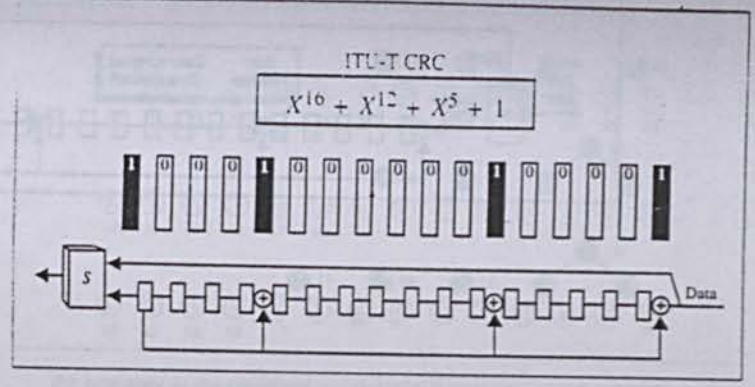


Figure E.9 *An example of a CRC generator*

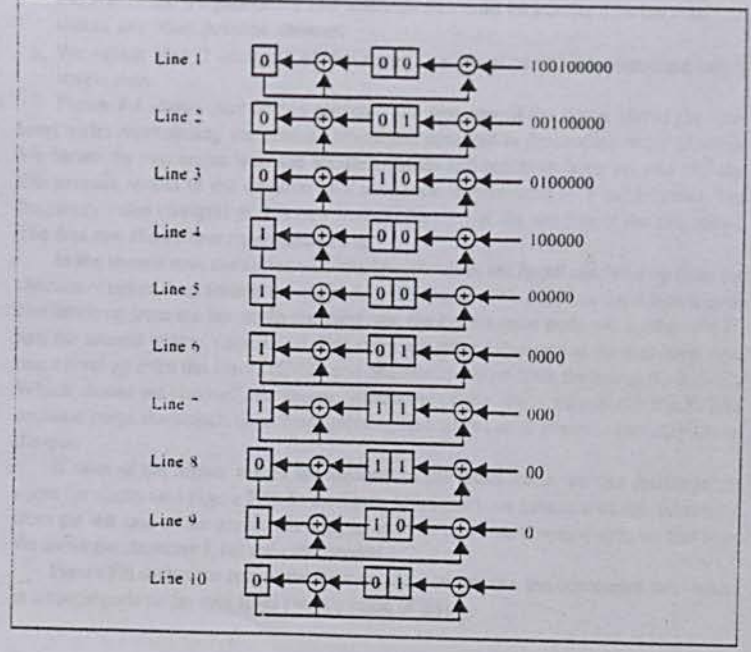
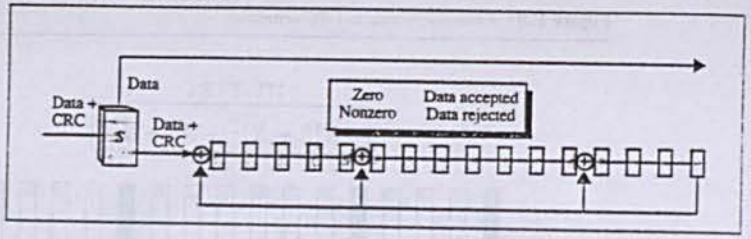


Figure E.10 *CRC checker*



APPENDIX F

Huffman Encoding

ASCII is a fixed-length code. Each ASCII character consists of seven bits. Character length does not vary. Although the character E occurs more frequently than the character Z, both are assigned the same number of bits in a given code. This consistency means that every character uses the maximum number of bits, resulting in lengthy encoded messages.

Huffman coding, however, makes coding more efficient. In this mechanism, we assign shorter codes to characters that occur more frequently and longer codes to those that occur less frequently. For example, E and T, the two characters that occur most frequently in the English language, are assigned one bit each. A, I, M, and N, which also occur frequently but less frequently than E and T, are assigned two bits each. C, G, K, R, S, U, and W are the next most frequent and are assigned three bits each, and so on. In a given piece of text, only some of the characters will require the maximum bit length. The overall length of the transmission, therefore, is shorter than that resulting from fixed-length encoding.

