A PIPELINED MULTIPLIER ACCUMULATOR FOCUSING ON MULTIPLIER

By

NORFADILAH KHALIL

WEK000276

Projek tahap akhir ini adalah untuk diserahkan kepada
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Malaya
Bagi memenuhi keperluan Ijazah Sarjana Muda
Sains Komputer

Session 2002/2003

A PIPELINED MULTIPLIER ACCUMULATOR
FOCUSING ON PIPELINE ARCHITECTURE

BY:

NORFADILAH KHALIL

DEPARTMENT OF SYSTEM AND COMPUTER TECHNOLOGY

SESSION 2002/2003

FACULTY OF COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA

Abstrak

Pipelined MAC adalah singkatan bagi ***Pipelined Multiplier-Accumulator***. Ia direka untuk meningkatkan masa larian bagi dan mengurangkan lengahan pada Conventional *MAC*. Disamping itu, dengan pengguanaan *pipeline* dalam pembinaan MAC, masa dan kos dapat dikurangkan. Dalam *Pipeline MAC,* proses pendaraban dan penambahan (*accumulator*) yang akan mengimplementasikan *pipeline.* Dengan pendekatan ini, kitar masa dapat dikurangkan tetapi, jumlah masa untuk operasi pendaraban masih belum dapat dikurangkan. Dalam merialisasikan tujuan utama pembinaan *Pipeline MAC* ini, beberapa pendekatan serta algorithma dikaji. Algorithma Booth di gunakan untuk melaksanakan beberapa proses pendaraban dalam suatu masa. Sementara kaedah penambahan *Carry Look Ahead* (CLA) dipercayai akan meningkatkan perlaksanaan bagi memproses input (8-bit data). Teknik pendekatan *pipeline* pula dikaji untuk digunakan bagi tujuan meningkatkan kelajuan oeprasi dan mengurangkan kitar masa. Dengan pendekatan *pipeline* dalam rekabentuk projek ini, diyakini bahawa lengahan dalam Conventional *MAC* dapat dielakkan.

Penghargaan

Syukur Alhamdulillah dengan berkat limpah kurniaNya dapat juga saya menyiapkan kursus ini. Dikesempatan ini, saya ingin mengucapkan ribuan terima kasih kepada En Mohd Yamani Idna selaku penyelaras saya sepanjang kursus ini. Terima kasih tidak terhingga juga buat moderator, En Zaidi Razak yang banyak mencetuskan idea-idea bernas dalam membantu memajukan sistem ini.

Buat kedua ibubapa saya, jasa kalian tidak dapat saya lupakan. Tanpa kalian, saya tidak mungkin sampai ke tahap ini. Serta buat rakan sepasukan, Laili Erni dan Rabiatul Adawiyah, kerjasama kalian dalam membantu menyelesaikan modul ini amat-amatlah dihargai.

Tidak lupa kepada semua pensyarah dan pelajar Fakulti Sains Komputer dan Teknologi Maklumat, Universiti Malaya yang telah banyak memberikan dorongan dan bimbingan sepanjang saya bergelar pelajar di fakulti ini.

Akhir sekali, buat fakulti tercinta, Fakulti Sains Komputer dan Teknologi Maklumat,Universiti Malaya, terima kasih atas segala panduan yang diberikan dalam membantu saya melengkapkan kursus ini.

Sekian, Terima Kasih.

Norfadilah bt. Khalil
WEK 000276

Isi Kandungan

2.0 Kajian Literasi

4.0 Rekabentuk Sistem

5.0 Rekabentuk Sistem

Senarai Gambarajah

Senarai Jadual

BAB SATU

PENGENALAN

1.0 Pengenalan

1.1 *Pipelined Multiplier-Accumulator*

Pipelined MAC merupakan singkatan bagi *Pipelined Multiplier Accumulator*. Tujuan utama pembinaan sistem ini adalah untuk mengurangkan lengahan yang terjadi dalam MAC tanpa talian paip. Punca utama penerapan *pipeline* ke dalam sistem MAC adalah untuk meningkatkan masa larian dan pendekatan baru ini boleh digunakan dalam algoritma pemprosesan isyarat digital seperti penapisan, penyahmodulasi dan penyamaan dimana MAC boleh memisahkan isyarat data yang dikehendaki dengan hingar. *Pipelined MAC* melibatkan perwakilan data dimana dalam pengiraannya, *Pipelined MAC* melibatkan nombor-nombor perpuluhan bagi meningkatkan ketepatan.

Input dalam *Pipelined MAC* adalah merupakan 8-bit *fixed*-point sementara output yang akan dijana oleh *Pipelined MAC* ialah 16-bit *fixed-point*. *Pipelined MAC* beroperasi dengan nombor-nombor kompleks termasuklah keatas nombor-nombor dengan titik perpuluhan yang dihasilkan oleh eksponen negatif bagi nombor binari. Penyelesaian pendaraban terdahulu akan diumpuk kepada nilai akhir keputusan.

Masalah utama yang dihadapi oleh MAC ialah lengahan.

Dalam MAC juga, penggunaan algoritma untuk melaksanakan operasi pendaraban juga kurang efisyen. Ini berjaya meningkatkan lagi jumlah masa untuk melaksanakan operasi pendaraban disamping menyumbang kepada lengahan masa larian. Jadi, untuk mengumpukkan nilai semua produk separa yang dijana oleh pendarab, masa yang lebih lama diambil.

Masalah yang dialami semasa proses pengumpukkan pula ialah limpahan (*overflow*) data. Dalam MAC, tiada operasi untuk mengawal limpahan data.

Jadi, secara keseluruhannya, untuk melarikan satu arahan lengkap, MAC memerlukan 3 kitar masa sebelum CPU atau Unit Pemprosesan Pusat dapat mengumpukkan semua produk yang terhasil daripada operasi pendaraban. CPU bukan sahaja membazir masa tetapi ianya tidak dapat bekerja dengan lebif efisyen.

1.2 Skop

Skop permasalahan yang perlu diselesaikan ialah mengurangkan lengahan. Untuk mengurangkan lengahan ini, pelbagai cara dikaji. Pengurangan lengah dalam masa larian adalah merupakan kejayaan yang besar kerana dengan pengurangan lengah, kerja dapat dilaksanakan dengan lebih cepat.

Pengurangan kitar masa juga harus dititik-beratkan bagi mengelakkan CPU tidak berfungsi dengan efisyen. Dengan mengurangkan kitar masa, CPU tidak perlu menunggu kerja dan masa CPU tidak terbazir begitu sahaja.

Pengawalan limpahan juga amatlah penting. Ini kerana sekiranya nilai output sudah melibihi julat limpahan, maka, dikhuatiri, pengiraan dalam MAC akan menghasilkan ralat yang tidak diingini. Dengan penghasilan ralat, ketepatan dalam MAC diragui dan faktor inilah yang patut dielakkan dalam membentuk satu unit yang tepat dan cekap dalam melaksanakan tugasnya.

Bab 1 dalam projek *Pipeline Multiplier Accumulator* ini menerangkan tentang pengenalan projek, jenis-jenis masalah yang dihadapi oleh sistem terdahulu; yang perlu diselesaikan oleh sistem ini serta objektif dan tujuan perlaksanaan projek ini. Jadual perancangan perlaksanaan projek juga disertakan sebagai memberikan gambaran tentang perjalan projek dari peringkat permulaan sehingga ke peringkat akhir projek ini.

Bab 2 pula menerangkan tentang semua kajian yang dijalankan bagi memilih algoritma yang terbaik; yang bersesuaian dan mencapai matlamat projek ini sepenuhnya. Aktiviti penganalisaan serta pencarian sistem-sistem serta sumber-sumber yang berkaitan dijalankan dan di bentangkan dalah bab ini.

Bab 3 pula menceritakan dengan lebih spesifik semua metodologi, keperluan dalam memilih perkakasan serta nahasa yang digunakan. Perbandingan dibuat bagi memilih metodologi yang terbaik untuk diimplementasikan dalam rekabentuk projek nanti. Perbandingan cip antara ASIC dan FPGA dibuat untuk memilih perkakasan yang terbaik serta kos-efektif bagi menyokong keperluan projek pada masa kini dan pada masa akan datang.

Bab 4 pula menjelaskan dengan lebih lanjut tentang fungsi sebenar *Pipeline MAC*, modul-modul yang terlibat serta integrasi dan cara penyambungan antara modul. Koordinasi dan kesinambungan antara modul-modul dalam projek ini seperti pendaraban, penambahan serta *pipeline* diterangkan. Fungsi bagi setiap modul diperjelaskan.

1.3 Objektif dan Tujuan

Seperti yang dinyatakan, objektif utama pembinaan *Pipeline* MAC adalah untuk mengurangkan lengahan dalam Conventional *MAC*.

Untuk memenuhi matlamat projek ini, berbagai cara dikaji dan diteliti. Pendekatan *pipeline* yang digunakan dalam MAC dipercayai dapat mengurangkan lengahan. Sementara Algoritma Booth dan kaedah penambahan *Carry Look Ahead* akan digunakan bagi mempercepatkan operasi pendaraban serta penambahan.

Konklusinya, tujuan perlaksanaan projek *Pipeline Multiplier Accumulator* ini adalah untuk mengurangkan lengahan, meningkatkan operasi pendaraban dan penambahan serta boleh dilaksanakan dalam nombor kompleks.

1.4 Penjadualan Projek

Jadual dibawah menunjukkan tentang penjadualan projek dari fasa permulaan hingga ke peringkat penghujung projek.

PERKARA	JUN	JULAI	OGOS	SEPTEMBER	OKTOBER	NOVEMBER	DISEMBER	JANUARI
Kajian Mengenai Projek								
Analisa Sistem Terdahulu								
Rekabentuk Sistem								
Simulasi dan Implementasi								
Sokongan dan Penyenggaraan								

Jadual 1.0 : Penjadualan Projek

Seperti yang ditunjukkan oleh jadual 1.0 , projek ini bermula pada awal bulan Jun 2002. Pada fasa permulaan projek, kajian untuk meningkatkan pemahaman serta persediaan mengenai projek dijalankan.

Bermula pada bulan Julai sehingga penghujung Ogos, analisa sistem-sistem terdahulu dijalankan.

Aktiviti merekabentuk sistem bermula pada bulan Ogos sehingga September.

Bab akan datang akan memperihalkan tentang kajian literasi yang telah dijalankan ke atas projek-projek serta sistem-sistem terdahulu. Dalam bab seterusnya, semua algoritma yang dipertimbangkan akan disenaraikan dan akan diterangkan dengan lebih spesifik. Pemilihan algoritma yang dianggap paling sesuai juga akan dilaksanakan dan sebab-sebab pemilihan akan dinyatakan.

BAB DUA

KAJIAN LITERASI

2.0 Kajian Literasi

2.1 Pengenalan

Dalam bab ini, beberapa kaedah dan pendekatan telah dikenalpasti untuk digunakan dalam sistem *Pipelined MAC*.

Semasa proses rekabentuk sistem, beberapa kaedah pembinaan bagi penambahan dan pendaraban telah dikaji dan dibandingkan sebelum teknik yang paling sesuai dipilih berdasarkan beberapa kriteria yang ditetapkan oleh sistem. Faktor utama dalam isu pemilihan algorithma dan tekn ik yang sesuai ini adalah kelajuan operasi dan senibina atau kaedah yang termudah untuk dilaksanakan.

Terdapat lima algorithma yang dikenalpasti untuk digunakan dalam implementasi rekabentuk bagi unit pendaraban iaitu:

1. *Add-Shift Sequential Multiplication*
2. *2's Complement Arrays Multiplication Technique*
3. *Booth Algorithm*
4. *Bit-Pair Recording*
5. *Wallace Tree*

Didalam *accumulator* pula, beberapa algorithma untuk kesegerakkan penambahan dikaji. Antara algorithma yang dikaji adalah:

1. *Ripple-Carry*
2. *Carry-Select*
3. *Conditional Sum*
4. *Carry-LookAhead*

merupakan antara algorithma yang akan dipilih. Teknik untuk meningkatkan kepantasan keupayaan bagi setiap penambah merupakan kriteria utama dalam pemilihan algorithma bagi operasi penambahan.

Penggunaan talian paip (*pipeline*) dalam MAC adalah untuk mengurangkan lengahan yang terjadi akibat daripada menggunakan banyak kitar masa hanya untuk melaksnakan operasi pendaraban.

Pipeline merupakan suatu teknik dimana perlaksanaan berbilang arahan ditindihkan diantara satu sama lain. Ini bertujuan untuk mengurangkan purata kitaran jam bagi setiap arahan. Pertindihan arahan ini membolehkan arahan baru bermula dengan kitaran baru tanpa perlu menunggu arahan terdahulu daripadanya selesai.

Pemilihan teknik *pipeline* ini akan dibandingkan dengan perlaksanaan arahan secara skalar akan diterangkan dengan lebih terperinci pada bahagian akan datang.

