LATIHAN ILMIAH (WXES 3182)

LITAR ARITMETIK REJA (BERMODUL) PENAMBAH CLA

DISEDIAKAN OLEH:
WAN NURUL RUKIAH BT WAN RASDI
WEK 000176
IJAZAH SARJANA MUDA SAINS KOMPUTER
(JABATAN SISTEM DAN TEKNOLOGI KOMPUTER)
UNIVERSITI MALAYA, KUALA LUMPUR

DISEDIAKAN UNTUK:
EN ZAIDI BIN RAZAK (MODERATOR)
EN NOORZAILY MOHAMED NOOR (PENYELIA)

ABSTRAK

Laporan projek latihan ilmiah dua ini adalah lanjutan daripada proposal yang telah dicadangkan dan diluluskan pada latihan ilmiah satu. Laporan ini menerangkan berkenaan dengan pembangunan penambah CLA (*carry lookahead adder*) telah dilaksanakan sepanjang semester dua sesi 2002/2003 ini. Penambah CLA telah dibangunkan menggunakan perisian PeakFPGA Designer Suite. Penambah yang dihasilkan akan berfungsi untuk melaksanakan operasi penambahan .Proses rekabentuk bermula dengan membuat pengekodan bagi penambah. Setelah coding bagi penambah berjaya dilaksanakan, perlaksanaan testbench pula dilakukan. Testbench yang telah selesai dilaksanakan kemudiannya dikompilasikan (*compiling*) dan kemudian dihubungkan (*linking*). Setelah semua perkara yang disebutkan sebelum ini selesai dilakukan, proses simulasi telah dilakukan. Penghasilan penambah ini kemudiannya digabungkan dengan modul *multiplier* dan *converter* yang dilakukan oleh dua ahli yang lain. Penggabungan ketiga-tiga modul ini membentuk litar aritmetik reja. Ia membuktikan bahawa kelajuan operasi aritmetik di dalam modul-modul reja dipertingkatkan dengan menghadkan perambatan bawaan (*carry propagation*). Litar penambah yang dihasilkan akan melibatkan dua vektor masukan iaitu a dan b, dan satu vektor keluaran, iaitu s. Standard logik yang akan digunakan ialah *ieee.std_logic_1164.ALL dan ieee.std_logic_arith.ALL*. Laporan ini juga mengandungi penerangan berkaitan dengan kekangan-kekangan yang dijangkakan akan dihadapi, kajian literasi dan rekabentuk litar.

PENGHARGAAN

Terlebih dahulu saya beryukur ke hadrat Ilahi kerana dengan limpah kurnia-Nya saya telah berjaya menyiapkan latihan ilmiah dua dan laporan ini dalam tempoh yang telah ditetapkan.

Ucapan setinggi-tinggi penghargaan diucapkan kepada penyelia saya Encik Noorzaily Bin Mohamed Noor, pensyarah Jabatan Sistem dan Rangkaian Komputer, Fakulti Sains Komputer dan Teknologi Maklumat, Universiti Malaya di atas kerjasama yang telah beliau berikan sepanjang dua semester ini. Terima kasih juga diucapkan kepada moderator, Encik Zaidi Bin Razak dan ahli-ahli kumpulan, Fadzlin Noor Arba'in Bt Mohd Said dan Leorna Shazerine Bt Abdul Mutalib di atas kerjasama yang amat konsisten dan permuafakatan yang telah diberikan sepanjang penyelidikan, perjumpaan dan viva. Setinggi perhargaan juga diucapkan kepada Encik Yamani Idna Bin Idris di atas tunjuk ajar berkaitan VHDL. Tidak lupa juga penghargaan kepada ibu bapa dan keluarga saya yang telah banyak memberi dorongan, kerjasama dan semangat sepanjang perlaksanaan latihan ilmiah ini. Terima kasih juga diucapkan kepada mana-mana pihak yang terlibat secara langsung atau tidak langsung di dalam perlaksanaan latihan ilmiah 1 dan 2, dan penghasilan laporan ini. Semoga budi baik kalian mendapat keberkatan daripada Ilahi.

7 Januari 2003

SENARAI ISI KANDUNGAN

PENDAHULUAN

ISI KANDUNGAN

BAB 01 PENGENALAN

BAB 02 KAJIAN LITERASI

SENARAI JADUAL

SENARAI RAJAH

BAB 01 PENGENALAN

1.1 PENGENALAN KEPADA TAJUK

Pembangunan yang akan dilaksanakan ialah penghasilan modul penambah menggunakan perwakilan nombor binari. Setelah melaksanakan pelbagai kajian dan penyelidikan, penambah berkelajuan tinggi telah dipilih.

Jenis penambah (adder) yang dipilih adalah *Carry Lookahead Adder* (CLA). Jenis ini dipilih kerana operasi penambahan yang lebih pantas boleh dilaksanakan berbanding jenis-jenis penambah yang lain. Kombinasi modulo penambah aritmetik , modulo pendarab aritmetik dan pengesan ralat akan menghasilkan litar bermodul yang diberi nama **Litar Aritmetik Reja.** Litar yang akan dihasilkan adalah litar aritmetik 8-digit.

Pengekodan bagi penambah CLA dilakukan menggunakan perisian PEAKVHDL. *Coding* dan *testbench* dibangunkan.

1.2 DEFINASI MASALAH

Definasi masalah merupakan apakah masalah-masalah yang perlu diselesaikan dalam pembangunan litar ini.

Masalah-masalah yang wujud dari segi berlakunya perambatan bawaan semasa penambahan, kelajuan operasi aritmetik yang perlahan dan semakan ralat yang perlahan akan cuba diselesaikan melalui penghasilan litar aritmetik reja.

Untuk melaksanakan litar menggunakan VHDL, perwakilan radix-2 *signed digit* perlu diberikan perwakilan binari. Ini adalah kerana komputer hanya membenarkan operasi menggunakan angka 0 dan 1 sahaja.

Masalah untuk merekabentuk litar aritmetik reja yang lebih padat juga perlu diatasi dengan pengurangan terhadap penggunaan get-get logik perlu dilaksanakan bagi mengurangkan kekompleksan litar.

Litar bermodul yang mempunyai rekabentuk yang paling dipercayai, dengan kos dan masa yang minimum perlu dihasilkan untuk memenuhi keperluan Projek Ilmiah tahap akhir.

1.3 SKOP PERMASALAHAN

Skop permasalahan yang perlu diselesaikan di sini dibahagikan kepada 2 bahagian:

Skop I (Skop permasalahan litar aritmetik reja SD)

Skop 1 merupakan skop permasalahan litar aritmetik reja SD. Pengesan ralat akan dihasilkan dengan menggunakan sistem nombor digit bertanda (8-digit dan 16-digit) yang berkelajuan tinggi. Proses ini berlaku bermula dengan merekabentuk penambah, merekabentuk pendarab dan diikuti dengan penyemak ralat . Akhirnya modul penambah , modul pendarab dan penyemak ralat akan digabungkan untuk membentuk litar aritmetik reja. Bagi merealisasikan matlamat ini, integrasi perlu dipastikan wujud di antara nombor perduaan (*binary numbers*), nombor digit bertanda (*signed digit*) dan nombor reja (*residue numbers*) kerana ketiga-tiga perwakilan ini perlu digabungkan. Setiap modul perlu dipastikan dapat berfungsi antara satu sama lain tanpa kewujudan sebarang ralat atau komplikasi.

Skop II (Skop permasalahan penambah aritmetik reja SD)

Skop permasalahan 2 ,merupakan permasalahan penambah aritmetik reja. Algoritma penambahan modulo m p-digit asas 2 perlu diimplementasikan bagi operasi penambahan 2 integer .Ini ditunjukkan sebagai : $[A+B]_{SD}$. Pengiraan penambahan perlu dilaksanakan dan kemudiannya dihuraikan di dalam bahasa pemprograman perkakasan (VHDL). Memandangkan penambahan aritmetik reja

melibatkan penggunaan modulo, jenis modulo m bagi algoritma penambahan modulo m perlu ditentukan di mana : $m=\{2^P, 2^{P+1}$ dan $2^{P-1}\}$ di mana p=digit yang dipilih

Rekabentuk penambahan aritmetik reja yang lebih padat perlu diperolehi dengan cuba mengurangkan penggunaan get-get logik. Di samping itu usaha untuk mendapatkan operasi penambahan dalam masa yang singkat juga akan dilaksanakan. Masa lengahan *(delay)* semasa simulasi pula akan cuba dikurangkan bagi mencapai matlamat ini.

1.4 OBJEKTIF/ TUJUAN

Pembinaan litar aritmetik reja menggunakan metodologi VHDL dilaksanakan untuk menyelesaikan masalah-masalah yang disebutkan sebelum ini. Metodologi VHDl digunakan kerana VHDL amat berkesan untuk mengimplementasikan FPGA *(Field Programmable Gate Array)* ke teknologi ASIC (*applications specific integrated circuit).* Teknologi ASIC merupakan teknologi di mana litar integrasi direkabentuk untuk mengeluarkan satu atau lebih fungsi yang spesifik dan proses ini dilaksanakan dalam satu cip semikonduktor dengan tujuan untuk mengurangkan saiz di dalam sistem, mengurangkan bilangan penyambungan yang diperlukan dan mengurangkan bilangan komponen yang perlu digunakan di dalam sesuatu sistem.Walaupun rekabentuk penambah aritmetik sehingga ke tahap simulasi sahaja, namun teknologi ASIC tetap digunakan, walaupun tidak sampai ke peringkat akhir, iaitu peringkat penghasilan cip.