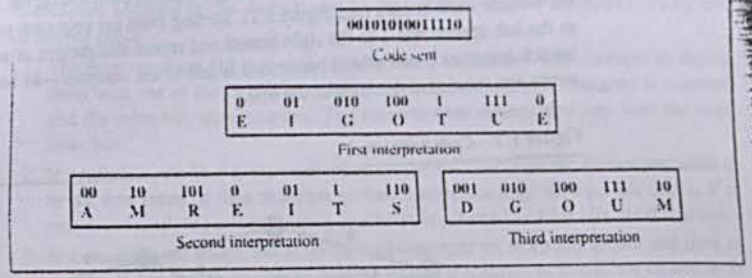
Difficulty arises, however, if the bit patterns associated with each character are assigned randomly. Consider the example in Figure F.1. Note that we have purposely limited the number of characters in the example to only a few from the complete alphanumeric and special character set in order to make the demonstration easier to follow.

Figure F.1 *Bit assignments based on frequency of the character*

E:0	T:1						
A:00	I:01	M:10	N:11				
C:000	D:001	G:010	K:011	O:100	R:101	S:110	U:111

As you can see, each character is represented by a unique bit pattern and is easily distinguishable when presented separately. But what happens when these characters are formed into a data stream? Figure F.2 shows the possible results. Without a predictable character bit length, the receiver may misinterpret the code.

Figure F.2 *Multiple interpretations of transmitted data*



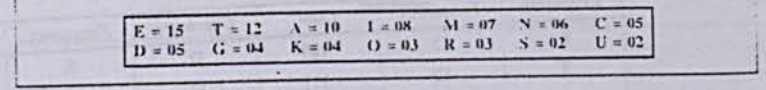
Huffman coding is designed to counter this ambiguity while retaining the bit count advantages of a compression code. Not only does it vary the length of the code based on the frequency of the character represented, but each character code is chosen in such a way that no code is the prefix of another code. For example, no three-bit code has the same pattern as the first three bits of a four- or five-bit code (prefix property code).

F.1 CHARACTER TREE

Using the character set from the example above, let's examine how a Huffman code is built.

Before we can assign bit patterns to each character, we assign each character a weight based on its frequency of use. In our example, we assume that the frequency of the character E in a text is 15 percent, the frequency of the character T is 12 percent, and so on (see Figure F.3).

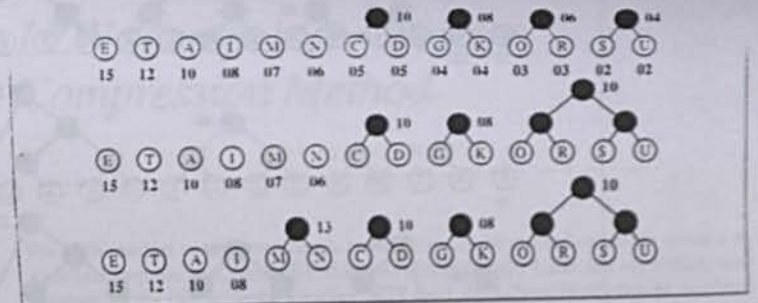
Figure F.3 *Character weights*



Once we have established the weight of each character, we build a tree based on those values. The process for building this tree is shown in Figure F.4. It follows two basic steps:

1. First we organize the entire character set into a row, ordered according to frequency from highest to lowest (or vice versa). Each character is now a node at the leaf level of a tree.
2. Next, we find the two nodes with the smallest combined frequency weightings and join them to form a third node, resulting in a simple two-level tree. The weight of

Figure F.4 *Huffman tree, part 1*



the new node is the combined weights of the original two nodes. This node, one level up from the leaves, is eligible to be combined with other nodes. Remember, the sum of the weights of the two nodes chosen must be smaller than the combination of any other possible choices.

3. We repeat step 2 until all of the nodes, on every level, are combined into a single tree.

Figure F.4 shows part of this process. The first row of the figure shows the leaf-level nodes representing the original characters arranged in descending order of value. We locate the two nodes with the smallest values and combine them. As you can see, this process results in the creation of a new node (represented by a solid circle). The frequency value (weight) of this new node is the sum of the weights of the two nodes. The first row shows four combined nodes.