2.2 Implementasi *Accumulator*

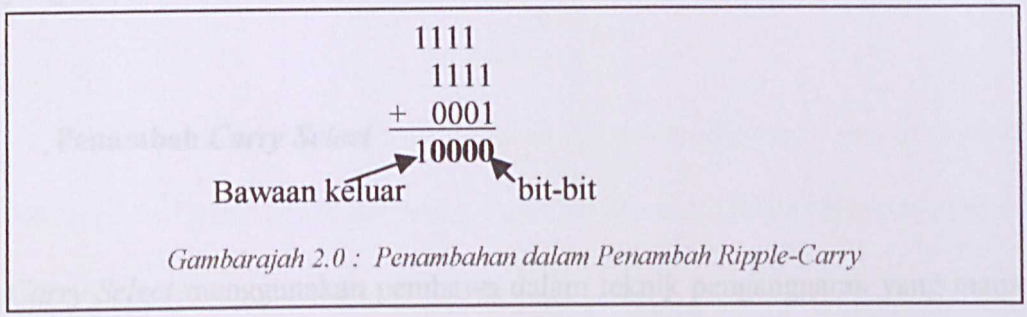
2.2.1 Rekabentuk *Accumulator*

Untuk keperluan rekabnetuk, dimana jenis *pipeline* yang digunakan adalah jenis *pipeline* yang segerak dalam. Maka, penambah yang dipertimbangkan untuk diimplementasikan adalah juga merupakan penambah segerak. Antara penambah yang seringkali digunakan ialah penambah *Ripple Carry*, *Carry Select*, *Conditional Sum* dan *Carry LookAhead* adders. Dalam pemilihan kaedah yang tepat, pemahaman mengenai teknik kelajuan pembawaan adalah satu kewajipan. Jadi, kajian mengenai organisasi perkakasan penambah tersebut diperlukan. Perbandingan antara penambah *Carry LookAhead* (CLA) dengan algorithma lain bersesuaian dengan keperluan sistem. Perbandingan tentang semua jenis penambah ini akan diterangkan dengan lebih jelas lagi pada bahagian-bahagian dalam bab ini.

2.2.2 Perbandingan Kaedah Penambahan

2.2.2.1 Penambahan *Ripple Carry*

Penambah dipanggil *Ripple Carry* kerana keluaran akhir *pipeline MAC* adalah bergantung kepada pembawa yang dijana oleh setiap pasangan bit sebelumnya. Jadi, jumlah bagi *most significant* bit hanyalah sesudah isyarat pembawa masuk secara melalui penambah dari *least significant bit* ke *most significant bit* .



Gambarajah 2.0 : Penambahan dalam Penambah Ripple-Carry

Dalam penambahan *Ripple-Carry* ini, isyarat pembawa akan dijana pada setiap kali penambahan dilakukan. Ini menyebabkan isyarat akan berpindah melalui semua aras pada operasi penambahan.

Operasi bagi kaedah ini hanya sah selepas 2(N-1) +1 lengahan get, *N* ialah bilangan bit. Bit bawaan keluar sah selepas *2N* lengah get. Lengahan ini mungkin akan menambahkan lengah yang terjadi dalam sambungan modul-modul dalam *Pipeline MAC.*

Kelemahan kaedah ini ialah ia akan melambatkan operasi sekiranya banyak bit yang perlu untuk ditambah.Sekiranya 32-bit diperlukan untuk melaksanakan operasi penambahan, anggap bahawa lengah get adalah 1ns, keseluruhannya, lengah bagi operasi ialah 63ns.

Kesimpulannya, penambah *Ripple Carry* adalah perlahan kerana mengalami lengahan penyebaran yang besar dalam operasinya. Impaknya, ia akan mengakibatkan lengahan

dalam operasi, walaupun begitu, implementasinya adalah lebih mudah berbanding penambah yang lain.

2.2.2.2 Penambah *Carry Select*

Penambah *Carry Select* menggunakan pembawa dalam teknik penganggaran yang mana akan meningkatkan kelajuan operasi.Teknik ini diimplementasikan bagi setiap bahagian bagi kumpulan empat bit atau bahagian penambah yang mengandungi rekabentuk yang sama. Struktur dalaman adalah sama dengan penambah *Ripple Carry* dimana pembawa kumpulan dijana.

Dalam operasinya, bagi setiap bahagian, dua penjumlahan akan dijana secara berjujukan. Ini akan menganggap pembawa dalam bahagiamn tersebut ialah sifar ataupun satu. Pemilihan penjumlahan akan dilaksanakan oleh litar pemultipleks. Pembinaan bagi struktur penambah jenis ini adalah kompleks kerana pemilihan menggunakan litar pemultipleks. Lagipun, lengahan penyebaran adalah tinggi berbanding penambah *Carry Look Ahead.*

2.1.2.2.3 Penambah *Carry Look Ahead*

Teknik *Carry Look Ahead* menyelesaikan masalah lengahan dengan mengira isyarat pembawa, berdasarkan isyarat input. Ini berdasarkan kepada dua fakta yang mana isyarat pembawa akan dijana iaitu:

1. Apabila kedua-dua input adalah bernilai 1.
2. Apabila salah satu daripada input adalah satu dan bawaan-masuk bagi penambahan sebelumnya adalah 1.

Jadi, penambahan dalam teknik ini boleh diwakili dengan:

$$C_{out} = C_{i+1} = A_i . B_i + (A_i \$ B_i) . C_i$$

Dimana $ adalah operasi *exclusive OR*. Ianya boleh juga diwakili dengan:

$$C_{i+1} = B_i + P_i . C_i$$

dimana

$$G_i = A_i . B_i$$

Adalah bersamaan dengan

$$P_i = (A_i \$ B_i)$$

Dan dipanggil sebagai *Generate* dan *Propagate*.

Penjumlahan boleh dijana selepas satu lengahan tambahan get. Kelebihan teknik ini ialah lengahan pada bit adalah tidak bergantung pada satu sama lain.

Kelemahan bagi teknik ini ialah *Carry Look Ahead* akan bertambah kompleks apabila pengiraan yang melibatkan 4 bit dilaksanakan. Oleh sebab itulah, penambah *Carry Look Ahead* selalu digunakan untuk mengimplementasikan modul yang mempunyai 4-bit dan boleh dilaksanakan dalam struktur hierarki kerana penambah tersebut hanya menyokong 4 bit sahaja. Gambarajah mengenai penambah logik mengenai *Carry Look Ahead* disertakan dalam lampiran 1.

2.3 Implementasi Pendaraban

Dalam *pipelined MAC*, pendarab memainkan peranan penting untuk meningkatkan kelajuan bagi unit. Dalam pemprosesan digital, kelajuan proses adalah perkara yang paling penting. Oleh itu, teknik pendaraban yang bakal digunakan mestilah boleh meningkatkan kelajuan operasi.

Pendaraban didefinisikan sebagai penambahan berulang. Nombor-nombor yang akan ditambah adalah *multiplicand* dan bilangan kali nombor ditambah dipanggil *multiplier* untuk menghasilkan output ataupun *product*. Untuk setiap operasi penambahan, *partial product* akan dijana.

Dalam pendaraban, lima algorithma dikaji dan dipertimbangkan untuk diimplementasikan dalam rekabentuk. Setiap algorithma mempunyai kelebihan dan kekurangan masing-masing.

Algorithma Booth, *Wallace Tree* dan *Bit-Pair Recording* merupakan teknik-teknik yang telah dipertimbangkan untuk digunakan dalam merekabentuk sistem.

2.2.1 Perbandingan Beberapa Kaedah dalam Pendaraban

Dalam ***Add-Shift Sequential Multiplication***, dua n-bit nombor didarabkan, *multiplicand* akan disimpan dalam pendaftar n-bit dan *partial product* dengan *multiplier* akan disimpan dalam pendaftar 2n-bit.

Kaedah ini hanya boleh digunakan dengan pendarab yang bernilai positif. Sekiranya pendarab negatif digunakan, kedua-dua *multiplicands* mestilah dalam pelengkap duaan sebelum operasi pendaraban dilaksanakan.

Kaedah ini akan menjadi lebih kompleks apabila melaksanakan pendaraban negatif kerana ia perlu menukarkan nombor dalam binarikepada pelengkap duaan dahulu. Algorithma ini mestilah sesuai dengan litar untuk melaksanakan tugasnya, dalam kata lain, teknik ini memerlukan pengawalan logik yang lebih kompleks. Untuk melaksanakan fungsinya, litar haruslah diperbetulkan untuk melarikan operasi dengan pendarab negatif.

Teknik ini kurang fleksibel kerana litar haruslah ditukar setiap kali melarikan kerja-kerja tertentu. Oleh itu, ia mempunyai potensi yang kurang dalam meningkatkan kelajuan seluruh sistem.

Teknik ***2's Complement Modular Arrays Multiplication*** menggunakan algorithma pendaraban yang umum dengan membahagikan seluruh operasi kepada beberapa modul yang berbeza. Setiap modul akan melaksanakan pendaraban setempat dimana operasi tersebut adalah tidak berkaitan dengan operasi bagi modul-modul lain.

Untuk mendapatkan produk akhir, teknik ini menggunakan kaedah penambahan *Carry-Save* dan *Carry Look-Ahead* untuk mengira jumlah bagi keseluruhan output bagi semua modul.

Tetapi, teknik ini memerlukan lebih masa untuk melaksanakan operasi pendaraban dalaman; menyebabkan operasi keseluruhan sistem akan bertambah lambat. Lagipun, untuk menggunakan teknik ini, kelemahan utamanya ialah kekompleksan serta saiznya yang agak besar.

2.2.2 Algorithma Booth

Algorithma Booth adalah algorithma yang baik dan sesuai digunakan untuk operasi pendaraban bagi *signed-number*. Ia menjana 2n-bit produk dengan input sebanyak n-bit. Algorithma ini merupakan suatu algorithma terus kerana ia melaksanakan operasi untuk nombor-nombor positif dan nombor-nombor negatif dengan cara yang sama.

Dengan menggunakan teknik ini, proses pendaraban dengan data yang mempunyai jujukan 0 yang banyak dapat dilajukan. Jadi, sekiranya pendarab mengandungi bit jujukan 1, ianya akan ditukarkan kepada bentuk dimana dengan bit jujukan 0 yang banyak. Dengan terhasilnya banyak jujukan 0, masa larian yang bergantung kepada operasi pendarab dengan nilai bit 1 dapat dikurangkan.

Seperti contoh :

```
          0 1 0 0 0 0 0 0   (64)
-         0 0 0 0 0-1 0 0   ( 4)
_____________________________
          0 0 1 1 1 1 0 0   (60)
_____________________________
```

Jadi, nombor 0 0 1 1 1 1 0 0 boleh juga ditulis sebagai

0 1 0 0 0 0 -1 0 0.

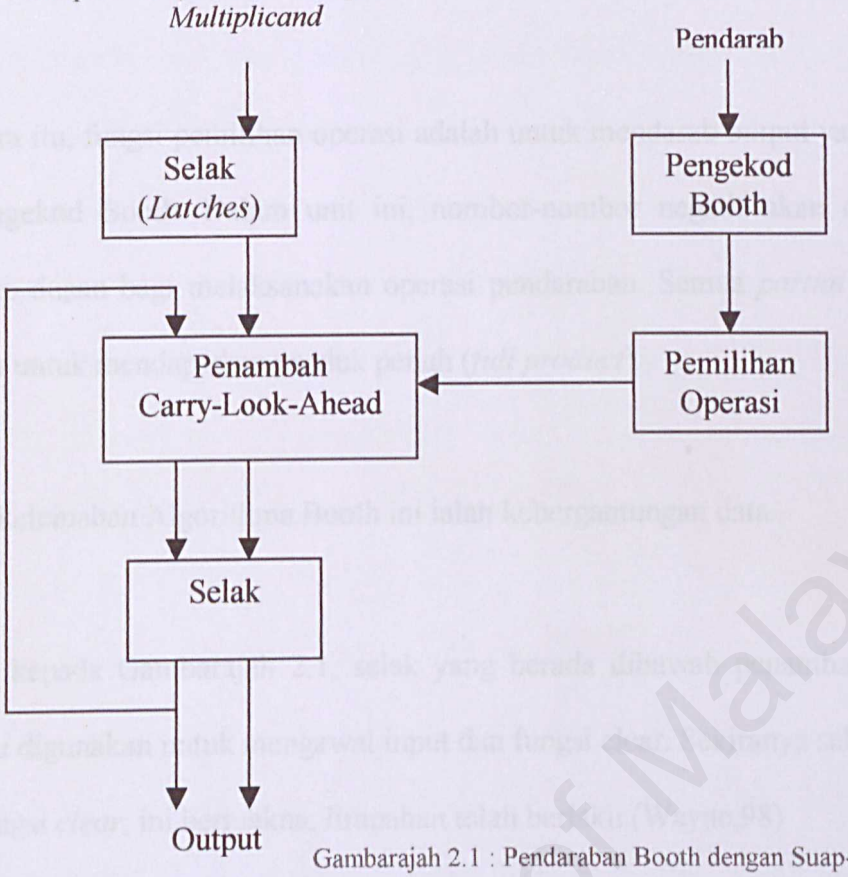
Dengan menggunakan Algorithma Booth, anggap *multiplicand* ialah 0 0 0 0 0 0 1 1 (+3) dan pendarab ialah 0 1 0 0 0 0 -1 0 0 (+60),

```
                0 0 0 0 0 0 1 1   ( 3)
X               0 1 0 0 0-1 0 0   ( 60)
_____________________________________

  1 1 1 1 1 1 1 1 1 1 1 1 1 0 1       -> dalam pelengkap duaan kerana bawaan
                                        nombor negatif
  0 0 0 0 0 0 0 0 1 1
_____________________________________

  0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0     1 ->bawaan
_____________________________________
```

Datagram 2.0 : Contoh Pengiraan Algorithma Booth

Kelajuan algorithma ini bergantung kepada bilangan bit 1 dalam pendarab. Sekiranya multiplier mempunyai bit dengan bit 1 dan bit 0 diselang selikan, ini merupakan keadaan *worst-case* dan Algorithma Booth mungkin tidak dapat beroperasi dengan baik.

Contoh, 0 1 0 1 0 1 0 1 .