Matlamat merekabentuk penambah aritmetik direalisasikan dengan menggunakan aplikasi VHDL, iaitu perisian PEAK VHDL. Sebelum ini, di dalam laporan ilmiah satu, perisian Xilinx foundation Series 2.1i telah dicadang untuk digunakan. Masalah yang timbul akibat daripada ketidaksesuaian perisian Xilinx pada komputer yang menggunakan Windows 2000 menyebabkan Peak VHDL digunakan bagi mengatasi masalah yang timbul. Masalah litar aritmetik sistem nombor perduaan juga menyumbang kepada sebab mengapa penghasilan penambah aritmetik dilakukan. Hasil daripada kombinasi penambah dan pendarab aritmetik akan dibandingkan dengan pengesan ralat SD.

1.5 KEKANGAN

Kekangan merupakan had-had yang dijangka tidak dapat diatasi atau dilakukan. Kekangan yang dijangka mungkin berlaku semasa merekabentuk litar aritmetik reja SD adalah:

a) keupayaan ruang ingatan yang terhad

b) kekangan aplikasi VHDL

c) kekangan masa

Litar aritmetik ini dijangka menggunakan perwakilan SD, 8-digit dan 16-digit. Keupayaan ruang ingatan yang ada adalah terhad. Ruang ingatan yang ada mungkin tidak dapat menampung beban pemprosesan. Masalah ini dijangka akan timbul ketika merekabentuk litar 16-digit. Masalah ini mungkin akan menyebabkan setiap pemprosesan mungkin mengambil masa yang agak lama untuk dilaksanakan.

Ralat kod sumber VHDL mungkin berlaku semasa pengkompilan merupakan kekangan yang dianggap mampu menggagalkan usaha merekabentuk litar penambah aritmetik.

Kekangan masa bagi menyiapkan latihan ilmiah dua (WXES3182) menyebabkan pembangunan litar aritmetik reja SD dijangka sehingga ke proses simulasi sahaja. Walaupun sehingga ke proses simulasi sahaja, namun usaha mencapai matlamat keseluruhan akan cuba dicapai sepenuhnya.

1.6 PENJADUALAN

Bahagian ini menerangkan secara ringkas penjadualan proses kerja telah selesai dilaksanakan dan juga yang dicadangkan daripada mulanya Latihan Ilmiah 1 (WXES3181) sehingga Latihan Ilmiah 2 (WXES3182).

Jadual 1.6 Penjadualan Proses Kerja bagi Latihan Ilmiah 1

BILANGAN	PROSES KERJA YANG DICADANGKAN	JANGKAMASA YANG DIANGGARKAN
1	Memilih dan mendapatkan tajuk.	1-7 Jun 2002
2	Menjalankan kajian dan analisis terhadap tajuk, mencari bahan rujukan yang bersesuaian.	1-30 Jun 2002
3	Membuat perjumpaan, membuat penyusunan terhadap isi-isi yang telah dikumpulkan. Membuat persediaan untuk Viva WXES3181. Menyiapkan slaid-slaid menggunakan Power Point.	1 Julai- 27 Ogos 2002
4	Viva 3181	28 Ogos 2002 (4.30-6.00 petang)
5	Menyiapkan laporan akhir WXES3181	29 Ogos-12 September 2002
6	Penghantaran laporan akhir WXES3181	13 September 2002

Jadual 1.6 Penjadualan Proses Kerja bagi Latihan Ilmiah 2

BILANGAN	PROSES KERJA YANG DICADANGKAN	JANGKAMASA YANG DIANGGARKAN
7	Memulakan proses rekabentuk menggunakan metodologi yang telah dicadangkan mengikut turutan bermula dari Blok Diagram, Pengekodan, Pengkompilan dan akhirnya Simulasi.	Oktober-Disember 2002
8	Menyiapkan jurnal yang dikehendaki oleh penyelia	Penghantaran telah dilakukan pada pertengahan bulan Januari 2003
9	Viva WXES3182	5 Februari 2003 (tarikh ditentukan oleh penyelia dan moderator)
10	Menyediakan dan menghantar laporan akhir bagi WXES3182.	7 Februari 2003 Penghantaran laporan kepada penyelia dan moderator 28 Februari 2003 Penghantaran laporan berjilid kepada Pejabat FSKTM

BAB 02 KAJIAN LITERASI

2.1 Pengenalan

Revolusi digital telah bermula sejak hampir lima dekad yang lalu dan kini telahpun meluas. Revolusi digital ini berlaku akibat daripada beberapa faktor. Di antaranya ialah:

a) Kebolehupayaan menghasilkan keputusan.

Jika kita memberi set masukan (input) yang sama (bagi *value* dan *time sequence*), rekabentuk litar digital akan sentiasa menghasilkan keputusan yang sama dengan yang dikehendaki. Faktor suhu ataupun *power supply voltage* tidak akan mempengaruhi keputusan sama sekali.

b) Rekabentuk dipermudahkan daripada kesulitan.

Ini bermaksud rekabentuk digital tidak memerlukan pengetahuan atau *skill* matematik yang khusus, dan kelakuan bagi litar logik boleh diingati dengan mudah tanpa perlu mengetahui sebarang pengetahuan dalaman yang khas mengenai operasi bagi setiap peranti yang diperlukan bagi pengiraan.

c) Kebolehan melakukan *programming*.

Rekabentuk digital kini boleh dilaksanakan dengan menulis program, di dalam VHDL. Bahasa ini membenarkan kedua-dua struktur dan fungsi bagi litar digital untuk dispesifikasikan sebagai model. Di samping *compiler*, HDL yang tipikal juga datang bersama program simulasi dan sintesis. Perisian ini digunakan untuk menguji kelakuan bagi model perkakasan sebelum sebarang perkakasan sebenar dibina.

d) Kelajuan

Peranti digital yang ada pada hari ini adalah berkelajuan tinggi. Contohnya transistor individu di dalam litar integrasi yang terpantas mampu *switch* kurang daripada 10 pikosaat, dan satu peranti kompleks yang dibina daripada transistor-transistor ini boleh memeriksa input-inputnya sendiri dan menghasilkan output kurang daripada 2 nanosaat. Ini bermakna bahawa peranti tersebut boleh menghasilkan 500 juta keputusan per saat. [1].

Jika dibandingkan dengan perisian, rekabentuk perkakasan mempunyai banyak kelebihan , seperti yang telah disebutkan sebelum ini. Contohnya ialah perisian MATLAB (Matrix Laboratory). Perisian ini merupakan perisian visual dan pengkomputeran yang terbukti berprestasi tinggi. Perisian ini membekalkan bahasa yang boleh melaksanakan operasi aritmetik berasaskan operasi matrix dan *array*. Kelemahan utama perisian ini adalah ia perlu mematuhi prinsip matrix dan lebih memfokuskan kepada konsep sains komputer dan kurang konsep matematik. Kelajuan dan keupayaan aritmetiknya bergantung kepada CPU dan ALU yang digunakan. Tambahan pula, pengguna perlu ada kemahiran sains komputer dan matematik untuk membolehkan mereka menggunakan perisian ini. Ini akan menyukarkan pengguna yang tidak mahir dan menghadkan pasaran.

2.2 Analisa Sistem Terdahulu

Pada tahun 1992, Bayoumi dan Jullien telah menghasilkan penambah reja menggunakan Sistem Aritmetik Binari. Perambatan bawaan (*carry propagation)* telah menghadkan kelajuan operasi aritmetik di dalam modul-modul reja tersebut.

A.Avizienis di dalam jurnalnya :-'Signed-Digit Number Representation for fast parallel aritmetic' telah menyatakan bahawa penggunaan Signed-Digit ketika penambahan boleh menghadkan *carry propagation.*

2.3 Sistem Nombor Reja

Sistem nombor reja (RNS) menarik minat ramai kerana sifat moduli yang berdiri sendiri di antara satu sama lain. Satu nombor di dalam RNS diwakili oleh modulus bagi setiap reja dan operasi aritmatik adalah dilaksanakan bergantung kepada setiap modulus. Oleh kerana moduli adalah berdiri sendiri di antara satu sama lain, tiada *carry propagation* yang wujud dan mudah untuk melaksanakan perkomputeran RNS kepada *multi –ALU system.*

Prinsip penukaran integer biasa kepada nombor reja adalah dengan menggunakan modulo.

- Persamaan penukaran reja yang asas adalah :-

  $x_m = X - m[X/m]$..... (1) x_m = integer reja

  X = integer

  m = modulo ; di mana $m = 2p\text{-}1$

- Di dalam sistem ini, modulo digambarkan seperti berikut:

  m=2p+- h, di mana **h € {-1,0,1}.**

  $m = 2p$-1 telah dipilih sebagai modulo.

- Biar A=integer ;

  A diwakili oleh n-tuple (A1,A 2,...,A n) dalam RNS bersama moduli m1,m2,...,mn.

  Maka A_1 berada dalam julat [0,mi] melalui: Ai=A mod mi= |A|mi

- Katakan X adalah integer dan m integer positif, maka x= <X>m diertikan sebagai integer dalam julat [-m, m].

  Rumus x= <X>m adalah:

  x= <X>m =sign (X)x |abs (X)|m

  ATAU

  x= <X>m =sign (X) x abs (X)|m-sign(X) x m

di mana sign(X)= $\begin{cases} -1, X<=0 \\ 1, X>=0 \end{cases}$

abs (X)= $\begin{cases} -X, X<=0 \\ X, X>=0 \end{cases}$

- Untuk menukarkan satu nombor reja kepada satu integer:-

Contoh pengiraan: Katakan integer=17, m=7 maka,

$x = <X>_m = <17>_7$

$= \text{sign}(X) \times |\text{abs}(X)|_m$

$= 1 \times |17|_7$

$= 1 \times 17 \bmod 7 = 1 \times 3 = 3$ #

ATAU

$x = <X>_m$

$= \text{sign}(X) \times \text{abs}(X)|_m - \text{sign}(X) \times m$

$= (1 \times 3) - (1 \times 7)$

$= -4$ #

*X=3 dan X=-4 (Reja membenarkan **redundant** berlaku)

- **Penukaran RNS kepada nombor desimal**

Katakan penukaran bagi $y= (3 \mid 2 \mid 4 \mid 2)_{RNS}$ kepada nombor desimal. Berdasarkan kepada *property* RNS:

$$(3 \mid 2 \mid 4 \mid 2)_{RNS} = (3 \mid 0 \mid 0 \mid 0)_{RNS} + (0 \mid 2 \mid 0 \mid 0)_{RNS}$$

$$+ (0 \mid 0 \mid 4 \mid 0)_{RNS} \quad (0 \mid 0 \mid 0 \mid 2)_{RNS}$$

$$= 3 \text{ X } (1 \mid 0 \mid 0 \mid 0)_{RNS} + 2\text{X } (0 \mid 1 \mid 0 \mid 0)_{RNS}$$

$$+ 4 \text{ X } (0 \mid 0 \mid 1 \mid 0)_{RNS} + 2 \text{ X } (0 \mid 0 \mid 0 \mid 1)_{RNS}$$

Dengan mengetahui nilai-nilai bagi empat konstan (posisi pemberat RNS) membolehkan kita menukar sebarang nombor dari RNS $(8 \mid 7 \mid 5 \mid 3)$ kepada desimal menggunakan proses pendaraban dan penambahan.