In the second row, the nodes with the lowest values are found one level up from the characters rather than among the characters themselves. We combine them into a node two levels up from the leaves. In the third row, the lowest value node has a value of 8 (I) and the second lowest value is 10. But there are three 10s—one at the leaf level (A), one a level up from the leaves (C-D), and one two levels up from the leaves (O-R-S-U). Which should we choose? We choose whichever of the 10s is adjacent to the 8. This decision keeps the branch lines from crossing and allows us to preserve the legibility of the tree.

If none of the higher values are adjacent to the lower value, we can rearrange the nodes for clarity (see Figure F.5). In the figure (third row), we have moved the character T from the left side of the tree to the right in order to combine it with a node on that side. We move the character E for the same reason.

Figure F.6 shows the rest of the process. As you can see, the completed tree results in a single node at the root level (with a value of 86).

Figure F.5 *Huffman tree, part 2*

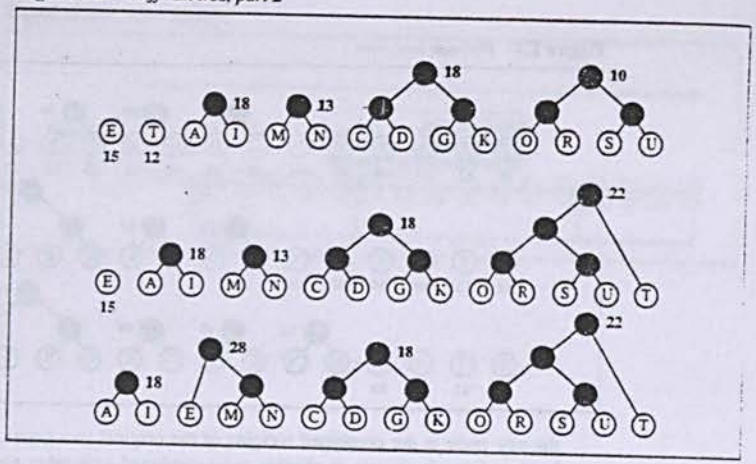
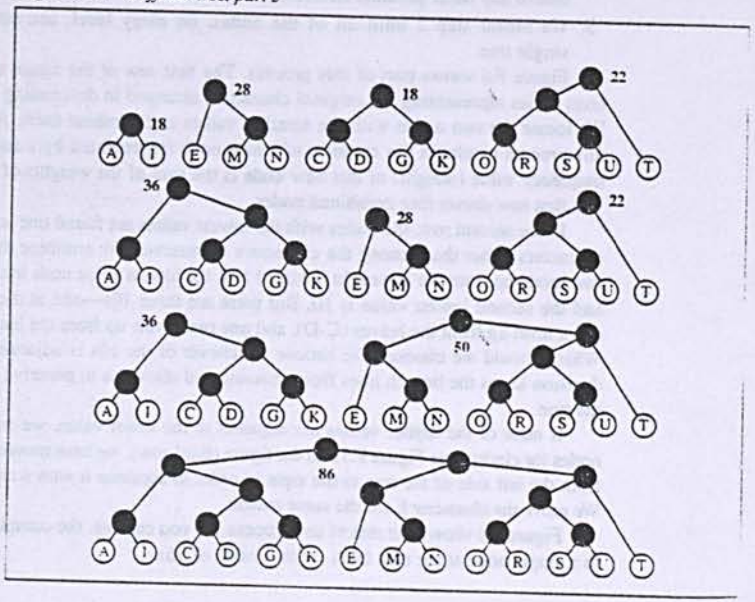


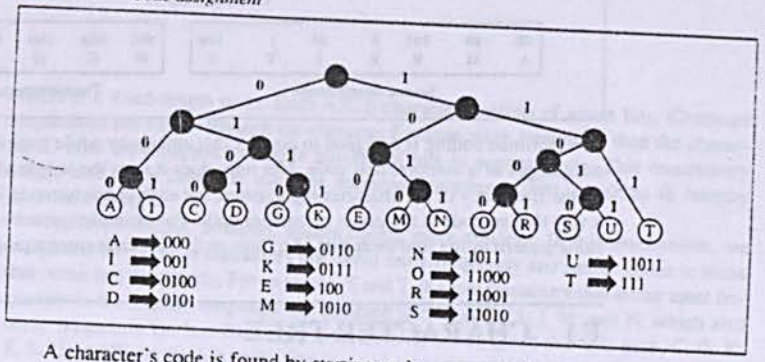
Figure F.6 *Huffman tree, part 3*



F.2 ASSIGNING THE CODES

Once the tree is complete, we use it to assign codes to each character. First, we assign a bit value to each branch (see Figure F.7). Starting from the root (top node), we assign 0 to the left branch and 1 to the right branch and repeat this pattern at each node. Which branch becomes 0 and which becomes 1 is left to the designer—as long as the assignments are consistent throughout the tree.