Gambarajah 2.1 : Pendaraban Booth dengan Suap-balik.

Multiplicand digunakan dalam operasi pendaraban sementara pendarab didefinisikan sebagai bilangan kali *multiplicand* didarabkan.

Selak atau *latches* digunakan untuk mengawal bila dan bagaimana input disimpan buat sementara waktu didalam sistem. Semasa tepian menaik (*rising edge*), selak akan dibuka untuk masukkan input data. Selak dikawal oleh *clock*.

Pengekod Booth digunakan untuk menukarkan jujukan bit 1 dalam pendarab menjadi jujukan bit 0 dalam melaksanakan operasi pendaraban. Dengan adanya unit ini, operasi pendaraban dapat dilajukan.

Sementara itu, fungsi pemilihan operasi adalah untuk mendarab output yang dikeluarkan oleh Pengekod Booth. Dalam unit ini, nombor-nombor negatif akan ditukar kepada pelengkap duaan bagi melaksanakan operasi pendaraban. Semua *partial products* akan ditambah untuk mendapatkan produk penuh (*full product*).

Namun, kelemahan Algorithma Booth ini ialah kebergantungan data.

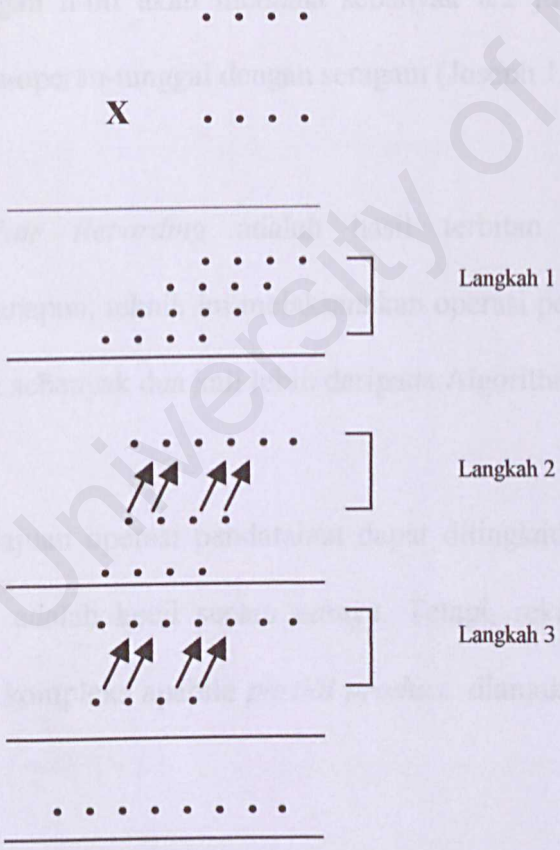
Merujuk kepada Gambarajah 2.1, selak yang berada dibawah penambah *Carry-Look-Ahead* itu digunakan untuk mengawal input dan fungsi clear. Sekiranya selak menyatakan status fungsi *clear*, ini bermakna, limpahan telah berlaku.(Wayne,98)

2.2.3 Wallace Tree

Prinsip bagi pendaraban *Wallace tree* ialah untuk melaksanakan pendaraban yang boleh memperluaskan dan memperbesarkan kepanjangan perkataan.

Prinsip ini ditunjukkan pada Gambarajah 2.2. Jelas disitu menunjukkan bahawa jumlah akhir adalah hasil penambahan 16 *partial products.* Langkah pertama dalam proses penambahan melibatkan pengumpulan (*grouping*) *partial products* kepada 3 set. Katakana baris *P* ialah *partial product*, sebanyak 3* ([p/3]) baris yang telah dikumpulkan dan sebanyak *p* mod 3 baris tidak berubah dan akan melalui langkah seterusnya.

Keputusan bagi penjumlahan isyarat dari penambahan separa dan penuh juga akan melalui langkah berikutnya. Ini akhirnya akan menghasilkan 3 baris *partial product* yang perlu dijumlahkan. Penjumlahan dan isyarat pembawa akan dihantar pada langkah ke 3. Oleh kerana hanya ada 2 baris *partial product* yang perlu dijumlahkan, langkah ini boleh diimplementasikan dengan menggunakan *carry-propagate* yang laju. Senibina *Wallace Tree* hanya memerlukan 5 penambahan penuh dan 3 penambahan separa bagi mendapatkan hasil akhir dengan penambahan 16 *partial product*.



Gambarajah 2.2 : Operasi Pendaraban dalam Wallace tree

Namun begitu, ada beberapa kelemahan teknik ini. *Wallace Tree* tidak sesuai bagi rekabentuk yang menyokong skala besar. Ia juga mempunyai masa yang terhad dan agak sukar untuk diimplementasikan dalam *pipeline*. Ini seterusnya akan meninggalkan kesan kepada masa *throughput* (masa pindahan bagi data).

2.2.4 *Bit-Pair Recording*

Bit-Pair Recording merupakan teknik pendaraban yang laju dengan jaminan bahaw pendarab dengan n-bit akan menjana sebanyak n/2 jumlah akhir dan akhirnya akan menangani kes-operan-tunggal dengan seragam (Joseph J.F Cavanagh, 1984).

Teknik *Bit-Pair Recording* adalah hasil terbitan daripada Algorithma Booth. Walaubagaimanapun, teknik ini melaksanakan operasi pendaraban dengan menghasilkan jumlah produk sebanyak dua kali lebih daripada Algorithma Booth.

Walaupun kelajuan operasi pendaraban dapat ditingkatkan, namun, jumlah masa bagi denyutan jam adalah kecil secara amnya. Tetapi, rekabentuk pengawal isyarat akan menjadi lebih kompleks apabila *partial product* dianjakkan beberapa kali dengan tidak konsisten.

Dengan menggunakan teknik ini, masalah bertambah dua kali ganda sekiranya bit 1 dan bit 0 ditukar secara berselang-seli, dan masa yang diambil untuk melengkapkan arahan tersebut adalah lebih lama daripada Algorithma Booth.

2.2.5 Pemilihan Pendarab

Algorithma Booth dipilih untuk meningkatkan kelajuan operasi pendaraban dan algorithma ini dipercayai boleh mencapai beberapa objektif bagi sistem yang telah dinyatakan.

Ia melaksanakan nombor negatifdan nombor positif secara seragam. Dalam teknik ini, sistem boleh mendarab operan lebih pantas dengan menggunakan kaedah mengekod bagi menghasilakan *partial product*. Dengan jujukan bit 0 bagi menggantikan bit 1 yang diselang-selikan dengan bit 0, perlaksanaan operasi pendaraban bagi algorithma ini dimudahkan.

Implementasi pendekatan ini dalam pembinaan perkakasan (hardware) merupakan antara faktor utama pemilihan kaedah ini. Dari segi perkakasan, algorithma ini agak mudah diimplementasi dan sesuai digunakan dalam memenuhi objektif dan tujuan sistem. Komponen yang melaksanakan algorithma Booth ini boleh digunakan lebih dari sekali dan ini mengurangkan kos perkakasan sistem.

2.3 Pendekatan *Pipeline*

Seperti yang telah dinyatakan, *pipeline* diaplikasikan di dalam MAC bagi mengelakkan lengahan yang mana ia diuruskan seperti *assembly line*.

Pendekatan *pipeline* ialah jangka masa kitar masa boleh dikurangkan kepada peringkat *pipeline* paling perlahan bukannya jumlah lengahan. *Pipeline* dilaksanakan pada setiap peringkat bagi menghasilkan satu pengaliran yang berterusan.

Perlaksanaan arahan dalam MAC secara scalar, arahan dilaksanakan pada semua kitar *pipeline* sebelum arahan seterusnya bermula.

Sementara dalam *pipeline*, arahan kedua bermula sebaik sahaja arahan pertama memasuki yang berikutnya. Proses ini dilaksanakan secara berterusan sehingga arahan lengkap diperolehi.

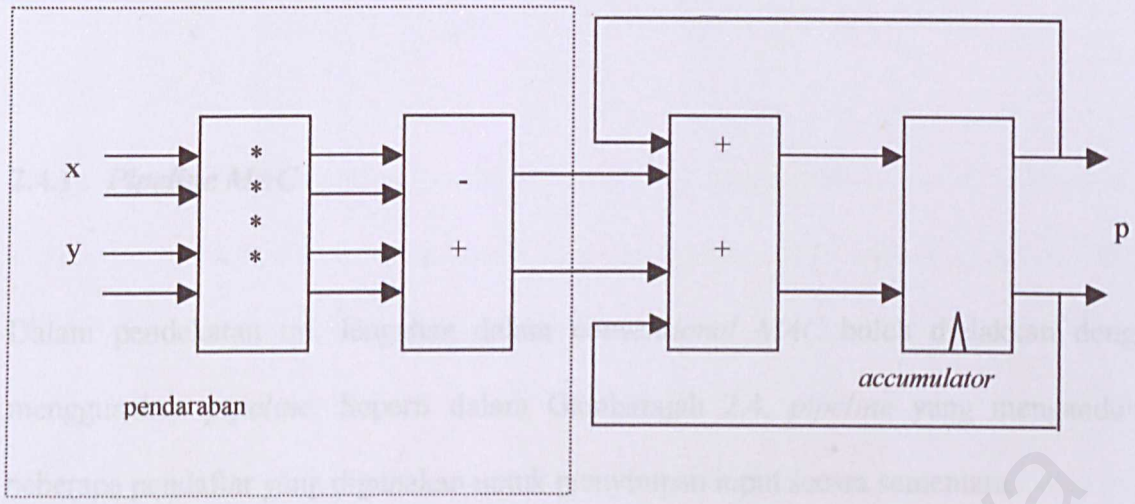
2.4 Perbandingan Antara *Conventional MAC* dengan *Pipeline MAC*

2.4.1 *Conventional MAC*

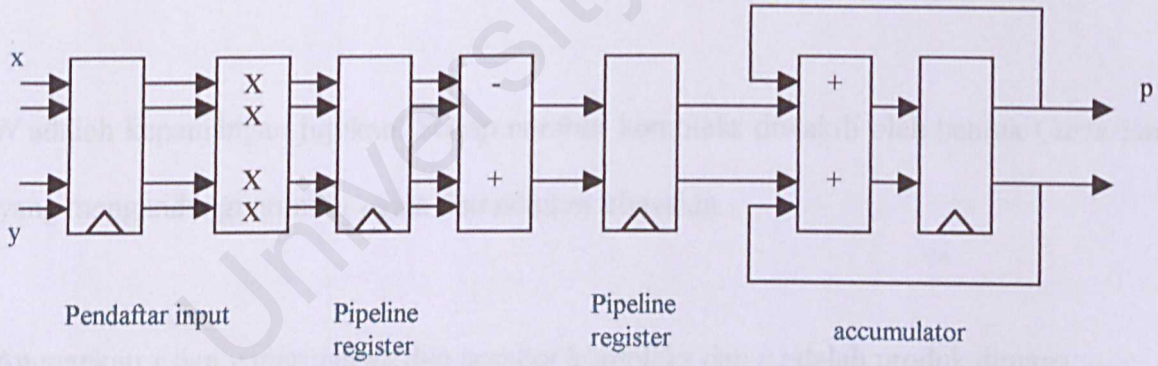
MAC dapatkan keputusan akhir dengan mengambil pasangan nombor-nombor kompleks, daripada setiap satu bagi dua jujukan input, produk kompleks akan dibentuk dan ditambah sebelum produk disimpan dalam pendaftar *accumulator*. *Accumulator* akan dimulakan dengan nilai sifar dan nilai ini akan di *reset* selepas setiap jujukan nombor selesai diproses.

Operasi keseluruhan sistem ini memerlukan empat operasi pendaraban bagi setiap pasangan input yang akan menghasilkan *partial product*, penambahan dan penolakan dan akhir sekali, proses penambahan sebanyak dua kali bagi melaksanakan proses pengumpukkan (*accumulator*) bagi menghasilkan produk penuh.

Gambarajah 2.3 menunjukkan bagaimana MAC melaksanakan tugasnya. Oleh kerana operasi ini perlu dilakukan dalam jujukan, *conventional MAC* memerlukan tiga kitar masa untuk memproses sepasang input sebelum keputusan diumpukkan untuk menjana produk akhir. Dalam menyempurnakan tugas-tugas ini, setiap pasangan input akan menyebabkan lengahan untuk tiga langkah dalam perlaksanaan aplikasi pemprosesan isyarat, keadaan ini akan meminimakan lebar jalur sistem.



Gambarajah 2.3 : Gambarajah Aliran Data yang menunjukkan operasi yang dilaksanakan dalam MAC : Kombinasi Organisasi



Gambarajah 2.4: Gambarajah Aliran Data yang menunjukkan jujukan operasi yang dijalankan oleh MAC : Organisasi Pipeline

2.4.1 *Pipeline MAC*

Dalam pendekatan ini, lengahan dalam *conventional MAC* boleh dielakkan dengan menggunakan *pipeline*. Seperti dalam Gambarajah 2.4, *pipeline* yang mengandungi beberapa pendaftar yang digunakan untuk menyimpan input secara sementara.

MAC yang kompleks beroperasi pada dua jujukan nombor kompleks iaitu, $\{x_i\}$ dan $\{y_i\}$. MAC akan mendarabkan elemen-elemen selari dalam jujukan nombor kompleks dan akan mengumpuk dan mengira jumlah produk bagi menghasilkan:

$$\sum_{i=1}^{N} x_i y_i$$

N adalah kepanjangan jujukan. Setiap nombor kompleks diwakili oleh bentuk *Cartesian* yang mengandungi nombor nyata dan nombor khayalan.