$(1 \mid 0 \mid 0 \mid 0)_{RNS} = 105$

$(0 \mid 1 \mid 0 \mid 0)_{RNS} = 120$

$(0 \mid 0 \mid 1 \mid 0)_{RNS} = 336$

$(0 \mid 0 \mid 0 \mid 1)_{RNS} = 280$

Oleh yang demikian,

$$(2 \mid 2 \mid 4 \mid 2)_{RNS} = [(3 \text{ X } 105) + (2 \text{ X } 120) + (4 \text{ X } 336) + (2 \text{ X } 280)]$$

$$= 779$$

2.4 Aritmetik Reja Berdasarkan Perwakilan Signed Digit

1. Dalam perwakilan radix 2-SD, integer X diwakili seperti berikut:

   $X = (X_{q-1}, X_{q-2}, X_{q-3}, \ldots, X_0)$ SD

2. Perwakilan nombor SD juga mempunyai *redundancy*.

   Contohnya bagi angka 7, di mana q=4

   7 boleh diwakili oleh:

   (0,1,1,1)SD' (1,-1,1,1)SD, (1,0,0,-1)SD

   Kaedah penukaran nombor SD ke binari,ditunjukkan pada jadual di bawah:

Jadual 2.4 Penukaran nombor signed digit ke binari

a_i	a_i (1)	a_i (0)
-1	1	1
0	0	0
1	0	1

a_i (1) = tandaan

a_i (0) = nilai mutlak bagi a_i

3. p-digit radix 2-SD bagi nombor a akan dipersembahkan oleh vektor berukuran 2p-bit.

$a = (a_{p-1}, a_{p-2}, \ldots, a_0)_{SD}$

$= [a_{p-1}(1), a_{p-1}(0), a_{p-2}(1), a_{p-2}(0), \ldots, a_0(1), a_0(0)]$

4. Sebagai contoh:

$(1, 0, 0, -1)_{SD} = [01\ 00\ 00\ 11]$

5. Dengan menggunakan perwakilan binari bagi 2 digit bertanda (SD) x_i dan y_j;

$p_{ij} = x_i y_j$;

$nX_i = -x_i$ boleh ditunjukkan oleh pengiraan di bawah:-

$p_{ij}(0) = x_i(0)$ and $y_j(0)$;

$p_{ij}(1) = (x_i(1)$ or $y_j(1))$ and $p_{ij}(0)$;

$nX_i(0) = X_i(0)$

$nX_i(1) = (\text{not } X_i(1))$ and $nX_i(0)$

di mana and = operasi get AND

xor = operasi get EXCLUSIVE-OR

not = operasi get NOT

2.4 Penambah modulo m signed digit (MDSA)

Penyelesaian bagi pengiraan <a+b>m

Penambah penuh SD dibina dengan 2 blok, iaitu ADD1 dan ADD2.

Setiap fungsi adder adalah berdasarkan pada algoritma di bawah :-

(ADD1) :

Apabila abs(a_i) = abs (b_i)

$$z_i = 0$$

dan $c_i = (a_i + b_i)$ div 2;

Apabila abs(a_i) = abs(b_i)

$$z_i = \begin{cases} -(a_i + b_i) & \text{jika } (a_i + b_i) \text{ dan } (a_{i-1} + b_{i-1}) \\ & \text{mempunyai tanda yang sama} \\ a_i + b_i & \text{sebaliknya} \end{cases}$$

dan

$$c_i = \begin{cases} a_i + b_i & \text{jika } (a_i + b_i) \text{ dan } (a_{i-1} + b_{i-1}) \\ & \text{mempunyai tanda yang sama} \\ 0 & \text{sebaliknya} \end{cases}$$

(ADD2) : $s_i = z_i + c_{i-1}$

Sentiasa benar supaya $2c_i + z_i = a_i + b_i$

s_i dan c_{i-1} tidak mempunyai tanda yang sama supaya

z_i terdiri dari unsur {-1,0,1}

Pengubahsuaian algoritma $<a+b>_m$ di atas dilakukan bagi memenuhi persamaan VHDL.

L1) Apabila abs (a_i) = abs (b_i) , maka

$Z_i = 0$ dan $C_i = (a_i + b_i)$ div 2

Apabila abs (a_i) $\neq$ abs (b_i),maka

$$Z_i = \begin{cases} -(a_i + b_i) & \text{jika } T = TL \\ a_i + b_i & \text{sebaliknya} \end{cases}$$

$$C_i = \begin{cases} a_i + b_i & \text{jika } T = TL \\ 0 & \text{sebaliknya} \end{cases}$$

di mana $T = a_i\,(1)$ OR $b_i\,(1)$

$TL = a_{i-1}\,(1)$ OR $b_{i-1}\,(1)$

L2) $S_i = Z_i + C_{i-1}$ bagi $i \neq 0$

$S_0 = Z_0 + (-h) \times C_{p-1}$

2.5 SEJARAH VHDL

VHDL telah dibangunkan pada awal tahun 80-an sebagai projek penyelidikan integrasi litar yang berkelajuan tinggi. Ketika program VHSIC dijalankan, penyelidik-penyelidik telah didedahkan dengan kebimbangan bagi menghuraikan litar yang berskala terlalu besar dan untuk menguruskan sebarang masalah rekabentuk litar yang bersaiz besar. Ketika itu, hanya alatan rekabentuk level-get sahaja yang wujud, jadi penyelidik bersetuju dengan hakikat bahawa kaedah-kaedah rekabentuk berstruktur perlu dihasilkan bagi mengatasi masalah-masalah yang sedia ada.

Tiga syarikat iaitu *IBM*, *Texas Instruments* dan *Intermetrics* telah memeterai kontrak dengan *Department Of Defense* (DoD) untuk menyudahkan spesifikasi dan perlaksanaan bagi satu bahasa baru, yang merupakan kaedah deskripsi bahasa rekabentuk. Versi pertama bagi bahasa tersebut, VHDL,versi 7.2, telah diwujudkan pada tahun 1985.

2.5.1 IEEE Standard 1076-1987

Pada tahun 1986, IEEE telah diperkenalkan bersama proposal untuk menyeragamkan bahasa , di mana ia telah berjaya dilakukan pada tahun berikutnya selepas beberapa penambahan dan pengubahsuaian yang kuat dan teguh. Keputusannya, penyelarasan (*standard*) yang diperkenalkan ialah IEEE 1076-1987 merupakan perkara asas bagi setiap produk simulasi dan sintesis yang

diniagakan kini. Pada tahun 1994, penambahan dan pembaharuan versi bagi bahasa tersebut, IEEE 1076-1993 telah diperkenalkan pada tahun 1994.

2.5.2 IEEE Standard 1164

Walaupun IEEE standard 1076 telah memberi pengertian yang lengkap bagi bahasa VHDL, namun ada beberapa aspek pada bahasa tersebut yang menyebabkan ianya menjadi sukar untuk menulis rekabentuk mudah alih yang lengkap. Masalah tersebut menghalang VHDL untuk menyokong pelbagai jenis pengabstrakan data. Setelah IEEE 1076-1987 di-*adopted*, syarikat-syarikat mula membuat penambahan bagi pelbagai jenis isyarat (*signals*) untuk membolehkan pelanggan-pelanggan mereka mensimulasikan litar elektronik yang mempunyai kekompleksan yang tinggi.

Ini menimbulkan masalah kerana deskripsi rekabentuk yang dimasukkan menggunakan satu simulator sentiasa tidak kompatibel dengan persekitaran simulasi yang lain. VHDL dengan pantasnya menjadi tidak seragam. Oleh sebab itu, satu *standard* lain telah diwujudkan oleh committee dan diterima oleh IEEE. *Standard* ini dinomborkan 1164, mendefinasikan satu pakej *standard* yang mengandungi pelbagai pengertian bagi satu standard jenis data *9-valued*. Jenis standard ini dikenali sebagai standard logic, dan pakej IEEE1164 sering dirujuk sebagai standard logic package atau kadangkala dikenali sebagai MVL9 (untuk pelbagai nilai logic-m*ultivalued logic,nine values*).

2.6 APAKAH VHDL?

VHDL merupakan singkatan bagi *Very High Speed Integrated Circuit (VHSIC) Hardware Description Language.* la diperkenalkan oleh United States Department of Defense pada tahun 1981 .

VHDL merupakan bahasa pemprograman bagi perkakasan yang telah direka dan dioptimumkan untuk memerihalkan kelakuan (behavior) bagi litar-litar digital dan sistem-sistem. VHDL menggabungkan ciri-ciri berikut:

1) **Bahasa Permodelan Simulasi**

VHDL mempunyai pelbagai cirri-ciri yang sesuai untuk memerihalkan kelakuan bagi komponen-komponen elektronik bermula daripada get-get logic yang mudah kepada mikropemproses yang lengkap dan cip-cip. Ciri yang ada pada VHDL membenarkan aspek-aspek elektronik bagi kelakuan litar seperti kenaikan dan kejatuhan masa bagi isyarat, kelambatan menerusi get-get, dan operasi kefungsian mudah diperihalkan. Keputusan simulasi bagi model VHDL kemudiannya boleh digunakan untuk membina blok-blok pada litar yang lebih besar, menggunakan schematics, diagram blok atau deskripsi aras-sistem VHDL bagi tujuan simulasi.