Figure F.7 *Code assignment*



A character's code is found by starting at the root and following the branches that lead to that character. The code itself is the bit value of each branch on the path taken in sequence. In our example, for instance, A = 000, G = 0110, and so on. The code for each character and the frequency of the character are shown in Table F.1. As you can see, no code is the prefix of any other code because each has been obtained by following a different path from the root. The three-bit codes representing the characters E, T, A, and I do not match the first three bits of any four- or five-bit code, and the four-bit codes do not match the first three bits of any five-bit code.

Table F.1 *Code assignment table*

Character	Frequency	Code	Character	Frequency	Code
E	15	100	D	5	0101
T	12	111	G	4	0110
A	10	000	K	4	0111
I	8	001	O	3	11000
M	7	1010	R	3	11001
N	6	1011	S	2	11010
C	5	0100	U	2	11011

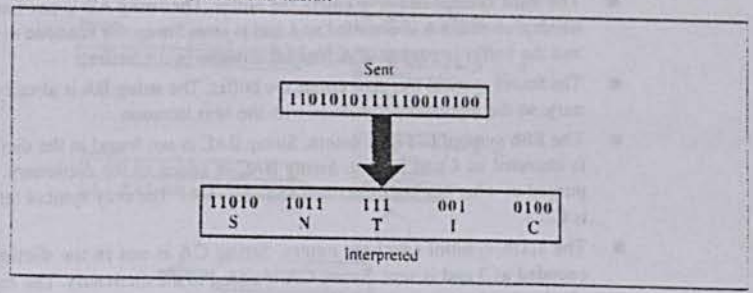
F.3 DECODING

A message encoded in this fashion can be interpreted without ambiguity, using the following process:

1. The receiver stores the first three bits received in memory and attempts to match them with one of the three-bit codes. If a match is found, that character is selected and the three bits are discarded. The receiver then repeats this step with the next three bits.
2. If a match is not found, the receiver reads the next bit from the stream and adds it to the first three. It then attempts to find a match among the four-bit codes. If a match is found, the corresponding character is selected and the bits are discarded.
3. If a match is not found, the receiver reads the next bit from the stream and tries to match all five bits to one of the five-bit codes. If a match is found, the character is assigned and the bits are discarded. If not, an error is issued.

Figure F.8 shows a series of bits and their interpretation by the receiver. The receiver reads the first three bits (110) and looks for a match among the three-bit codes. Not finding a match, it adds the next bit (1). It now tries to match the sequence 1101 with a four-bit code. Again finding no match, it adds the next bit (0) and compares 11010 to the five-bit codes. 11010 is found to represent S. S is selected, those five bits are discarded, and the process starts again with the next three bits (101).

Figure F.8 *Unambiguous transmission*



APPENDIX G

LZW (Lempel-Ziv-Welch) Compression Method

The LZW method to compress data is an evolution of the method originally created by Abraham Lempel and Jacob Ziv (Lempel-Ziv method). Later the algorithm was improved by the two originators (called LZ77 and LZ78). Terry Welch further modified the algorithm and it is now called the LZW (Lempel-Ziv-Welch) method.

We present here a very simplified version as an introduction for students and readers encountering it for the first time. We have deliberately omitted a lot of details and error-checking subtleties.

The beauty of the method is that it is a lossless compression method, in which the whole text can be totally recovered. The method is sometimes preferable to Huffman encoding because prior knowledge of the symbol frequencies is not required.

G.1 COMPRESSION

The compression, which takes place at the sender site, has the following components: a dictionary, a buffer, and an algorithm.

Dictionary

To compress and decompress, LZW uses a dictionary made of two rows (or columns). The first row defines the code; the second row defines the string corresponding to that code.