Anggapkan *x* dan *y* merupakan dua nombor kompleks dan *p* adalah produk dimana:

$$p\_real = x\_real \times y\_real - x\_imag \times y\_imag$$

$$p\_imag = x\_real \times x\_imag + y\_imag \times y\_real$$

Jumlah keseluruhan bagi nombor kompleks, *s* dikira seperti dibawah:

$$s\_real = x\_real + y\_real$$

$$s\_imag = x\_imag + y\_imag$$

Pengiraan pertama melibatkan pendaraban dan akan menghasilkan *partial products* atau separa produk sementara *accumulator* atau pengumpuk akan mengumpukkan *partial products* bagi menjana hasil akhir keluaran *Pipelined MAC*.

Sebagai kesimpulan, output akhir bagi *Pipelined MAC* adalah *s_real* dan *s_imag* yang terhasil daripada *partial product*, *p_real* dan *p_imag*.

Dalam *pipeline*, nilai permulaan dan *restart pipeline* haruslah diambil kira. Ini adalah kerana untuk tujuan mengumpukkan jumlah produk bagi bilangan jujukan input.

2.4.1 Conventional MAC vs. Pipeline MAC

Pipeline MAC membawa banyak pendekatan dan perubahan kepada *Conventional MAC*. *Pipeline MAC* dipercayai boleh meningkatkan kelajuan operasi dan ia mengurangkan masa dan kos berbanding *Conventional MAC*. Dengan *pipeline*, pendaftar boleh digunakan untuk menyimpan input secara sementara dan boleh juga untuk mengambil pasangan input baru.

Dengan *pipeline* juga, tiga langkah lengahan telahpun dikurangkan. Pendekatan penambahan *Carry-Look Ahead* dan Algorithma Booth dalam pendaraban digunakan

dalam sistem. Dengan teknik-teknik tersebut, *pipeline MAC* boleh beroperasi dengan pantas daripada sistem terdahulu.

Dalam *pipeline MAC,* isyarat status limpahan digunakan untuk mengawal atau mengurangkan semua kemungkinan bagi limpahan untuk terjadi.

Sebagai kesimpulan, pendekatan baru ini,MAC dengan *pipeline* dan beberapa pendekatan dilaksanakan dalam kaedah operasi arithmetik seperti pendaraban,penambahan dan penolakan meningkatkan kelajuan masa larian bagi unit lama dan mengurangkan bilangan kitar masa dan lengahan masa.

Dalam bab seterusnya, penjelasan terperinci mengenai metodologi yang digunakan, kebaikan dan keburukan metodologi yang dipilih berbanding yang lain. Juga, perbincangan mengenai kelebihan dan keburukan peralatan perkakasan dan perisian (*hardware* dan *software*) yang hendak digunakan dalam mereka bentuk sistem.

3.0 Metodologi

3.1 Pengenalan

Dalam bab ini, semua keperluan dalam membina sistem akan dijelaskan. Dalam sistem ini, keperluan yang paling utama adalah untuk membenarkan MAC dilarikan dengan cepat dan perkakasan yang paling sesuai untuk menyokong sistem ini untuk berfungsi bakal dibincangkan.

Ada dua bahasa pengaturcaraan yang sesuai digunakan bagi merekabentuk, menganalisa, mensimulasi dan mensintesiskan sistem. Antara bahasa pengaturcaraan yang diambil kira untuk digunakan dalam siste ialah VHDL ataupun ery high integrated circuit **H**ardware **D**iscription **L**anguage (VHDL) dan Verilog.

Kedua-dua bahasa pengaturcaraan digital ini digunakan secara meluas. Dalam bahagian seterusnya, kebaikan dan keburukan kedua-dua bahasa ini akan dibincangkan.

Sementara dalam implementasi perkakasan pula, ada dua peranti yang dipilih untuk memenuhi keperluan sistem. Cip yang dipilih ialah *Application Specific Integrated Circuit* (ASIC) dan *Field Programmable Logic Array* (FPGA).

Dalam implementasi perkakasan bagi *Pipelined MAC*, beberapa kajian dilaksanakan keatas kedua-dua perkakasan tersebut. Antara cirri-ciri yang akan diperhatikan ialah senibina kedua-dua perkakasan, simulasi serta pengujian yang akan dilaksanakan semasa merekabentu peranti tersebut.

3.2 Bahasa Permodelan Isyarat Digital

3.2.1 VHDL vs Verilog

Kajian mengenai bahasa ini tebahagi kepada dua bahagian. Pada mulanya, kajian akan meliputi kesamaan antara kedua-dua bahasa dan kemudian, perbezaan antara kedua-dua bahasa tersebut akan di cari.

Kini, terdapat dua piawaian industri bagi bahasa pengaturcaraan isyarat digital iaitu VHDL dan Verilog. Dengan pembinaan cip yang kompleks seperti FPGA dan ASIC, telah meningkatkan bilangan pakar dalam bidang penasihat rekabentuk dengan menggunakan peralatan spesifik dan dengan menggunakan perpustakaan (libraries) tersendiri dan sel mega yang ditulis dalam VHDL dan Verilog. Hasilnya, penting untuk setiap peraka cip mngetahui kedua-due jenis bahasa ini kerana pembekal peralatan, EDA telah menyediakan persekitaran untuk kedua-dua bahasa ini berinteraksi dalam suatu kesatuan. Contohnya, perekabentuk cip mungkin akan menulis model bagi antaramuka bas PCI dalam VHDL dan mahu menggunakan Verilog bagi rekabentuk makronya.

3.2.1.1 Latarbelakang VHDL dan Verilog

VHDL menjadi piawai bagi IEE 1076 pada tahun 1987. Ia dikemaskinikan semula pada tahun 1993 dan kini dikenali sebagai piawai *IEEE 1076 1993*.

Sementara itu, bahasa pengaturcaraan Verilog telah lama digunakan berbanding VHDL. Ianya mula digunakan sejak dilancarkan oleh Gateway pada tahun 1983. Cadence telah membeli *Gateway* pada 1989 dan membuka Verilog kepada domain awam pada 1990. Pada 1995, ia menjadi piawai bagi IEEE 1364.

3.2.1.2 Perbandingan antara VHDL dan Verilog

Perbezaan VHDL dan Verilog meliputi beberapa aspek seperti:

Isu Perbandingan	Verilog	VHDL
Kompilasi	Entiti, *architecture* di simpan dalam satu fail.	Entiti, *architecture* di simpan dalam fail yang berbeza
Jenis Data	Jenis data didefinasikan oleh bahasa	Didefinasikan oleh pengguna
Penggunaan semula	Tidak disimpan dalam *package*	Disimpan dalam *package untuk digunakan semula*
Kekompleksan	Mudah dipelajari	Agak sukar pada peringkat permulaan
Pembinaan Peringkat-Tinggi	Mudah dan tidak menyokong senibina peringkat tinggi	Boleh digunakan dalam senibina peringkat tinggi
Perluasan Bahasa	Tidak boleh dimodelkan dalam bahasa lain	Boleh dimodelkan dengan menggunakan bahasa lain.
Pengurusan Rekabentuk yang Besar	Tidak sokong rekabentuk yang besar	Boleh uruskan struktur rekabentuk yang besar
Operasi	Tidak melaksanakan gelung (*loop*)	Gelung dilaksanakan untuk menjimatkan penggunaan pemboleh ubah yang banyak.
Perpustakaan	Tiada konsep perpustakaan	Perpustakaan tempat simpanan entity, *architevture, package* dan konfigurasi rekabentuk yang telah dikompil.

Jadual 3.0 : Perbezaan antara VHDL dan Verilog

Merujuk kepada Jadual 3.0, menunjukkan bahawa penggunaan VHDL adalah lebih fleksibel dalam fungsi dan penggunaan semulanya serta ia boleh menyokong rekabentuk yang besar. Kelebihan VHDL menyokong keperluan rekabentuk *Pipelined MAC.*

3.3 Implementasi Perkakasan

3.3.1 FPGA vs ASIC

ASIC atau *Application Specific Integrated Circuit* merupakan cip yang direkabentuk oleh jurutera tanpa pengetahuan yang khusus tentang semikonduktur fizikal atau proses-prosesnya. Pembekal ASIC telah menbina perpustakaan sel dan fungsi-fungsi yang boleh digunakan oleh perekabentuk tanpa perlu ketahui bagaimanakan fungsi-fungsi tersebut diimplementasikan ke dalam silikon.

Namun, rekabentuk dalam ASIC kini menjadi semakin rumit dan menelan belanja yang tinggi dan pengguna cip ini tidak boleh mengelakkan masalah pengurusan inventori yang tidak wujud pada FPGA.

FPGA pula dikenali sebagai *Field Programmbale Gate Array* mempunyai struktur yang hampir sama dengan ASIC. Ini menyebabkan PFGA adalah sesuai untuk pemprototaipan bagi ASIC.

Walaupun get-get logic pada FPGA agak perlahan daripada ASIC, namun, perbezaan itu dapat diatasi dengan menggunakan lebih banyak keselarian dalam merakabentuk litar FPGA. (Jonathan Rose, profesor di Universiti Toronto, dan firma kejuruteraan FPGA oleh *Altera Corp*).

Namun, kedua-dua rekebentuk tidak menyokong kepada pengurangan kos unit, meningkatkan perlaksanaan proses, dan bayaran yang berpatutan terhadap *nonrecurring charge*. (Zvi Or-Bach, Presiden bagi pembekal PLD eASIC Corp.)

FPGA adalah lambat, mahal dan memerlukan kuasa yang banyak berbanding ASIC. Namun, FPGA membenarkan pembenamaan cip ASIC kedalam senibinanya.

3.3.2 Pemilihan FPGA

FPGA dipilih kerana bukan sahaja ia mempunyai struktur yang sama dengan PAL(*Programmable Array Logic*), ia juga mempunyai kesamaan dengan get tatasusunan bagi ASIC. ASIC boleh digunakan dalam FPGA untuk mengurangkan kos dan meningkatkan pengeluaran.

FPGA dipilih kerana ianay bersifat *volatile*. Ini bermakna apabila bekalan kuasa ditutup, cip ini tidak boleh melaksanakan tugasnya. Ia juga menggunakan teknologi RAM (*Static*

Random Access Memory). Pendekatan FPGA kepada get tatasusunan tetapi masih boleh dilaksanakan pengaturcaraan.

FPGA menggunakan teknologi yang lebih canggih berbanding ASIC. FPGA juga mengggunakan get yang banyak untuk meningkatkan kelajuan cip. Dengan ini, FPGA mengurangkan kos bagi cip dan secara automatiknya akan mengurangkan kos projek ini.

Dengan pendekatan ini, FPGA akan dipilih dan digunakan semasa implementasi perkakasan.

Bab berikutnya akan menjelaskan tentang proses yang berlaku pada keseluruhan sistem, penyambungan modul pendaraban dengan modul-modul lain (*accumulator* dan *pipeline*) serta kesinambungannya. Perbincangan tentang fungsi dan bagaimanakan modul dapat meningkatkan kelajuan serta mengurangkan lengahan sistem diketengahkan.

BAB EMPAT

REKABENTUK SISTEM

4.0 Rekabentuk Sistem

4.1 Pengenalan

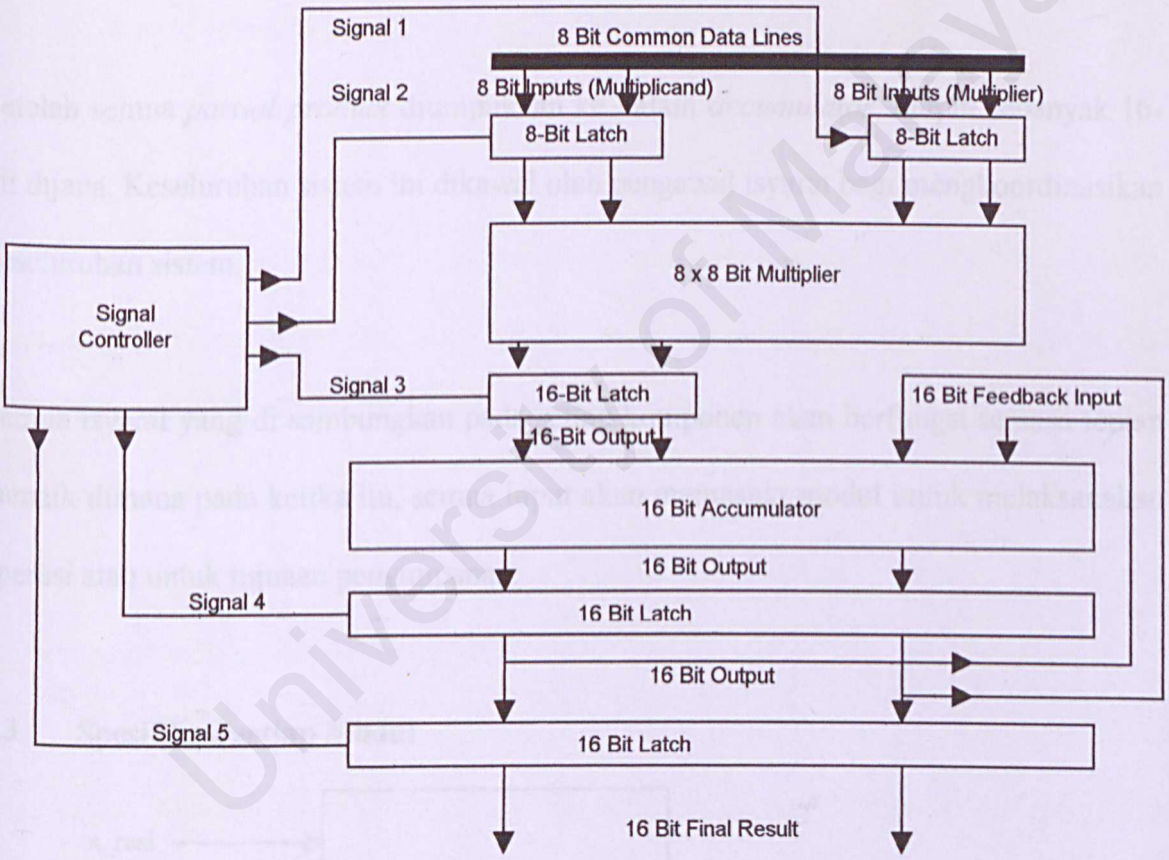
Implementasi dalam *Pipelined MAC* ialah melaksanakan operasi arithmetik keatas 8-bit masukan dalam binari dan akan mengeluarkan output bagi operasi pendaraban sebanyak 16-bit dan bit-bit ini akan menjalankan operasi penambahan dan menghasilkan output sebanyak 16-bit binari.