2) **Kemasukan Bahasa Rekabentuk**

VHDL membenarkan kelakuan bagi litar elektronik yang kompleks diletakkan ke dalam system rekabentuk bagi sintesis litar automatik. Seperti Pascal, C dan C++, VHDL mengandungi ciri-ciri yang berguna

untuk teknik merekabentuk, dan menawarkan set kawalan dan perwakilan data yang ciri-cirinya diperkayakan. Tidak seperti bahasa pemprograman yang lain, VHDL membekalkan ciri-ciri yang membenarkan kejadian serentak (*concurrent*) dijelaskan.

3) Bahasa Pengujian

Salah satu daripada aspek VHDL yang penting adalah keupayaannya untuk mengetahui spesifikasi prestasi bagi satu litar, yang sering dirujuk sebagai testbench. Testbench merupakan perwakilan VHDL bagi stimulus litar dan persamaan output yang dijangka yang menunjukkan kelakuan bagi litar melepasi masa. Testbench seharusnya sebagai bahagian yang diperlukan bagi mana-mana projek VHDL, dan sepatutnya dibentuk secara selari dengan deskripsi lain yang terdapat dalam litar.

4) Bahasa *Netlist*

VHDL merupakan bahasa yang kuat bagi kemasukan rekabentuk baru pada level yang tinggi, malah ia juga berguna sebagai salah satu bahagian bagi level yang rendah sebagai komunikasi di antara alatan-alatan yang berbeza di dalam rekabentuk berpandukan komputer. Ciri bahasa berstruktur bagi VHDL membolehkan ia untuk digunakan secara efektif sebagai bahasa *netlist*, menggantikan bahasa *netlist* yang lain seperti *EDIF*.

2.7 KELEBIHAN VHDL

Kelebihan VHDL adalah dari segi:

a) Meningkatkan produktiviti.

b) Meminimumkan kos dan masa yang digunakan.

c) Membekalkan rekabentuk yang lebih baik.

d) Boleh digunakan semula untuk teknologi terkini.

e) Alatan yang independence.

a) Meningkatkan produktiviti.

VHDL dapat meningkatkan produktiviti dengan penjimatan masa kepada pasaran. Simulasi bagi behavioral boleh mengurangkan masa merekabentuk dengan membenarkan masalah-masalah rekabentuk dikesan lebih awal. Ini dapat mengelakkan untuk merekabentuk semula rekaan pada tahap get.

b) Meminimumkan kos dan masa yang digunakan.

VHDL membekalkan proses rekabentuk yang paling *reliable*, dengan kos dan masa yang minimum.

c) **Membekalkan rekabentuk yang lebih baik.**

Simulasi behavioral membekalkan rekabentuk yang lebih baik dengan exploring senibina alternatif, yang akhirnya memberikan rekabentuk yang lebih baik.

d) **Boleh digunakan semula untuk teknologi terkini.**

Sesebuah system mungkin digunakan semula pada keadaan yang lain. Untuk menukar sesuatu rekabentuk kepada teknologi yang baru, satu spesifikasi diperlukan, bukan dengan dimulakan semula menggunakan lakaran atau *reverse-engineer.* Malah, pokok rekabentuk kepada pemerihalan VHDL boleh dilaksanakan dalam teknologi yang baru dengan mengetahui bahawa kefungsian yang digunakan adalah betul.

e) **Alatan yang independence.**

Pemerihalan VHDL bagi rekabentuk perkakasan dan test bench adalah mudah disesuaikan di antara mana-mana alatan rekabentuk, pusat rekabentuk dan projek *partners.*

Di antara syarikat yang membekalkan simulasi/sintesis adalah: ALDEC, ALTERA, CYPRESS, MENTOR GRAPHICS, MODELSIM, SYNOPSYS, CADENCE, XILINX dan sebagainya.

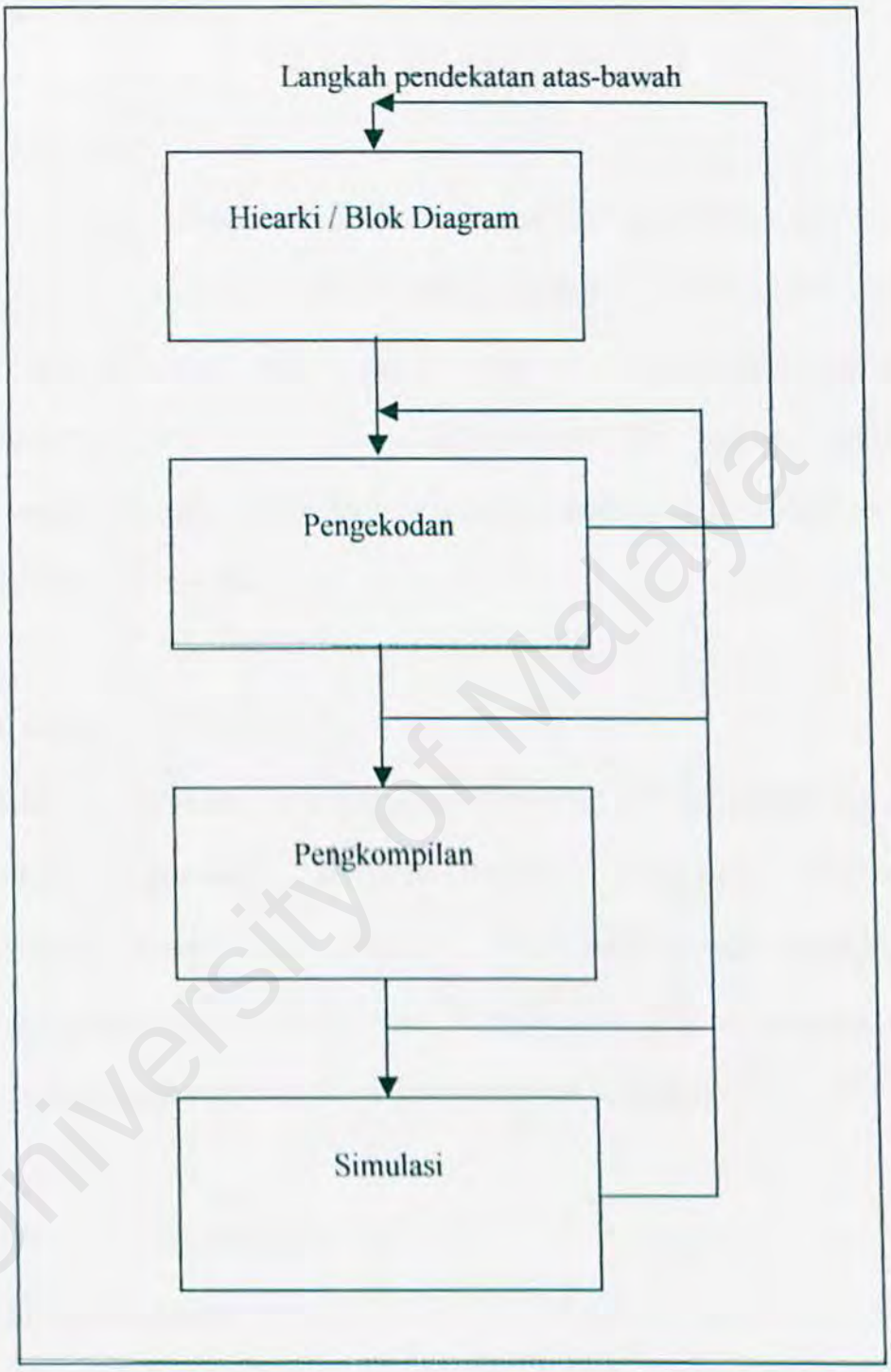
Bab 03 Metodologi

Bab ini menerangkan metodologi dan kelebihan rekabentuk yang digunakan di dalam pembangunan litar. Metodologi rekabentuk yang digunakan ialah bahasa perihalan perkakasan VHDL dan perisian Xilinx Foundation Series 2.1i digunakan bagi merealisasikan proses rekabentuk.

3.1 Metodologi rekabentuk VHDL

Metodologi VHDL akan digunakan untuk merekabentuk litar aritmetik. Metodologi ini merupakan metodologi rekabentuk peringkat tinggi, yang mempunyai ciri-ciri pengabstrakan yang tinggi, penganalisaan tukaran saling (tradeoff) yang cepat, penggunaan semula rekaan, boleh berulang dan sistem penyelakuan yang lengkap. Aliran metodologi rekabentuk ditunjukkan seperti dalam rajah 3.1a.

Proses rekabentuk adalah menggunakan pendekatan atas-bawah di mana litar keseluruhan dipisahkan kepada modul-modul yang berbeza. Deskripsi VHDL bagi setiap modul akan dihasilkan di mana dibahagikan kepada dua bahagian. Bahagian pertama memberi maklumat terperinci mengenai port-port input dan output bagi model-model. Bahagian yang kedua pula menunjukkan kelakuan fungsi bagi modul-modul tersebut.



Rajah 3.1a Metodologi yang digunakan untuk merekabentuk penambah

Penerangan bagi Rajah 3.1

Hierarki/Blok Diagram

Langkah pertama yang dilakukan adalah mengenalpasti pendekatan asas dan menghasilkan blok diagram. Litar aritmetik reja dipecahkan kepada blok-blok yang saling bersambungan-penambah, pendarab dan penyemak reja. Setiap blok mempunyai fungsi dan antaramuka yang tertakrif berasaskan spesifikasi litar. Kaedah ini dapat memberikan rangka kerja yang baik untuk menentukan modul dan antaramuka serta memenuhi ciri-ciri yang dikehendaki.