Before compression begins, the dictionary is initialized with only the set of alphabet characters in the text. For example, if the text consists of only three symbols, A, B, and C, the dictionary originally has only 3 columns, but it becomes larger and larger as the text is being compressed or decompressed (see Table G.1).

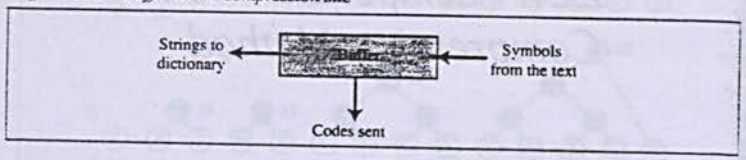
Table G.1 *Original dictionary for a three-symbol text*

1	2	3				
A	B	C				

Buffer

LZW uses a buffer. The symbols enter the buffer from one side one by one. The strings are transferred to the dictionary and the corresponding code is sent out according to the algorithm discussed below. Figure G.1 shows the buffer.

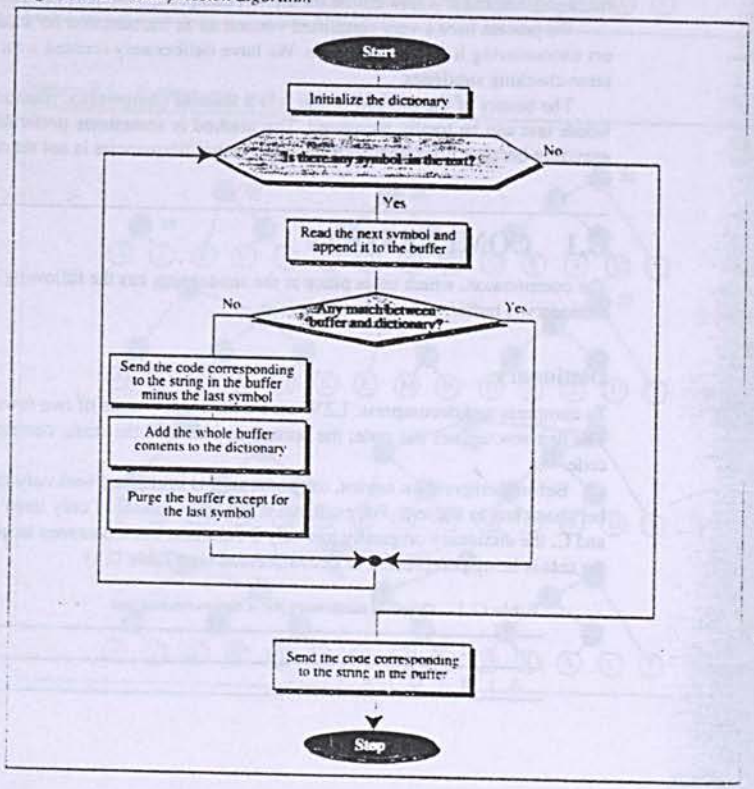
Figure G.1 *Buffer at the compression site*



Compression Algorithm

Figure G.2 shows the flowchart for the compression algorithm.