Faktor utama dalam rekabentuk sistem ini ialah untuk mewujudkan suatu sistem *Pipeline MAC* yang boleh meningkatkan operasi bagi unit pendaraban dan penambahan keatas jujukan data. Untuk memangkin kelajuan operasi ini, maka, *pipeline* diperlukan. Dengan penggunaan *pipeline,* setiap kali masukan memerlukan satu segmen.

Penerangan terperinci mengenai bagaimana keseluruhan sistem *Pipeline MAC* ini beroperasi, dan bagaimana unit-unit yang lebih kecil dan terperinci di integrasikan dalam sistem *pipeline MAC* ini akan dinyatakan.

4.2 Gambaran Keseluruhan Sistem

Secara amnya, sistem digambarkan secara amnya dengan Gambarajah 4.0. Dari Gambarajah tersebut, keseluruhan sistem meliputi modul-modul seperti penambahan, dan pendaraban disambungkan. Sistem menerima input dan menjanakan output dua kali ganda.



Gambarajah 4.0 : Carta Aliran Pipeline MAC

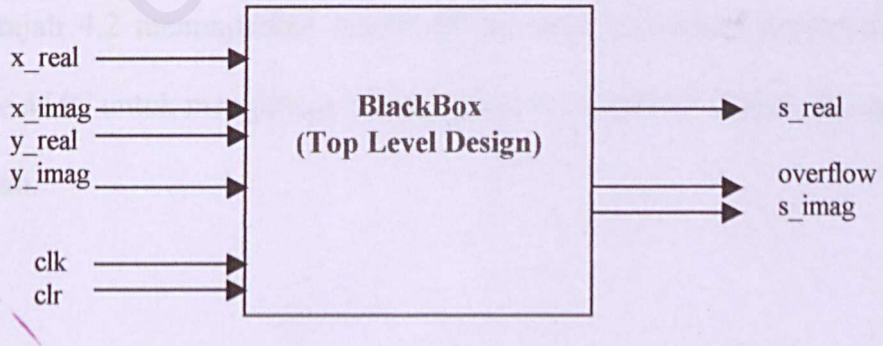
Penggunaan selak (*latches*) adalah untuk merialisasikan teknik *pipeline* disamping itu, tujuan utama penggunaan selak adalah untuk storan data secara sementara. Pada mulanya, sepasang data 8-bit akan dibawa masuk ke dalam sistem oleh dua selak.

Pendarab 8 X 8 yang akan menjana *partial product* sebanyak 16-bit. Pendarab 8 X 8 (*8 X 8 Multiplier*) bermaksud, pendarab tersebut menerima sepasang 8-bit input untuk tujuan pendaraban. Hasil daripada operasi pendaraban, kesemua output pendarab akan dimasukkan kedalam *accumulator*. Pada *accumulator*, semua *partial product* akan ditambah dan nilai atau hasil penambahan (16-bit data) ini akan di suap balik kedalam sistem bagi mendapatkan jumlah hasil penambahan dengan pasangan yang lain.

Setelah semua *partial product* diumpukkan ke dalam *accumulator*, output sebanyak 16-bit dijana. Keseluruhan sistem ini dikawal oleh pengawal isyarat bagi mengkoordinasikan keseluruhan sistem.

Semua isyarat yang di sambungkan pada setiap komponen akan berfungsi semasa tepian menaik dimana pada ketika itu, semua input akan memasuki modul untuk melaksanakan operasi atau untuk tujuaan penyimpanan.

4.3 Spesifikasi setiap Modul



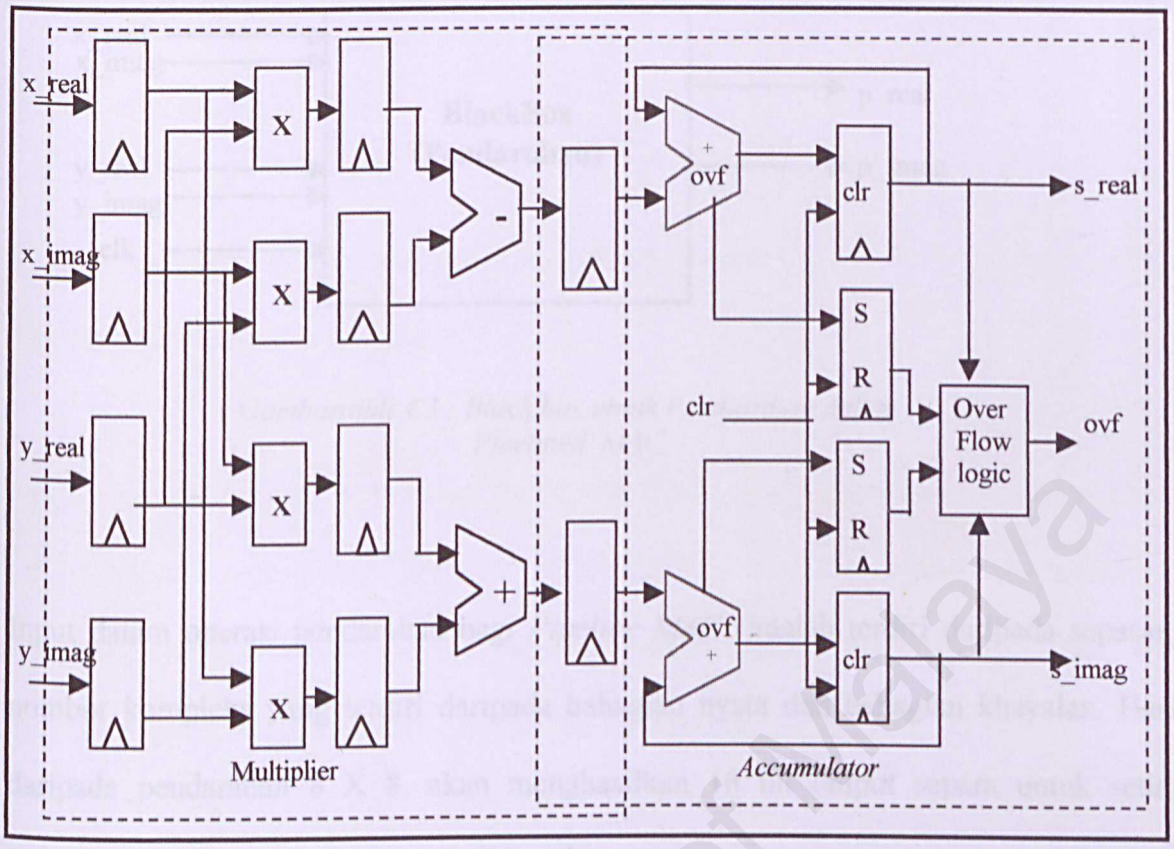
Gambarajah 4.1 :Black Box bagi Pipeline Accumulator Multiplier

Gambarajah 4.1 menunjukkan input dan output bagi keseluruhan sistem. Input yang telibat dalam *Pipeline MAC* ini adalah nombor khayalan dan nombor nyata (merupakan nombor kompleks) sebagai *operand* dan menggunakan *clk* dan *clr* untuk tujuan pengisyaratan digital.

Sistem akan menjana keluar hasil penambahan penuh bagi setiap operasi keatas *partial product* yang dilaksanakan bagi setiap kali masukan diambil. Isyarat limpahan juga dibawa keluar dari sistem.

Dalam rekabentuk ini, kesegerakkan pindahan data bagi data bit antara blok pendarab dengan *accumulator* digunakan, dimana selak digunakan untuk menyelak data masuk dan keluar dari sistem. Kaedah penguncian ini adalah bertujuan untuk menyegerakkan operasi pendaraban dan penambahan. Unit permasaan bagi setiap selak dikawal oleh blok pengawalan isyarat. Pada blok tersebut, denyutan permasaan adalah berbeza dan disalurkan kepada selak-selak yang berkenaan.

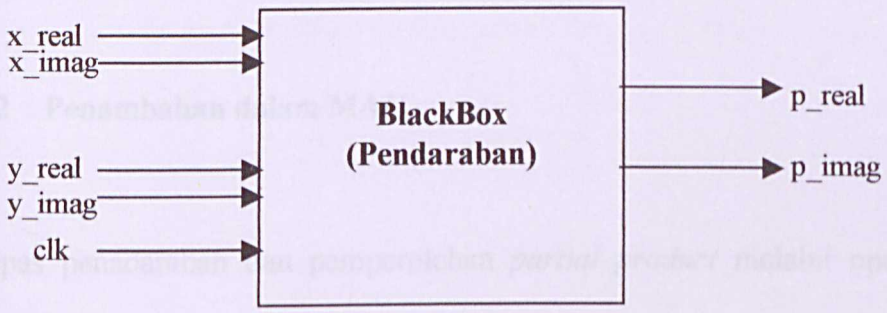
Gambarajah 4.2 menunjukkan kesegerakkan serta koordinasi keseluruhan sistem bagi *Pipeline MAC* untuk mengintegrasikan keseluruh modulnya dengan menggunakan isyarat pemasaan.



Gambarajah 4.2 : Aras-pindahan-pendaftar bagi Pipeline MAC

4.3.1 Pendaraban dalam MAC

Pipeline MAC memerlukan sebanyak empat kali proses penambahan untuk membentuk empat *partial product*. Dengan menggunakan *partial product* ini , barulah *Pipeline MAC* menjana keluaran sebanyak 16-bit bagi nombor nyata dan khayalan.



Gambarajah 4.3 : Black box untuk Pendaraban dalam Pipelined MAC

Input dalam operasi pendaraban bagi *Pipeline MAC* adalah terdiri daripada sepasang nombor kompleks yang terdiri daripada bahagian nyata dan bahagian khayalan. Hasil daripada pendaraban 8 X 8, akan menghasilkan 16 bit output separa untuk setiap bahagian nombor nyata dan nombor khayalan.

Secara spesifiknya, sebelum operasi pengumpukkan dilaksanakan ke atas *partial product* yang terhasil, operasi penambahan dan penolakan dilaksanakan bagi membentuk nombor nyata dan nombor khayalan yang telah didarabkan oleh pendarab 8 X 8 itu .

Dalam *Pipeline MAC*, operasi pendaraban dilakukan pada setiap input dan menghasilkan semua kemungkinan output, rujuk Gambarajah 4.2. Hasil daripada operasi ini adalah penjanaan *partial product* yang akan dilengkapkan semasa operasi pengumpukkan dalam *accumulator* bagi mendapatkan output bagi keseluruhan sistem.

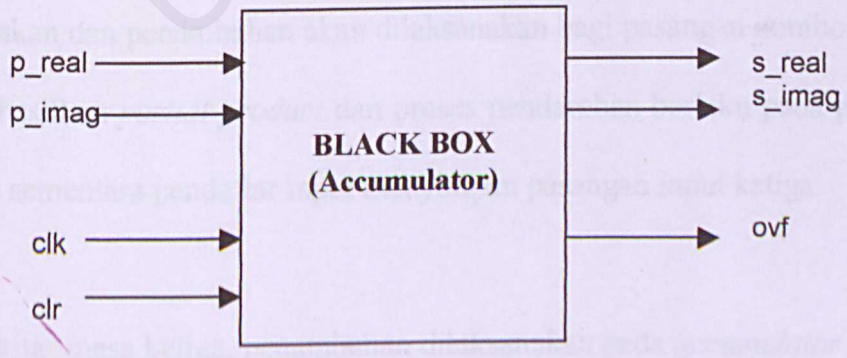
4.3.2 Penambahan dalam MAC

Selepas penadaraban dan pemperolehan *partial product* melalui operasi pendaraban, nombor kompleks akan dijumlahkan.

MAC mengira jumlah dengan mengambil pasangan nombor kompleks (*partial product*), satu daripada setiap jujukan dua input, membentuk produk kompleks dan menambahnya ke dalam daftar *accumulator*. *Accumulator* akan diumpukkan nilai sifar dan diset kembali selepas setiap pasangan jujukan selesai diproses.

Perkara yang perlu diambil kira dalam *pipeline MAC* semasa memulakan dan *restart pipeline* di mana langkah ini perlu untuk mengira semua jumlah bagi produk, mengikut jujukan input.

Dengan memasukkan input *clear* kepada pendaftar *accumulator*, kandungan daftar akan di *reset* kepada sifar pada tepian menaik akan datang.



Gambarajah 4.4 : Black Box bagi Accumulator dalam Pipelined MAC

Merujuk kepada gambarajah diatas, masukan bagi *accumulator* adalah *partial product* yang terhasil daripada operasi pendaraban.