Pengekodan (*Coding*)

Pada tahap kedua ini, penulisan dan penghasilan kod-kod VHDL, antaramuka dan maklumat dalaman bagi penambah dilaksanakan. Bahasa pengaturcaraan VHDL yang memenuhi keperluan spesifik akan menjadikan kerja lebih mudah. Kata kunci (keywords) *highlighting* VHDL dibangunkan menggunakan struktur program dan dibangunkan untuk pemeriksaan sinteks dan untuk membuat perbandingan.

Pengekodan dibahagikan kepada dua:

a) Kod untuk penambah

b) *Testbench* untuk penambah

Pengkompilan

Analisis pembanding kod VHDL untuk kesalahan sintax dan pemeriksaan untuk kesesuaian dengan modul-modul yang lain dilaksanakan pada peringkat ini. Lakaran maklumat yang diperlukan dicipta untuk membolehkan simulator melakukan proses lakaran yang berikutnya.

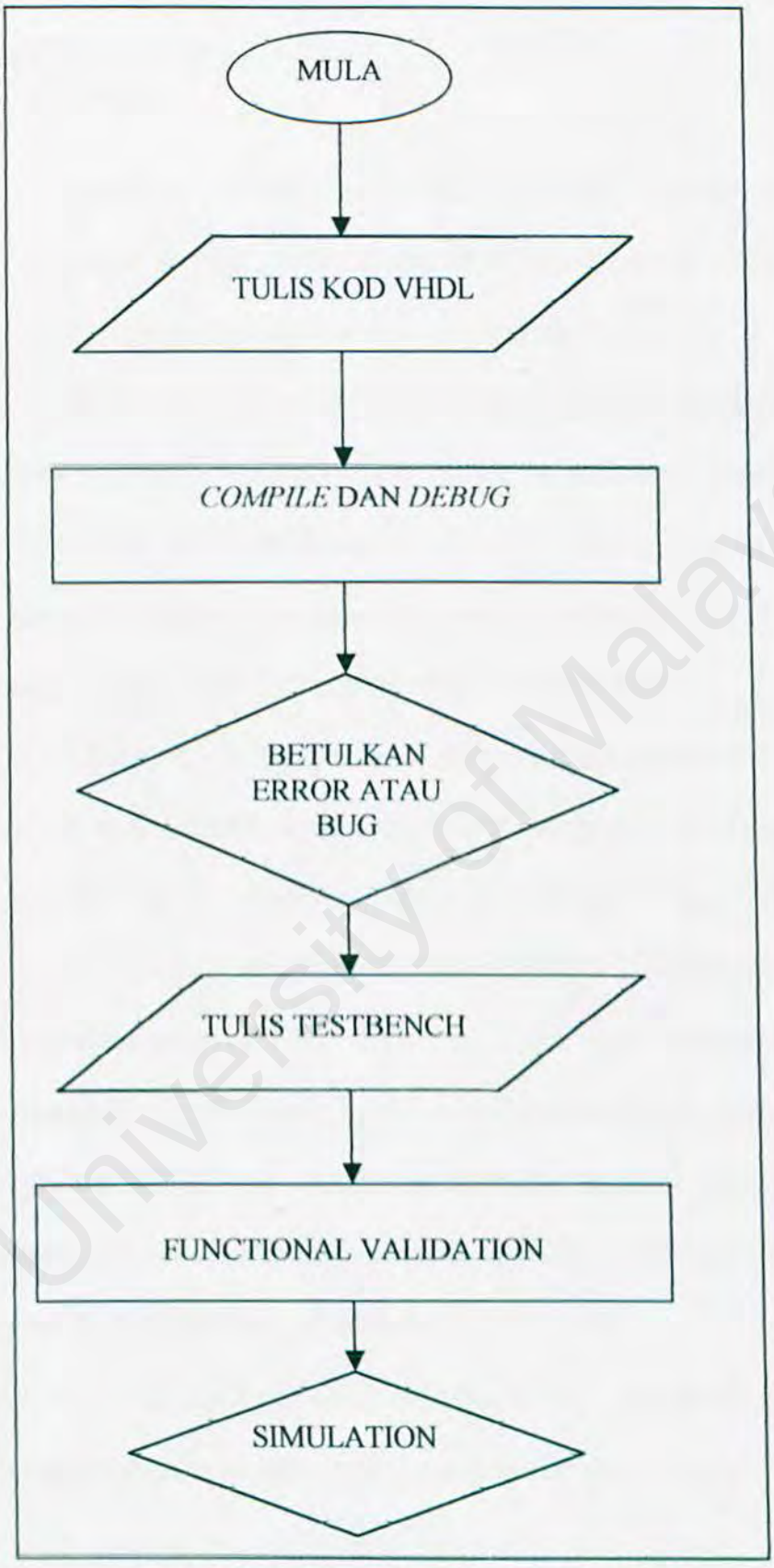
Simulation

Simulation dilakukan dengan tujuan untuk menentukan dan memohon input untuk lakaran. Ia dapat menunjukkan output walaupun tanpa mempunyai litar fizikal. Ia merupakan projek yang kecil, di mana input dan output dibuat secara manual. Bagi penghasilan projek yang besar pula , VHDL menggunakan "test benches" supaya permintaan input dibuat secara automatik dan dapat membandingkan dengan output yang dijangkakan. VHDL compiler akan digunakan untuk menganalisa kod yang telah dihasilkan dan kenalpasti *syntax error* dan semak kesesuaian untuk diguna dengan modul lain.

Langkah-langkah yang terlibat diringkaskan seperti berikut:

1) Analisa-semak syntax dan semantik,hasilkan perwakilan pertengahan.
2) Penerangan(elaboration)-hasilkan model simulasi dari hierarki rkbtk.
3) Penemulaan-beri nilai awalan kepada setiap pembolehubah.
4) Simulasi-Laksana simulasi berpandukan nilai input pengguna.

Rajah bagi perlaksanaan simulasi ditunjukkan pada muka sebelah:



Rajah 3.1b Aliran Perlaksanaan Simulasi

3.2 Kelebihan VHDL

Kelebihan utama menggunakan VHDL adalah kerana VHDL memudahkan seorang perekabentuk untuk menyiasat dan menerokai senibina rekabentuk yang direka. Pemerihalan domain berstruktur di dalam VHDL membolehkan seseorang itu memperihalkan seinibina secara semulajadi, yakni di dalam ertikata penyambungan set-set komponen. Pendekatan VHDL membenarkan pereka menumpukan sepenuh perhatian kepada pengoptimunan rekabentuk. Terbitan pemasaan juga senang digabungkan ke dalam rekabentuk selagi ia wujud dalam format yang dibenarkan oleh VHDL.

VHDL dapat mengenalpasti fungsian pada peringkat awal proses dan seterusnya melaksanakan simulasi ke atas rekabentuk di mana data-data atau maklumat-maklumat masukan dapat diperihalkan di dalam VHDL. Perihalan VHDL boleh digunakan atau ditukarkan kepada litar logik secara automatik berdasarkan teknologi yang digunakan, maka masa merekabentuk litar dapat dikurangkan. VHDL juga merupakan dokumentasi yang bebas dari teknologi. Perihalan VHDL juga sangat mudah dan difahami daripada *netlist* atau gambarajah dan ia boleh digunakan semula untuk menjanakan rekabentuk dalam teknologi lain tanpa mengubahkannya.

VHDL juga boleh dijadikan sebagai harta intelek (HI).Ini membenarkan ia digunakan semula oleh mana-mana pihak dengan keizinan.

3.3 Perbandingan VHDL dengan SYNOPSYS dan SCUBA

Seperti yang diketahui, pembentukan litar aritmetik reja melibatkan penambah (adder), pendarab (multiplier) dan pembanding (comparator).Bagi rekabentuk penambah, VHDL membekalkan penghalaan yang sangat laju dan rekabentuk yang lebih padat. Bagi Synopsys pula, untuk merekabentuk penambah, "designware library" diperlukan untuk merekabentuk penambah yang laju dan padat.

Bagi pendarab pula, VHDL membekalkan banyak pilihan senibina entity pendarab, yang bercirikan senibina tahap-bawah, simulasi yang dipermudahkan dan lebih fleksibel. Synopsys juga memerlukan jumlah sel-sel logik yang banyak jika saiz operands meningkat. Scuba pula hanya boleh menjana pendarab yang tidak bertanda (*unsigned multiplier*) sahaja. Ia tidak boleh digunakan untuk digit bertanda (*signed multiplier*).

Bagi pembanding pula, VHDL membenarkan perbandingan dibuat dengan cepat dan masa lengah perambatan adalah rendah. SYNOPSYS pula mempunyai kelajuan perbandingan yang perlahan dan masa lengah perambatan yang tinggi. Bagi SCUBA pula, kelajuan perbandingan yang rendah, dan masa lengah perambatan adalah tinggi.

3.4 Kelebihan menggunakan PeakVHDL

PeakVHDL merupakan produk perisian yang digunakan bagi membangunkan penambah CLA ini. Perisian ini berkeupayaan melaksanakan projek rekabentuk digital menggunakan VHDL . Produk PeakVHDL product mengandungi VHDL simulator, editor file sumber (*source file editor*), brosur hierarki (*hierarchy Browser)* dan lain-lain sumber bagi kegunaan pengguna-pengguna VHDL.

Sintesis produk bagi PeakFPGA synthesis berkongsi antaramuka pengguna yang sama dengan PeakVHDL, dan membolehkan seseorang untuk menterjemahkan deskripsi rekabentuk VHDLkepada perkakasan sebenar, menggunakan *advanced* teknologi sintesis FPGA .