Figure G.2 *Compression algorithm*



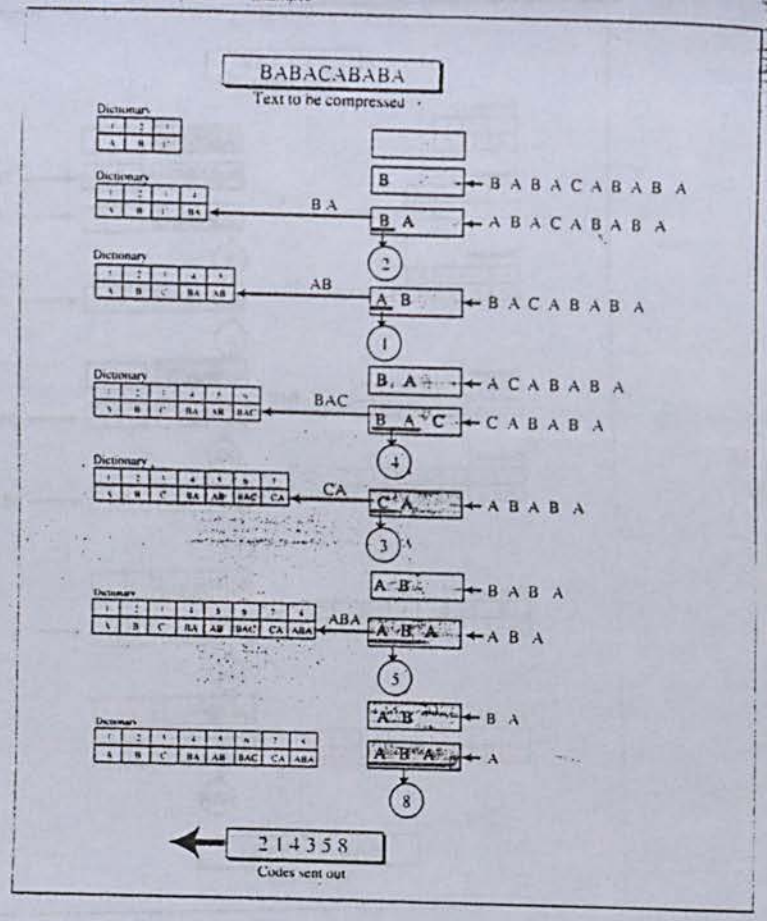
The algorithm, which is a very simplified version of the one in the literature, tries to match the input to the longest string in the dictionary. To accomplish this, it reads symbols, one by one, from the input string and stores them in the buffer. For each reading, it checks to see if the matching string can be found in the dictionary. If found, it tries to read more symbols. If after reading one more symbol there is no match, the process sends the code corresponding to the string without the last symbol (it is guaranteed that it can be found in the dictionary). It then adds the whole string to the dictionary (for later use). It purges the buffer except for the last symbol because the last symbol is not sent out yet. The last character remains in the buffer to become part of the next string.

Example of Compression

Figure G.3 shows an example of compression using the LZW method. The input text, BABACABABA, is compressed to 214358 as follows:

- The dictionary is initialized to the three symbols (A, B, and C in this example). The buffer is empty.
- The first symbol of the text (B) enters the buffer. This symbol is already in the dictionary, so the algorithm continues with the next iteration.
- The second symbol (A) enters the buffer. The string BA is not found in the dictionary, so string B (the whole string without the last) is encoded as 2 and is sent. String (BA) is added to the dictionary and the buffer is purged of B (only A remains in the buffer).
- The third symbol (B) now enters the buffer. The string AB is not found in the dictionary, so string A is encoded as 1 and is sent. String AB is added to the dictionary and the buffer is purged of A (only B remains in the buffer).
- The fourth symbol (A) now enters the buffer. The string BA is already in the dictionary, so the algorithm continues with the next iteration.
- The fifth symbol (C) now enters. String BAC is not found in the dictionary, so BA is encoded as 4 and is sent. String BAC is added to the dictionary. The buffer is purged of what has been encoded and sent (BA). The only symbol left in the buffer is C.
- The sixth symbol (A) now enters. String CA is not in the dictionary, so C is encoded as 3 and is sent. String CA is added to the dictionary. The buffer is purged of what has been encoded and sent (C). The only symbol left in the buffer is A.
- The seventh symbol (B) now enters. String AB is in the dictionary.
- The eighth symbol (A) now enters. String ABA is not found in the dictionary, so AB is encoded as 5 and is sent. String ABA, is added to the dictionary. The buffer is purged of what has been encoded and sent (AB). The only symbol left in the buffer is A.
- The ninth symbol (B) now enters. String AB is in the dictionary.
- The tenth symbol (A) now enters. String ABA is in the dictionary, so the algorithm wants to start the next iteration but there are no more symbols. The algorithm terminates the loop and sends the code corresponding to the string ABA (code 8).

Figure G.3 *Compression example*



G.2 DECOMPRESSION

The decompression process, which takes place at the receiver site, uses the same components as the compression process.