Isyarat *clk* dan *clr* masih lagi digunakan untuk mengekalkan kesegerakan antara komponen ini dengan komponen pendaraban.

Hasil daripada operasi yang dijalankan oleh *accumulator* ini adalah hasil penuh bagi nombor khayalan dan nyata serta isyarat limpahan.

4.3.3 *Pipeline* dalam MAC

Dengan merujuk Gambarajah 4.3, apabila tepian menaik pertama, pasangan nombor kompleks yang pertama disimpan kedalam pendaftar input. Semasa kitar masa pertama, pendarab akan mengira dan menghasilkan *partial products* bagi pasangan nombor yang pertama,sementara itu, sistem akan menyediakan pasangan nombor yang kedua untuk diambil dan disimpan kedalam pendaftar input. Semasa kitar masa yang kedua, penolakan dan penambahan akan dilaksanakan bagi pasangan nombor yang pertama bagi menghasilkan *partial product* dan proses pendaraban berlaku pada pasangan input yang kedua sementara pendaftar input menyimpan pasangan input ketiga.

Pada kitar masa ketiga, penambahan dilaksanakan pada *accumulator* untuk mendapatkan produk penuh dengan menambahkan *partial product* bagi input pasangan pertama dengan

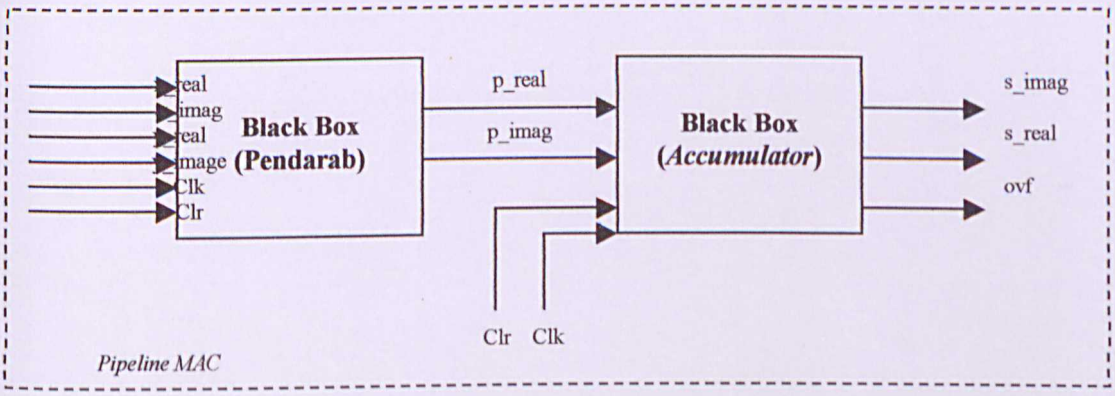
jumlah penambahan sebelumnya dan akan meneruskan tugas yang sama dalam mengira pasangan nombor kedua dan ketiga.

Dalam tepian menaik pada masa keempat, jumlah dalam *accumulator* akan dikemaskinikan (*update*). Tiga kitar masa selepas pasangan input nombor kompleks pertama masuk kedalam pendaftar input, jumlah pasangan ini boleh diperolehi pada output bagi *Pipeline MAC*. Oleh itu, jumlah wujud pada setiap kitar masa. Proses ini boleh mengurangkan dikurangkan kepada peringkat talian paip paling perlahan bukannya jumlah lengahan..

Dengan operasi *pipeline* ini, lengahan dapat dikurangan serta penggunaan sumber dan kuasa dapat dikawal.

4.4 Integrasi Modul-modul dalam *Pipeline*

Modul-modul dalam *Pipeline MAC* boleh diintegrasikan dengan menggunakan isyarat *clr* dan *clk* yang bertujuan untuk mensegerakkan setiap modul.



Gambarajah 4.5 : Integrasi antara modul pendaraban denganaccumulator dalam Pipeline MAC

Dalam Gambarajah 4.5, integrasi antara blok diagram bagi pendarab dan *accumulator* digabungkan. Input bagi seluruh sistem ialah sepasang nombor kompleks yang mempunyai bahagaian khayalan serta nombor nyata.

Isyarat digunakan untuk segerakkan litar. Pendarab akan menghasilkan *partial product* yang akan dikira oleh *accumulator* untuk menghasilkan produk akhir.

Pipeline MAC terdiri daripada beberapa modul atau sub-unit yang melaksanakan fungsi yang berbeza. Untuk mengkoordinasikan komunikasi antara semua sub-unit dalam *Pipeline MAC*, pengawalan isyarat digital (*Digital Signal* Controller) adalah diperlukan.

Dalam bab seterusnya, penerangan tentang penggunaan sintaks serta algoritma dalam mengaturcara modul pendaraban bagi *Pipeline* MAC akan dibincangkan.

BAB LIMA

SISTEM IMPLEMENTASI

5.0 Implementasi Sistem

5.1 Pengenalan

Seperti yang dinyatakan, objektif tesis ini adalah untuk merekabentuk perkakasan bagi *Pipeline Multiplier Accumulator* (*Pipelined* MAC). Dalam melaksanakan pembinaan perkakasan ini, teknologi yang digunakan ialah *Field Programmable Gate Array* (FPGA). Dalam bab ini, penerangan tentang aturcara yang dibuat bagi membina modul pendaraban akan diperjelaskan dengan lebih lanjut.

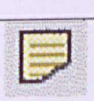
Dalam membina modul pendaraban bagi *Pipeline* MAC, penggunaan bahasa pengaturcaraan yang paling sesuai iaitu VHDL digunakan (sepertimana yang telah dibincangkan dalam bab 3). Bab ini juga akan menyatakan tentang pendekatan yang digunakan dalam aturcara modul tersebut, gaya aturcara dan penjelasan aturcara dengan lebih terperinci.

5.2 Penggunaan *PeakFPGA DESIGNER SUITE FPGA SYNTHESIS EDITION*

PeakVHDL merupakan perisian yang membantu dalam menggunakan VHDL bagi tujuan membina rekabentuk digital bagi sesebuah projek. Perisian ini dapat membantu dalam mensimulasikan bahas pengaturcaraan VHDL dan dapat *link* kan projek dengan bahan pengujian untuk memeriksa output.

PeakVHDL mengandungi integrasi *VHDL simulator*, *VHDL source file editor*, *Hierarchy Browser* dan sumber-sumber lain bagi pengguna VHDL.

Jadual 5.0 menunjukkan tentang kegunaan setiap ikon yang ada pada tetingkap utama PeakFPGA

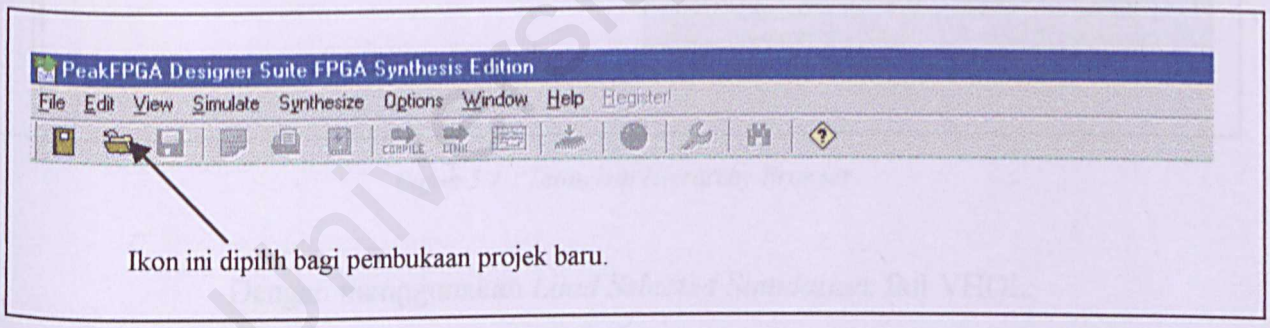
Nama Ikon	Ikon	Penerangan
New Project		Untuk menjana projek baru
Open Existing Project		Untuk membuka projek-projek sedia ada dalam aplikasi
Save Project		Untuk menyimpa perubahan serta untuk simpan projek-projek baru
Create New Module		Untuk menjana atau membina modul baru untuk ditambah kedalam projek
Add Module		Untuk menambah modul baru ataupun *text file*
Compile Selected Module for Simulation		Untuk menkompil modul yang telah di pilih untuk disimulasikan

Link Item for Simulation		Untuk me*link* modul yang telah selesai dikompilkan
Load Selected Simulation		Untuk menguji kesahihan modul
Synthesizes Selected Module		Bagi mensintesiskan modul
Display or Change Program Options		Buka dialog *option* dengan pembukaan *folder compile* adalah aktif
Search Project Files		Untuk mencari fail-fail projek yang berada dalam sistem
Help		Membantu pengguna untuk menggunakan PeakFPGA

Jadual 5.0 : Kegunaan ikon-ikon dalam PeakFPGA

Secara ringkas, berikut adalah penerangan tentang cara-cara penggunaan PeakVHDL:

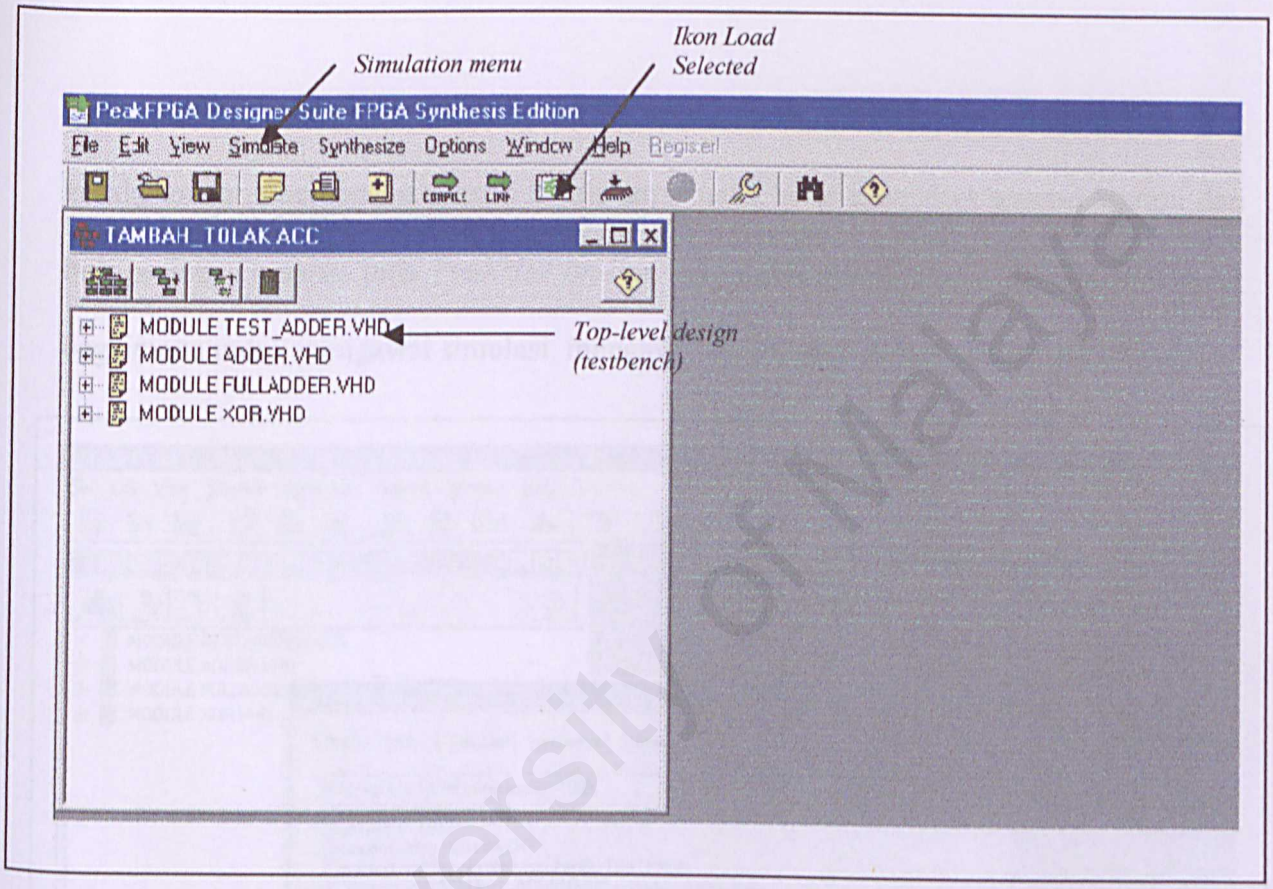
Langkah 1 : sebagai permulaan, buka satu projek contoh daripada subdirektori PeakVHDL.



Rajah 5.0 :Tetingkap utama dalam PeakFPGA Designer Suite FPGA Synthesis Edition

Langkah 2 : untuk melaksanakan simulasi bagi PeakVHDL, pilih fail sumber teratas *(testbench)* dengan mengkliknya dan kemuadian pilih menu *Load Selected*

dari *Simulate menu* atau klik pada ikon *Load Selected* dari *toolbar* PeakVHDL .