PeakVHDL and PeakFPGA boleh digunakan untuk mencipta satu projek baru (penggunaan *Modul* dan *Testbench Wizard* dapat membantu mencipta deskripsi rekabentuk) atau untuk membantu menguruskan projek-projek VHDL yang sedia ada. Memandangkan VHDL merupakan bahasa standard, PeakVHDL and PeakFPGA boleh digabungkan dengan alatan-alatan lain (termasuk editor skematik, alatan rekabentuk level tinggi, dan alatan lain yang yang boleh didapati daripada pihak ketiga (third parties) untuk membentuk satu persekitaran rekabentuk elektronik yang lengkap.

BAB 04 Rekabentuk Sistem

Titik permulaan bagi proses merekabentuk litar logik ialah memperihalkan apa yang litar sepatutnya lakukan dan membuat *formulation* bagi struktur am litar tersebut. Langkah ini dilakukan secara manual oleh perekabentuk litar.

4.1 Litar Aritmetik Reja menggunakan perwakilan SD.

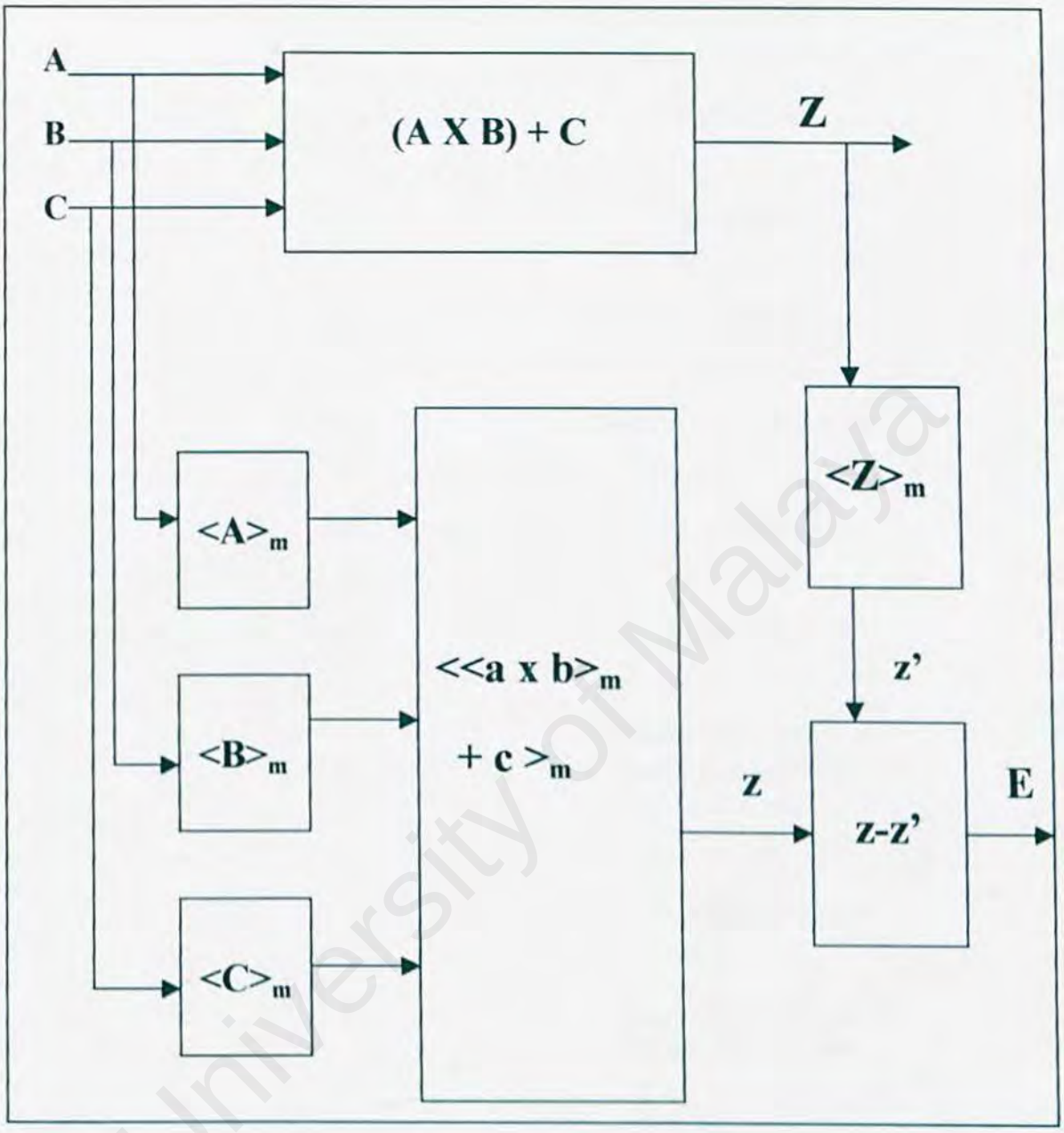
Litar aritmetik reja yang akan dibangunkan adalah terdiri daripada litar darab-tambah (*product-sum*) yang mengandungi penambah modulo m dan pendarab modulo m dan penyemak ralat yang di dalamnya mengandungi empat penukar perduaan-ke-reja. Operasi penambahan SD dan pendaraban SD berlaku di litar tambah-darab SD.

Penukar perduaan-ke-reja melaksanakan 4 operasi penukaran dengan perwakilan nombor nombor P-digit SD iaitu:-

$$a = \langle A \rangle m\ ,\ b = \langle B \rangle m,\ c = \langle C \rangle m \text{ dan } z = \langle Z \rangle m.$$

Operasi tambah (+) modulo m dan darab (x) modulo m berlaku di dalam litar penyemak ralat .Jika $E(z - z') = 0$, maka ralat pengiraan boleh dikesan sama ada di litar darab-tambah atau di penyemak ralat.

Senibina pembinaan litar aritmetik reja secara amnya ditunjukkan seperti di muka sebelah di mana modul-modul adalah terlibat dengan operasi-operasi yang telah dikhususkan.



Rajah 4.1 **Blok Diagram Yang Menunjukkan Operasi-Operasi Yang Terlibat Dalam Setiap Modul**

Simbol dan pengertian bagi setiap fungsi ditunjukkan dalam jadual di bawah:

Jadual 4.1 Simbol dan pengertian bagi setiap fungsi

SIMBOL	PENGERTIAN
A, B dan C	Input biasa
Z	Output biasa
m	Modulo yang digunakan
$<A>_m$, $<B>_m$, $<C>_m$	Input reja
$<Z>_m$	Output reja
z'	Hasil tambah biasa yang ditukarkan ke dalam bentuk reja
z	Hasil tambah bagi reja
E	Isyarat yang menunjukkan ralat telah berlaku

4.2 Jenis-jenis Penambah (adder)

Prestasi bagi sistem digital adalah bergantung kepada kelajuan litar-litar yang membentuk pelbagai unit-unit berfungsi yang tersendiri. Seperti yang telah diketahui, prestasi yang lebih baik boleh dicapai dengan menggunakan litar-litar aritmetik yang berkelajuan tinggi. Perkara ini boleh dicapai dengan menggunakan teknologi terkini yang dapat mengurangkan kelambatan (delay) dengan menambahkan kelajuan operasi di dalam get-get asas yang digunakan. Ini juga boleh dicapai dengan menstrukturkan semula unit berfungsi secara keseluruhan, yang boleh membawa kepada peningkatan prestasi.

Penambahan integer merupakan operasi yang paling penting. Walaupun untuk program yang tidak secara langsung melibatkan operasi aritmetik, penambahan mesti dilakukan untuk meningkatkan *program counter* dan mengira alamat. Perlaksanaan proses penambahan dengan pantas adalah satu kelebihan. Untuk memilih penambah (adder) yang sesuai, kajian dan analisis dilakukan terhadap jenis-jenis adder yang ada berdasarkan pembacaan dan rujukan. Beberapa penambah yang telah dikenalpasti melaksanakan penambahan dengan pantas adalah seperti yang berikut:

a) Penambah Pembawa Langkau (*skip carry adder*)

b) Penambah Pembawa Pemilih (*pembawa select adder*)

c) Penambah Pembawa Lihat depan (*carry lookahead adder*)

4.2.1 Penambah pembawa langkau (*skip carry adder*)

Penambah pembawa pemintas merupakan perantaraan di antara penambah ripple pembawa(*ripple carry*) dengan penambah lihat depan (*CLA Adder*).

Pengiraan *propagate(p)* adalah lebih mudah jika dibandingkan dengan *generate*. Penambah pembawa langkau hanya mengira *propagate(p)*. Pembawa akan *ripple* secara serentak melalui kesemua blok. Jika mana-mana blok menghasilkan *pembawa*, dan kemudian pembawa tersebut akan dikeluarkan daripada blok berkenaan, dan disetkan *true* walaupun *pembawa* berkenaan belum lagi dianggap betul.

Jika pada setiap permulaan operasi penambahan, *pembawa* yang masuk ke dalam blok adalah sifar, maka pembawa yang keluar dari blok tidak dihasilkan. Oleh itu, *pembawa* yang keluar daripada setiap blok boleh dianggap sebagai isyarat *generate (G)*. Apabila pembawa keluar dari blok LSB dihasilkan, ia bukan sahaja dihantar ke blok seterusnya, tetapi juga dihantar melalui get AND dengan isyarat *propagate* daripada blok seterusnya. Jika pembawa keluar dan isyarat P kedua-duanya adalah *true* kemudian pembawa ini akan memintas *(skip)* blok seterusnya dan bersedia untuk dihantar ke blok ketiga dan seterusnya. Penambah *pembawa* pintas ini hanya praktik jika pembawa dalam isyarat senang dihilangkan *(cleared)*.

Untuk menganalisa kelajuan penambah pembawa pemintas ini, anggapkan ia memerlukan 1 unit masa untuk isyarat itu melalui dua logic level. Kemudian ia akan mengambil katakan, k unit masa untuk pembawa melalui blok yang bersaiz k dan ia akan mengambil 1 unit masa untuk

pembawa memintas satu blok. Laluan yang terpanjang dalam penambah pembawa pemintas ini bermula dengan pembawa yang dihasilkan pada kedudukan 0. Jika penambah adalah n-bit kemudian ia perlukan k unit masa untuk melalui blok pertama, n/k-2 unit masa untuk memintas (skip) blok dan lebih dari k unit masa diperlukan untuk melalui blok terakhir.