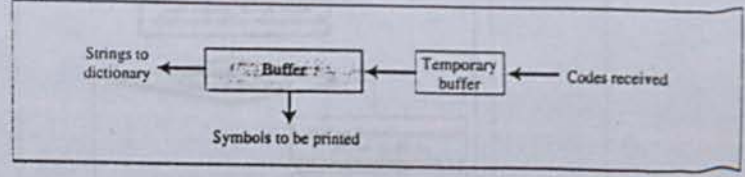
Dictionary

A very interesting point is that the sender does not send the dictionary created by the compression process; instead, the dictionary will be created at the receiver site and, surprisingly, it is the exact replica of the dictionary created at the sender site. In other words, the information in the dictionary is somehow embedded in the codes transmitted.

Buffers

The decompression process uses two buffers. The arriving codes are decoded and the resulting symbols are kept in a temporary buffer before entering, one symbol at a time, the main buffer. Since each code may represent more than one symbol, the temporary buffer is needed to hold symbols before individual consumption by the main buffer (see Figure G.4).

Figure G.4 *Buffers at the decompression site*



Decompression Algorithm

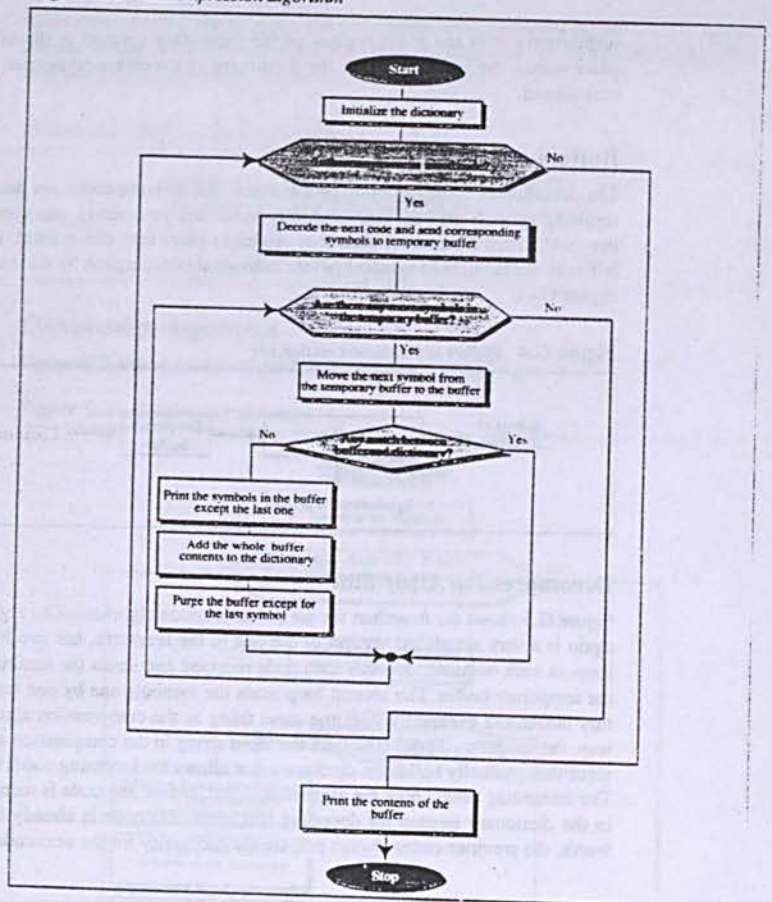
Figure G.5 shows the flowchart for the decompression algorithm. Our algorithm, which again is a very simplified version of the one in the literature, has two loops. The first loop, in each iteration, decodes each code received and sends the resulting symbols to the temporary buffer. The second loop reads the symbols one by one from the temporary buffer and eventually does the same thing as the compression algorithm. In this way, the temporary buffer simulates the input string in the compression algorithm. The algorithm gradually builds the dictionary that allows the incoming codes to be decoded. The interesting point about the algorithm is that, before any code is received, the entry in the dictionary needed for decoding that particular code is already there. In other words, the previous codes always prepare the dictionary for the next code.

Decompression Example

Figure G.6 shows an example of decompression. We use the same code sent by the sender in the compression example and see if we can get the original text without loss. The process is almost the same as compression.