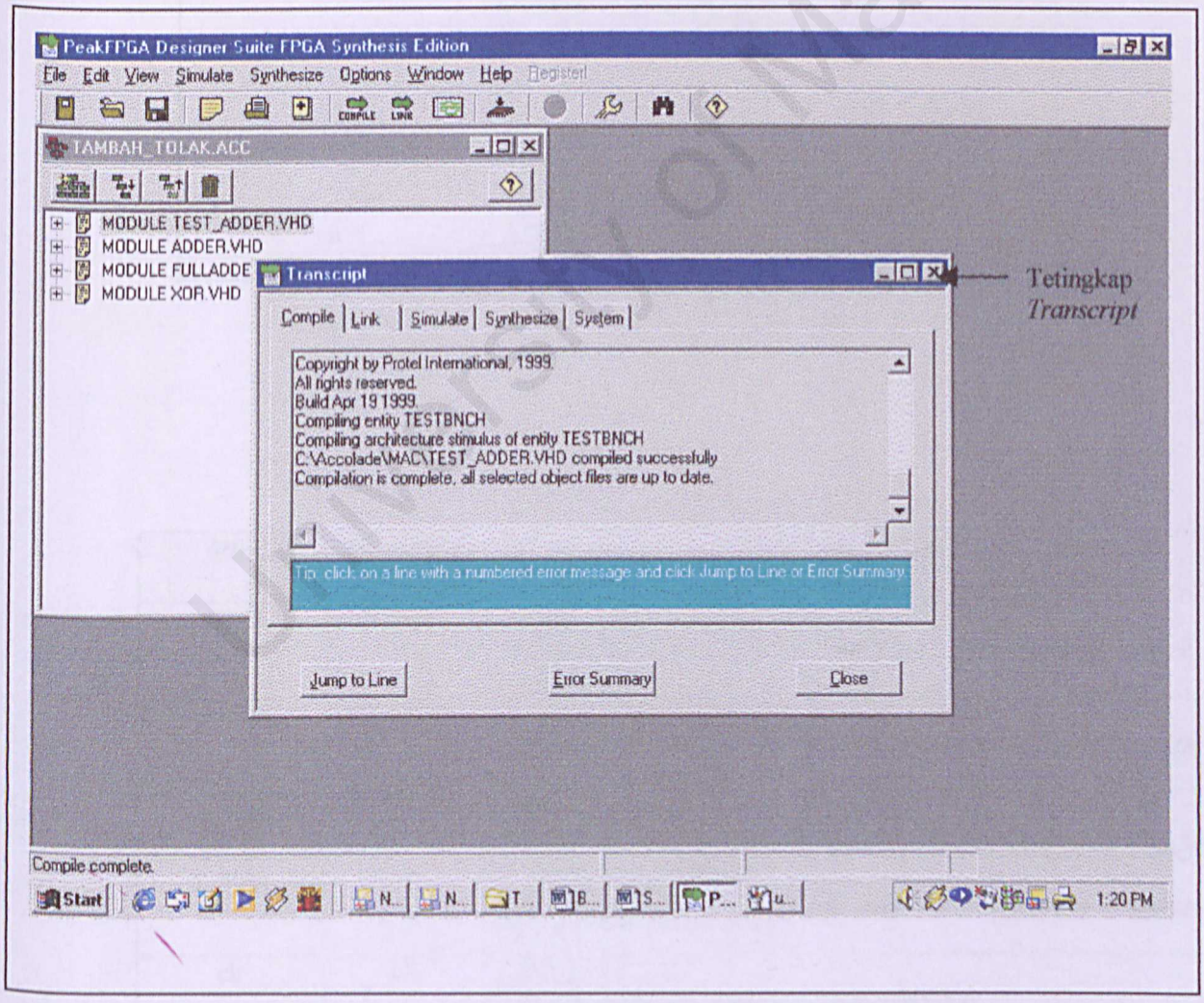
Rajah 5.1 : Tetingkap Hierarchy Browser

Dengan menggunakan *Load Selected Simulation,* fail VHDL:

- Kompilkan modul menggunakan turutan *bottom-up* sepertimana dalam *Hierarchy Browser*
- Kod yang telah dikompil akan di *link* kan dan menjana simulasi .VX

- Akhir sekali, simulasi .VX akan di masukkan kedalam simulasi aplikasi PeakFPGA

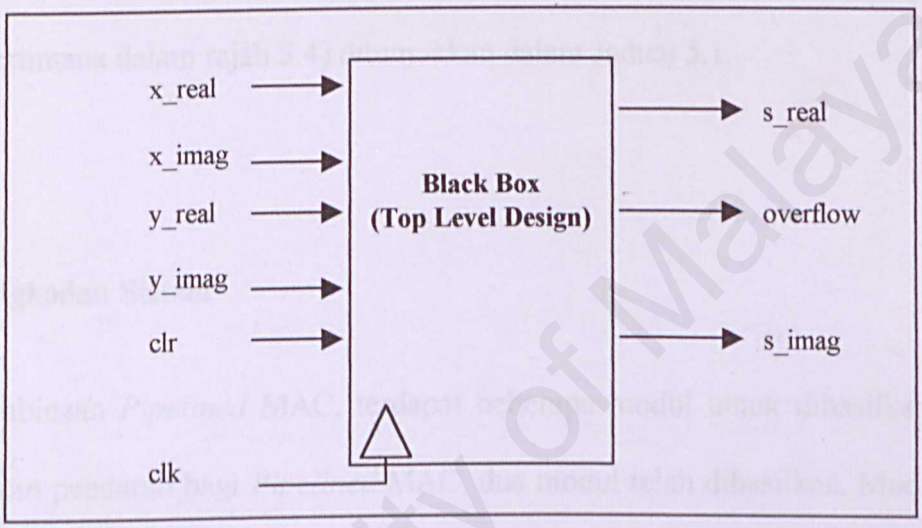
Langkah 3 : tetingkap transkrip VHDL akan menguji dan memberitahu sebarang kesalahan yang berkenaan dengan kesalahan sintaks, serta kesalahan semasa kompil dan link projek. Terdapat juga *PeakSIM on-line* help untuk memberikan maklumat tentang bagaimana untuk mengawal simulasi, mengawal isyarat dan *debug* rekabentuk projek.



Rajah 5.3 : Tetingkap Transkrip

5.3 Penerangan Pin Bagi *Pipeline MAC*

Penerangan mengenai pin dan pendaftar bagi kod bagi system digital dalam VHDL adalah penting. Dalam bahagian ini, penjelasan dan penerangan tentang spesifikasi rekabentuk bagi *Pipelined MAC* akan dibincangkan. Jadi, fungsi bagi setiap pin dan pendaftar akan diterangan.



Rajah 5.4 : Simbol Rekabentuk Top-Level bagi Pipelined MAC

Pin	Masukan/ Keluaran	Penerangan
x_real	IN	**X Real Number** Menggunakan masukkan 8 bit input nombor-nombor nyata yang pertama
x_imag	IN	**X Imaginary Number** Menggunakan masukkan 8 bit input nombor-nombor khayalan yang pertama
y_real	IN	**Y Real Number** Menggunakan masukkan 8 bit input nombor-nombor nyata yang kedua
y_imag	IN	**Y Imaginary Number** Menggunakan masukkan 8 bit input nombor-nombor khayalan yang kedua
clr	IN	**Clear** Untuk mereset input yang ada pada pendaftar
clk	IN	**Clock** Memasukkan input hanya pada tepian menaik

s_real	OUT	**Sum Real** Menghasilkan produk nombor nyata yang mempunyai 16 bit keluaran
s_imag	OUT	**Sum Imaginary** Menghasilkan produk nombor nyata yang mempunyai 16 bit keluaran
Overflow	OUT	**Overflow Control** Mengeluarkan nilai limpahan yang dihasilkan dalam sistem

Jadual 5.1 : Penerangan Pin Dalam Pipelined MAC

Penerangan mengenai pin-pin yang terlibat dalam pembinaan aras atas bagi *Pipelined MAC* (sepertimana dalam rajah 5.4) ditunjukkan dalam Jadual 5.1.

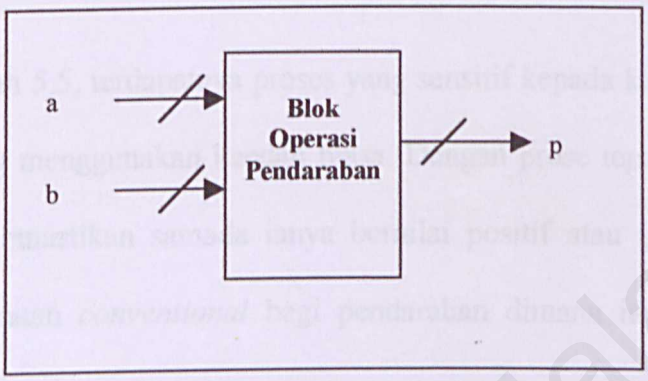
5.4 Pengkodan Sistem

Dalam pembinaan *Pipelined* MAC, terdapat beberapa modul untuk dihasilkan. Dalam menghasilkan pendarab bagi *Pipelined* MAC, dua modul telah dihasilkan. Modul-modul yang terhasil ialah pendarab biasa dan pendarab dengan menggunakan pendekatan algorithma Booth yang bertujuan bagi memendekkan masa pendaraban biasa.

Dalam bahagian ini juga akan membincangkan secara lebih terperinci tentang pembinaan senibina bagi operasi tambah dan tolak yang digunakan bagi mendapatkan produk lengkap yang terdiri daripada nombor nyata dan nombor khayalan sahaja. Untuk menghasilkan operasi penolakan, bit-bit binari harus ditukarkan terlebih dahulu kepada pelengkap duaan.

Bagi pengkodan lengkap untuk kedua-dua modul, rujuk lampiran 4 Pengkodan sistem.

5.4.1 Modul Pendaraban



Rajah 5.5 : Blok Entiti bagi Modul Pendaraban dalam Pipelined MAC

Rajah 5.5 merupakan rajah bagi input dan output yang terlibat dalam membina modul pendaraban bagi *Pipelined* MAC.

Input yang terlibat ialah a dan b yang merupakan dua masukan 8 bit sementara bagi output modul tersebut ialah p yang merupakan hasil darab dua input (a dan b) yang bernilai 16-bit keluaran.

5.4.1.1 Operasi Pendaraban Mengikut Kaedah Biasa

Dengan merujuk Rajah 5.5, terdapatnya proses yang sensitif kepada kedua input a dan b bagi pendaraban yang menggunakan kaedah biasa. Dengan prose tersebut, input bagi a dan b diuji untuk memastikan samada ianya bernilai positif atau negatif. Modul ini menggunakan pendekatan *conventional* bagi pendaraban dimana ianya menggunakan cara pendaraban yang panjang.

Bagi menguji *multiplicand* adalah nombor postif atau negatif, gelung seperti rajah 5.6 digunakan. Tujuan gelung itu adalah untuk memastikan *multiplicand* yang terlibat adalah bernilai negatif dan melaksanakan operasi pelengkap duaan keatasnya bagi meneruskan proses penambahan. Ini kerana, nombor binari tidak boleh melaksanakan operasi penolakan secara terus. Ianya haruslah ditukarkan kepada pelengkap duaan dan kemudiannya akan melaksanakan operasi penambahan bagi menghasilkan produk akhir.

Dalam gelung tersebut, nilai *carry_in* adalah 1 dimana ianya akan ditambah pada input a atau b yang telah dilaksanakan operasi pelengkap satuan keatasnya.

```
    negative_result := (op1(7) = '1') xor (op2(7) = '1');      -- untuk digunakan kemudian
dalam menentukan bit tanda

    if (op1(7) = '1') then        -- pengujian bagi memastikan nombor bagi multiplicand
tersebut adalah negatif atau tidak
            carry := '1';         -- anggap carry sebagai 1
            for index in 0 to 7 loop
                carry_in := carry;
                carry := carry_in and op1(index);
                op1(index) := not op1(index) xor carry_in;
            end loop;             -- gelung bertujuan untuk melaksanakan pelengkap duaan
bagi menukarkan nombor negatif bagi input a
        end if;
        if (op2(7) = '1') then
        carry := '1';
            for index in 0 to 7 loop
                carry_in := carry;
                carry := carry_in and op2(index);
                op2(index) := not op2(index) xor carry_in;
            end loop;         -- gelung bertujuan untuk melaksanakan pelengkap duaan bagi
menukarkan nombor negatif bagi input b
        end if;
```

Rajah 5.6 : Gelung yang akan Menukarkan Nombor-nombor Negatif kepada Pelengkap Duaan Sebelum Melaksanakan Operasi Penambahan.

```
-- melaksanakan operasi pendaraban yang panjang

        result := (others => '0');   -- sekiranya nilai yang lain adalah sifar; nombor
adalah positif dan tidak perlu melaksanakan apa-apa perubahan keatas nombor tersebut

        for count in 0 to 7 loop
            carry := '0';
            if (op2(count) = '1') then
                for index in 0 to 7 loop
                carry_in := carry;
                carry := (result(index+count) and op1(index))
                          or (carry_in and (result(index+count) xor op1(index)));
                result(index+count) := result(index+count) xor op1(index) xor
carry_in;
            end loop;    -- gelung melaksanakan operasi penambahan bagi melaksanakan
operasi pendaraban.
                         -- menambahkan setiap kali masukan bit untuk menghasilkan
partial product
                result(count+8) := carry;
            end if;
        end loop;
```

```
        if negative_result then
            carry := '1';
            for index in 0 to 15 loop
                carry_in := carry;
                carry := carry_in and not result(index);
                result(index) := not result(index) xor carry_in;
            end loop;    -- gelung untuk melaksanakan operasi penukaran semula nombor
negatif kepada bentuk yang asal.

        end if;
        p <= result after Tpd_in_out;

        -- product diperolehi selepas 40ns.
```

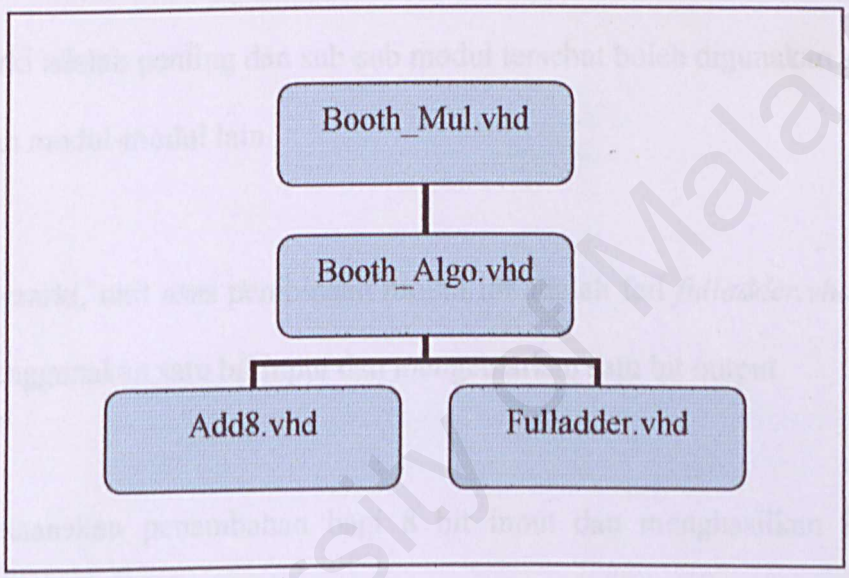
Rajah 5.7 : Gelung untuk Melaksanakan Pendaraban Panjang serta Penukaran Semula NilaiPelengkap duaan.