4.2.2 Penambah Pembawa Pemilih (pembawa select adder)

Penambah pembawa pemilih ini beroperasi mengikut prinsip berikut. Dua penambahan dilakukan secara selari. Satu menganggap bahawa pembawa (carry) masuk adalah sifar dan yang lain menganggap pembawa masuk adalah satu. Apabila pembawa ini akhirnya diketahui, jumlah sebenar (yang mana telah dikira dahulu)dipilih. Penambah 8-bit misalnya akan dibahagikan kepada dua bahagian dan pembawa keluar daripada bahagian separuh bawah digunakan untuk memilih jumlah bit daripada separuh bahagian teratas. Setiap blok mengira jumlahnya menggunakan *rippling*, iaitu algoritma masa linear.

4.2.3 Penambah Pembawa Lihat depan (*carry lookahead adder*)

Untuk mengurangkan kelewatan yang disebabkan oleh perambatan bawaan (*carry propagation*) menerusi penambah biasa (contohnya *ripple-carry adder*), kita boleh menaksir samada pada setiap peringkat, pembawa masuk (carry-in) daripada stage sebelumnya akan mempunyai nilai 0 atau 1. Jika 'evaluation' yang betul dapat dilakukan dalam masa yang singkat, maka prestasi bagi penambah yang telah lengkap dilakukan boleh diperbaiki.

Fungsi bagi pembawa keluar (carry-out) untuk peringkat i boleh dikenali sebagai:

$$C_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

Jika faktor ini diungkapkan sebagai:-

$$C_{i+1} = x_i y_i + (x_i + y_i) c_i$$

Maka persamaan di atas boleh ditulis sebagai:

$$C_{i+1} = g_i + p_i c_i \qquad (4.2.3.1)$$

di mana $g_i = x_i y_i$

$p_i = x_i + y_i$

Fungsi g_i adalah bersamaan dengan 1 apabila kedua-dua input x_i dan y_i adalah sama dengan 1.Oleh sebab dalam peringkat (*stage*) ini stage i dijamin untuk menghasilkan satu pembawa keluar (*carry-out*), maka g dipanggil fungsi penghasil (*generate*).

Fungsi p_i adalah bersamaan dengan 1 apabila sekurang-kurangnya satu daripada input x_i dan y_i adalah bersamaan dengan 1. Dalam kes ini satu pembawa keluar (*carry-out*) akan dihasilkan jika C_i bersamaan dengan 1.
Kesan yang akan berlaku di sini adalah pembawa masuk (*carry-in*) bagi 1 akan disebarkan menerusi peringkat i. Dengan itu p_i dikenali sebagai fungsi penyebar (*propagate*).

Apabila ungkapan 1.1 dikembangkan kepada peringkat $_{i-1}$, ini akan memberikan:- $C_{i+1} = g_i + p_i (g_{i-1} + p\, i_{-1}\, c_{i-1})$

$= g_i + p_i\ g_{i-1} + p_i\, p\, i_{-1}\, c_{i-1}$ (setelah kurungan dikeluarkan)

Pengembangan yang sama untuk peringkat-peringkat yang lain, berakhir dengan peringkat 0 akan memberikan:

$C_{i+1} = g_i + p_i\ g_{i-1} + p_i\, p_{i-1}\, g_{i-2} + ..p_i\, p_{i-1}\, p_2\, p\, g_0 + p_i\, p_{i-1}... p_i\, p_0\, c_0$ (4.2.3.2)

Ungkapan yang disebutkan sebelum ini mewakili litar 2-peringkat (*level*) DAN-ATAU di mana C_{i+1} dapat dinilai dengan pantas. Penambah berdasarkan ungkapan ini dikenali sebagai penambah CLA.

4.3 Penambah yang dipilih: Penambah carry-lookahead

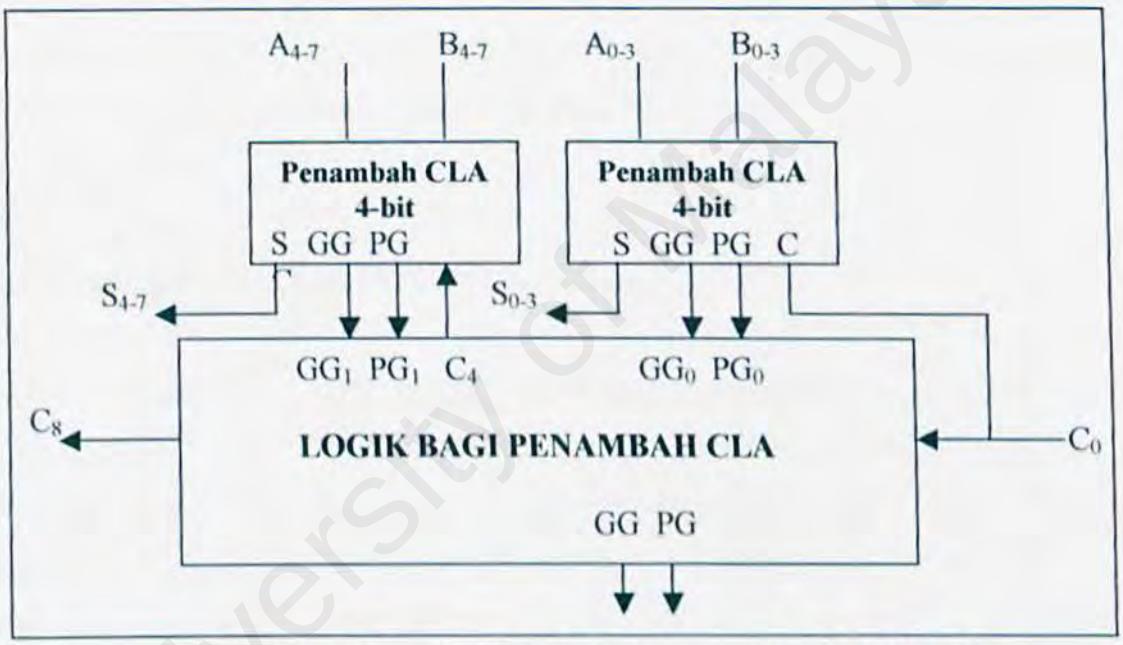
Penambah **carry-lookahead** dipilih kerana penambah jenis ini dapat mengurangkan kelambatan yang diakibatkan oleh perambatan bawaan (*carry propagation*) sekaligus dapat memberi kelajuan operasi aritmetik yang tinggi jika dibandingkan dengan penambah-penambah yang lain. Sebagai contoh, ripple adder berkelajuan rendah kerana terpaksa melalui laluan yang panjang bagi menyebarkan isyarat pembawa. [2].

Penambah carry-lookahead (CLA) dipilih di atas beberapa sebab yang berdasarkan kepada keseimbangan di antara saiz dan kelajuan. Bit pembawa *(Carry bit)* di dalam CLA dikira secara selari (*parallel*) kepada

operasi XOR pada bit-bit input (carry bit). Struktur selari (*parallel*) membenarkan input-input

dikira secara serentak dan ini merupakan perlaksanaan rekabentuk penambah yang berkelajuan tinggi.

Blok diagram am bagi penambah CLA yang dicadangkan ditunjukkan seperti di bawah:



Rajah 4.3 Gambarajah blok penambah CLA 8-bit

4.4 Coding untuk penambah carry lookahead

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity CLA_adder IS

PORT (a_in, b_in : IN std_logic_vector(7 downto 0);
        c_out : OUT std_logic_vector(7 downto 0));
end CLA_adder;

architecture fast_carry_behavior OF CLA_adder is

SIGNAL  sum        :  STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  g          :  STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  p          :  STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  c_internal :  STD_LOGIC_VECTOR(7 DOWNTO 1);
SIGNAL  carry_in   :  STD_LOGIC;

BEGIN
sum <= a_in XOR b_in;

--  Isyarat bagi Carry_generate (g) and carry_propagate (p)

g <= a_in AND b_in;
p <= a_in OR b_in;
```

```
-- Membolehkan penambah untuk 'look ahead'
-- and determine if carry signals will be required.

PROCESS (g,p,c_internal)
BEGIN
carry_in <= '0';
c_internal(1) <= g(0) OR (p(0) AND carry_in);

FOR i IN 1 TO 6 LOOP
    c_internal(i+1) <= g(i) OR (p(i) AND c_internal(i));
END LOOP;

END PROCESS;

-- Assign final values to c_out signal

c_out(0) <= sum(0) XOR carry_in;
c_out(7 DOWNTO 1) <= sum(7 DOWNTO 1) XOR c_internal(7 DOWNTO 1);

END fast_carry_behavior;
```

4.4.1 Penerangan bagi *coding* untuk penambah carry lookahead

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

--These statements provide access to the std_logic_1164 package, which defines the STD_LOGIC type

```
entity CLA_adder IS

PORT (a_in, b_in : IN std_logic_vector(7 downto 0);
    c_out : OUT std_logic_vector(7 downto 0));
end CLA_adder;

architecture fast_carry_behavior OF CLA_adder is

SIGNAL  sum       :  STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  g         :  STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  p         :  STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  c_internal :  STD_LOGIC_VECTOR(7 DOWNTO 1);
SIGNAL  carry_in  :  STD_LOGIC;

BEGIN
sum <= a_in XOR b_in;

--  Isyarat bagi Carry_generate (g) and carry_propagate (p)

g <= a_in AND b_in;
p <= a_in OR b_in;
```

--The form of entity declaration

As a multibit SIGNAL data object because a number is represented in VHDL code a a multibit SIIGNAL data object.

if input a_in and b_in=1, so g=1

If at least one of the input, a_in/b_in =1,so p=1