- The dictionary is initialized with the three symbols that we are using in this example. The buffer and the temporary buffer are empty.
- The first code (2) is decoded and the corresponding symbol (B) enters the temporary buffer. The buffer reads this symbol. The string is in the dictionary.
- The second code (1) is decoded and the corresponding symbol (A) enters the temporary buffer. The buffer reads it. Now the string BA is in the buffer: the symbol B

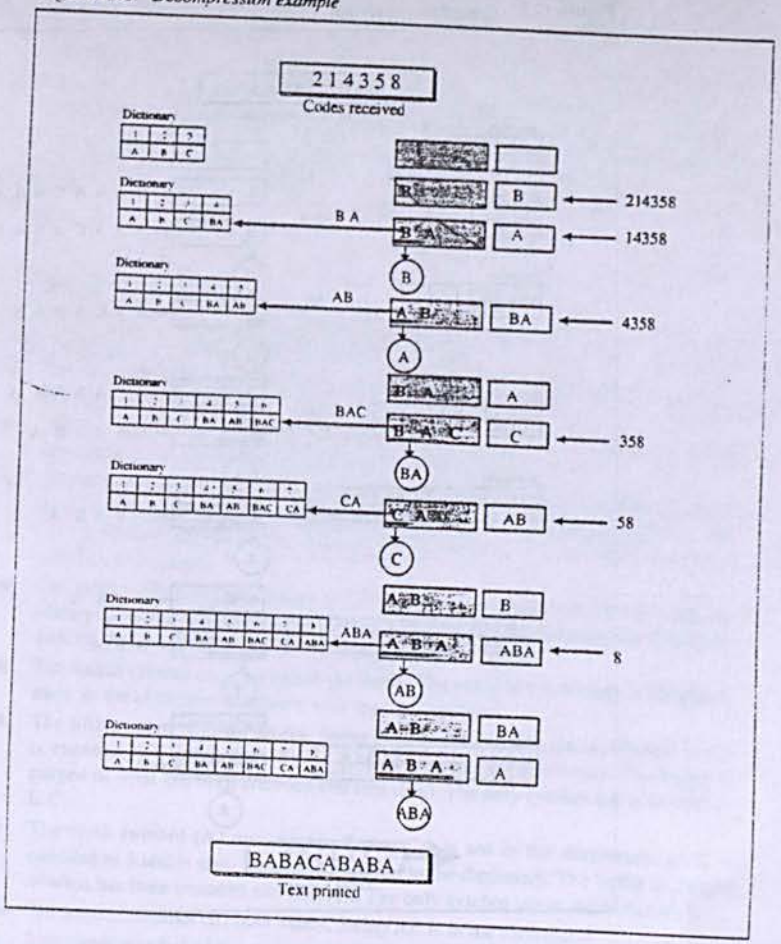
Figure G.5 *Decompression algorithm*



is printed: the whole string BA is added to the dictionary: the buffer is purged of B: the only symbol left is A.

- The temporary buffer is empty, so the next code should be decoded. The next code is 4 (note that the corresponding entry for 4 is already in the dictionary as it was entered by the previous iteration) which is decoded as BA. The string enters the temporary buffer. The buffer reads only the first character and appends this to the buffer contents. The result is AB. The code is not in the dictionary, so A is printed and AB is added to the dictionary and the buffer is purged of A: the only symbol left over is B.
- The rest of the process is similar. We leave it to the reader to follow through.

Figure G.6 *Decompression example*



Rujukan

Rujukan

- Behrouz Forozan, "Introduction to Data Communication and Networking", International Edition, McGraw-Hill, 1998.
- Zainalabedin Navabi, "VHDL-Analysis and Modelling of Digital System", International Edition, McGraw-Hill, 1998.
- Sudhakar Yalamanchili, "Introductory VHDL : From Simulation To Synthesis", Prentice Hall, 2001.
- Zaidi Razak, "Design a Neuron Chip together with Sigmoid Transfer Function", Fakulti Sains Komputer dan Teknologi Maklumat Universiti Malaya, 1996/1997.
- Noorizan Muhamad Mursid, "Rekabentuk Perkakasan Khas ECC", Fakulti Sains Komputer dan Teknologi Maklumat Universiti Malaya, 2000/2001.
- William A. Shay, "Understanding Data Communication & Network", Second Edition, International Thomson Publishing Company, 1999.
- www.rdrop.com/~cary/html/data_compression.html
- www.ee.uwa.edu.au/~roberto/teach/itc314/java/Huffman/huff.html
- www.data_compression.com/lossless.htm
- www.eeglosarry.com/vhdl.htm