```
PROCESS (g,p,c_internal)                    --Allowing the adder to look ahead
BEGIN                                       --determine if carry signals will be required
carry_in <= '0';
c_internal(1) <= g(0) OR (p(0) AND carry_in);

FOR i IN 1 TO 6 LOOP
    c_internal(i+1) <= g(i) OR (p(i) AND c_internal(i));
END LOOP;

END PROCESS;

-- Assign final values to c_out signal

c_out(0) <= sum(0) XOR carry_in;
c_out(7 DOWNTO 1) <= sum(7 DOWNTO 1) XOR c_internal(7 DOWNTO 1);

END fast_carry_behavior;
```

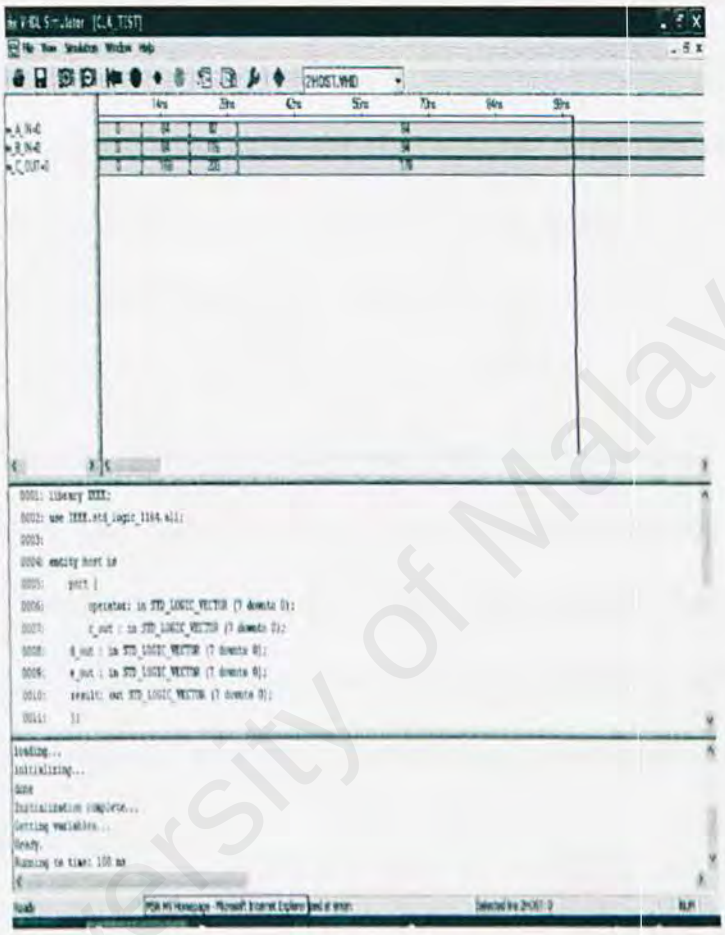
4.5 Coding untuk testbench

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

entity testbench_CLA_adder is
end testbench_CLA_adder;

architecture behavioral of testbench_CLA_adder is

component CLA_adder
    port(A_IN   : IN std_logic_vector (7 downto 0);
         B_IN   : IN std_logic_vector (7 downto 0);
         C_OUT  : OUT std_logic_vector (7 downto 0));

end component;


constant PERIOD : time := 10 ns;

signal w_A_IN    : std_logic_vector ( 7 downto 0);
signal w_B_IN    : std_logic_vector ( 7 downto 0);
signal w_C_OUT   : std_logic_vector ( 7 downto 0);

begin

DUT : CLA_adder port map (
            A_IN   => W_A_IN,
            B_IN   => W_B_IN,
            C_OUT  => W_C_OUT
```

```
    );


STIMULI : process
begin
 w_A_IN   <=  (others => '0');
 w_B_IN   <=  (others => '0');
 wait for PERIOD;                          --8-bit inputs
 w_A_IN   <=  "01010100";
 w_B_IN   <=  "01010100";
 wait for PERIOD;
 w_A_IN   <=  "01010111";
 w_B_IN   <=  "01110100";
 wait for PERIOD;
 w_A_IN   <=  "01010100";
 w_B_IN   <=  "01011110";

 wait for PERIOD;


 wait;
 end process STIMULI;

 end behavioral;


 configuration CFG_TB_CLA_adder of testbench_CLA_adder is
 for behavioral
 end for;

 end CFG_TB_CLA_adder;
```

BAB 05 Pengujian Litar

Pengujian litar bermula apabila proses pengekodan dan *testbench* selesai dilaksanakan ke atas litar penambah. Setelah dipastikan tiada ralat dalam pengkompilan, proses simulasi litar penambah dilakukan. Rajah simulasi ditunjukkan seperti di bawah:



Rajah 5.1 Simulasi bagi penambah *carry-lookahead*

Sebagai contoh, pada selang masa 14ns, input yang dimasukkan adalah:

A_in: 84 (Dalam binari = 01010100)

B_in: 84 (Dalam binari = 01010100)

Output yang dikeluarkan adalah:

c_out: 168 # Maka penambah CLA berjaya dilaksanakan.

BAB 06 PERBINCANGAN

6.1 Perbincangan

Penyediaan jurnal bagi RNS yang diminta oleh moderator telah selesai dan dihantar pada pertengahan bulan Januari. Ini adalah bertepatan dengan kehendak moderator yang menetapkan syarat penghantaran jurnal adalah sebelum Viva 3182 dijalankan. Penyediaan jurnal dilakukan oleh ketiga-tiga ahli kumpulan dan digabungkan pada peringkat akhir. Kajian dan penyelidikan berkaitan RNS telah dilakukan di Perpustakaan Utama, Perpustakaan Kejuruteraan, Bilik Dokumen FSKTM dan juga melalui internet. Jurnal telah diletakkan pada appendix. Tajuk jurnal adalah ***The Explanation of Residue Number System and Arithmetic Circuits.*** Penghantaran jurnal yang telah siap dicetak dan *soft copy* telah dilakukan ke email dan pigeon hole moderator. Manakala senarai jurnal-jurnal rujukan pula dihantar kemudian. Diharapkan agar moderator berpuas hati dengan hasil jurnal yang dilakukan.

Penghasilan modul penambah CLA telah berjaya dilakukan sehingga ke tahap simulasi. Namun begitu, masalah untuk menggabungkan ketiga-tiga modul telah wujud. Ini adalah kerana ralat pengekodan yang wujud pada *coding* tidak dapat diperbetulkan. Maka modul-modul hanya berjaya dipersembahkan secara berasingan sahaja. Masalah wujud pada modul ahli kumpulan yang lain, yang berjaya disiapkan pada saat akhir menyebabkan kesuntukan masa untuk

berbincang bagi memperbetulkan ralat pada *coding* penggabungan kesemua modul.

Selain daripada itu, kekangan masa yang dihadapi juga berpunca daripada semua ahli menunggu untuk tempoh masa yang agak lama untuk mendapatkan makmal bagi membangunkan litar. Walaupun permintaan dilakukan pada peringkat awal, namun kira-kira tiga minggu terakhir barulah setiap ahli dibenarkan menggunakan komputer di Makmal Teknologi Komputer. Namun pembangunan litar tetap dilaksanakan, tanpa menyalahkan mana-mana pihak. Ini adalah kerana setiap ahli tidak mahu memanjangkan sesuatu perkara yang telah berlaku. Yang paling penting, pembangunan litar tetap dilaksanakan.

Penambah CLA 8-bit telah berjaya dihasilkan. Walau bagaimanapun, di atas faktor masa juga telah menyebabkan penambah CLA 8-bit sahaja yang selesai dilakukan. Usaha untuk menglobalkan input bagi membenarkan sebarang bit masukan tidak berjaya dilaksanakan akibat daripada kesuntukan masa bagi menyiapkan litar, laporan, jurnal dan bersedia untuk viva.

BAB 07 KESIMPULAN

Pembinaan litar penambah CLA menggunakan metodologi VHDL dilaksanakan di mana sepatutnya pada peringkat terakhir penggabungan penambah (adder) dan pendarab (multiplier) dalam litar aritmetik reja akan dilaksanakan dan hasilnya dibandingkan dengan pengesan ralat bagi membentuk litar aritmetik reja. Metodologi VHDl digunakan kerana VHDL amat berkesan untukmengimplementasikan FPGA *(Field Programmable Gate Array)* ke teknologi ASIC (*applications specific integrated circuit*).

Matlamat merekabentuk penambah CLA direalisasikan dengan menggunakan perisian PEAKVHDL . Pengekodan dan testbench berjaya dilaksanakan tanpa ralat dan penambah telah berjaya disimulasikan.

Rekabentuk litar penambah 8-bit menggunakan penambah Carry Lookahead (CLA) , yang merupakan penambah berkelajuan tinggi berjaya dilakukan. Penghasilan jurnal di atas permintaan moderator juga berjaya dilakukan dan dihantar pada tempoh masa yang telah ditetapkan.

SENARAI RUJUKAN

Brown, S., Vranesic, Z. (2000), "*Fundamental of Digital Logic With VHDL Design.*" Singapore : Mc Graw-Hill

Chin Wee Yoon (1994) *A Reformulation of The Residue Number System.Degree Thesis.* Universiti Malaya.

Ibrahim B Ahmat.(2000) *Penambahan dan Penolakan Nombor.* Tesis Sarjana,Universiti Malaya.

J. Gosling, "*Design of Arithmetic Units for Digital Computers*" ,University of Manchester: Mac Millan.

M.Dugdale, (1990) " *VLSI Implementation of Residue Adders Based On Binary Adders*",BILCON 90.

N. S. Szabo and R.I. Tanaka (1967), "Residue Arithmetic and Its Applications to Computer Technology", New York : Mc Graw-Hill.

S. Wei and K. Shimizu, "*Error Detection of Arithmetic Circuits Using A Residue Checker With Signed-Digit Number System*", Journal of IEEE International Symposium on Defect And Fault Tolerence in VLSI Systems (DFT'01).