Sekiranya input a atau b bernilai positif, input akan dilaksanakan penambahan tanpa sebarang perubahan yang dilakukan kepada input tersebut. Setiap kali proses pendaraban akan menghasilkan produk separa. Proses pendaraban menggunakan gabungan operasi penambahan serta penambahan penuh bagi melaksanakan proses tersebut dengan jayanya.

Gelung kedua merupakan penukaran semula nombor pelengkap duaan kepada nombor asal. Keluaran dijana selepas 40 ns.

5.4.1.2 Operasi Pendaraban Menggunakan Pendekatan Algorithma Booth

Untuk melaksanakan pendekatan ini, beberapa sub-sub modul perlu diintegrasikan. Rajah 5.8 menunjukkan tentang hierarki pembinaan modul ini.



Rajah 5.8: Pepohon Hierarki bagi perlaksanaan Algorihma Booth dalam Pendaraban

Dalam melaksanakan pendekatan algorithma Booth dalam modul pendaraban, beberapa sub module terlibat iaitu modul *Booth_Mul*, *Booth_Algo*, *Add8* dan *Fulladder*. Penjelasan bagi setiap modul secara ringkas boleh didapati dengan merujuk Jadual 5.0 .

Fail VHDL	Penerangan
Booth_Mul.vhd	- Terletak pada hierarki teratas. - Melaksanakan penggunaan modul Booth_Algo - Melaksanakan operasi pendaraban yang melibatkan algorithma Booth
Booth_Algo.vhd	- Aturcara yang akan melaksanakan pendekatan algorithma Booth keatas operasi pendaraban - Mengintegrasikan penggunaan sub modul *fulladder* dan add8

Add8.vhd	- Merupakan operasi pendaraban secara bersiri. - Akan menambah 8 bit input menggunakan modul *fulladder* menggunakan isyarat.
Fulladder.vhd	- melaksanakan operasi penambahan bagi menghasilkan produk lengkap - melaksakan operasi bagi 1 bit input dan menghasilkan 1 bit output.

Jadual 5.2 : Penerangan Ringkas Tentang Kegunaan Fail Modul Pendaraban dengan Pendekatan Algorithma Booth

Dalam modul ini, banyak menggunakan sub modul bagi membina sub modul yang lebih besar. Hierarki adalah penting dan sub-sub modul tersebut boleh digunakan semula untuk menghasilkan modul-modul lain.

Mengikut hierarki, unit asas pembinaan modul ini adalah fail *fulladder.vhd*. Sub modul ini hanya menggunakan satu bit input dan mengeluarkan satu bit output.

Untuk melaksanakan penambahan bagi 8 bit input dan menghasilkan 8 bit output, penambah bersiri diperlukan. Disinilah pentingnya penjanaan fail *Add8.vhd*. Dalam *Add8.vhd*, penggunaan sub modul *fulladder.vhd* yang sama untuk mendapatkan 8 bit output bagi 8 bit input.

Fail VHDL *Booth_Algo.vhd* pula mengenai implementasi algoritma Booth dalam melaksanakan operasi pendaraban. Dalam fail ini, pendarab di anjak bagi mempercepatkan masa pendaraban dan *multiplicand* di ubahsuai kepada bit-bit yang mempunyai banyak jujukan sifar bagi memudahkan serta meningkatkan masa

pendaraban. Aturcara untuk menganjak serta mengimplementasikan algorithma Booth ini boleh dilihat pada rajah 5.9.

Akhir sekali, fail *Booth_Mul.vhd* pula melaksanakan operasi pendaraban berasaskan penggunaan algorithma Booth dalam *Booth_Algo.vhd*. Ini bagi memastikan proses pendaraban memakan masa yang singkat dan dapat mengabaikan jujukan bit-bit 1 dan 0 yang berselangan.

```
two_b <= b(6 downto 0) & '0';-- anjakan 1 tempat
b_bar <= not b;       -- 1's complements for multiplicand
bb <= b when a= "01"                -- umpukkan nilai b kepada bb apabila a adalah bernilai
01(penambahan)
        else b_bar when a="10"      -- umpukkan nilai b_bar kepada bb apabila a adalah
bernilai 10(penolakan)
        else x"00";                 -- umpukkan nilai x'00' kepada bb apabila a adalah
bernilai 11 atau 00 (tiada perubahan)

cin <= '1' when a="10"              -- untuk operasi tolak; perlukan penambahan bit 1;
umpukkan nilai 1 pada cin bila operasi penolakan
           else '0';                -- tiada perubahan untuk nilai cin bila a adalah
01,00,11

topbit <= b(7) when a = "01"                    -- bit teratas b diumpukkan kepada topbit
bila a = 01
                  else b_bar(7) when a = "10" -- bit teratas b_bar diumpukkan kepada
topbit bila a = 10
                  else '0';                     -- bit teratas 0 diumpukkan kepada
topbit bila a = 11 atau 00

-- untuk laksanakan operasi penambahan; gunakan add8 dan fulladder
a1 : entity work.add8 port map(sum_in,bb,cin,psum,cout);                          --
petakan add8 kepada booth_algo
a2 : entity work.fulladder port map(sum_in(7), topbit, cout, topout, nc1);     -- petakan
fulladder kepada booth_algo


sum_out (5 downto 0) <= psum(7 downto 2);
sum_out (7) <= topout;
sum_out (6) <= topout;
prod <= psum ( 1 downto 0);
```

Rajah 5.9 : Badan Aturcara bagi Pendaraban yang Menggunakan Pendekatan Algorithma Booth

5.4.2 Modul Penambahan dan Penolakan

Modul ini melibatkan tiga modul iaitu modul *XOR*, *Fulladder* dan *Adder*. Sama seperti modul pendaraban menggunakan pendekatan algorithma Booth, modul ini juga terhasil daripada hierarki.

Modul ini mempunyai input a dan b yang masing-masing mempunyai 16 bit masukan. Disamping itu, untuk menentukan operasi yang patut dilaksnakan, penggunaan *mode* yang bernilai 1 bit diperlukan. Apabila nilai *mode* di setkan kepada satu, maka, operasi penolakan akan dilaksanakan dan sekiranya nilai *mode* adalah sifar, maka operasi penambahan akan dijalankan.

```
 component FullAdder      -- menggunakan komponen modul fulladder untuk operasi penambahan
binari
port (  x: in std_logic;
        y: in std_logic;
        cin: in std_logic;
        cout: out std_logic;
        sum: out std_logic
        );
end component;
-- memetakan port masukan/keluaran bagi modul Adder dengan port masukan/keluaran komponen
fulladder

component hasil_XOR      -- menggunakan komponen modul hasil_XOR untuk operasi penolakan
binari
port (
        X: in std_logic;
        Sub: in std_logic;
        XOR_out: out std_logic
    );
end component;
-- memetakan port masukan/keluaran bagi modul Adder dengan port masukan/keluaran komponen
hasil_XOR


signal C : std_logic_vector (15 downto 0);     -- menggunakan isyarat yang akan mewakili
setiap bit.
signal D : std_logic_vector (15 downto 0);

-- signal C mewakili bawaan masuk dan signal D mewakili sub(input yang bernilai negatif
atau positif)
```

```
begin

F0      : hasil_xor port map ( b(0), mode, D(0));                    -- petakan masukan
adder kepada fulladder dan modul hasil_xor
F0a     : fulladder port map ( a(0), D(0), mode, C(1),sum(0));
F15     : hasil_xor port map ( b(15), mode, D(15));
F15a    : fulladder port map ( a(15), D(15), C(15), cout,sum(15));

-- penggunaan komponen hasil_xor dengan fulladder sebanyak 16 kali untuk menghasilkan
output
-- perlaksaan berdasarkan bit demi bit
```

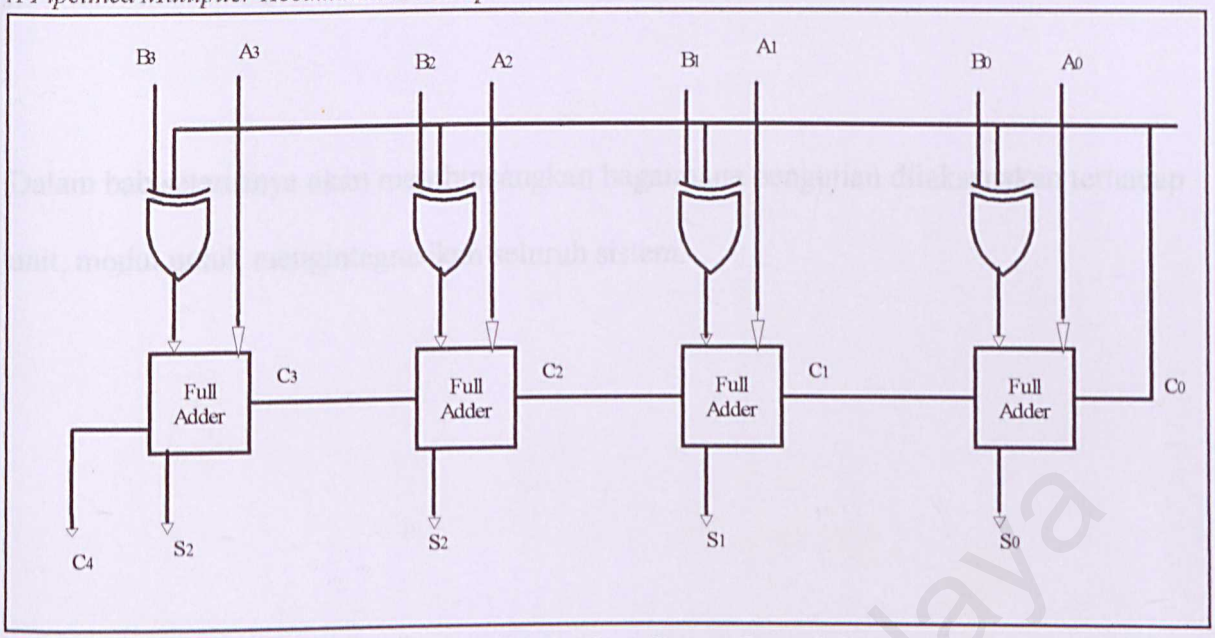
Rajah 5.10 : Senibina untuk Melaksanakan Modul Penambahan dan Penolakan bagi Pipelined MAC

Untuk melaksanakan modul ini, *Adder* bertujuan untuk mengintegrasikan sub-modul *fulladder* dan *XOR*. Tujuan sub-modul *fulladder* ialah untuk mendapatkan produk bagi penambahan penuh. Sementara sub-modul *XOR* adalah untuk melaksakan operasi pelengkap duaan bagi operasi penolakan(Rujuk Rajah 5.11)

```
-- untuk melaksanakan pelengkap duaan bagi nombor-nombor khayalan sebelum
-- operasi tambah.
-- apabila:
-- a) sub == 0; B XOR 1 = NOT B; Cout adalah 1
-- b) sub == 1; B XOR 0 = B
-- jadi,sub yang akan mengawal operasi yang bakal dilaksanakan
--
    XOR_out <= X XOR Sub;                    -- melaksanakan operasi logik xor keatas operan
dimana
                                             -- operasi xor == operasi pelengkap satuan(1st
complement)

                                          -- yang dilaksanakan
```

Rajah 5.11 : Operasi dan Penerangan Sub-modul XOR bagi Tujuan Penolakan

Modul ini dibina berdasarkan pemerhatian keatas get-get logik bagi operasi penambahan dan penolakan. Rajah 5.12 merupakan rajah skematik bagi operasi tersebut.



Rajah 5.12 : Gambarajah skematik(4 bit) bagi Operasi Penambahan dan Penolakan.

Secara kesimpulannya, untuk membina modul-modul *Pipelined* MAC, berbagai peringkat perlaksanaan perlu dilakukan dengan mengikut tertib yang betul seperti contoh simulasi -> *link* -> paparan *waveform.*

Bagi integrasi antara sub-modul, hierarki yang betul adalah penting.

Dalam pembinaan unit pendaraban menggunakan pendekatan algorithma Booth memerlukan sub-unit *fulladder*, *Add8* dan *Booth_algo* yang akan diintegrasikan oleh *Booth_mul.*

Pembinaan modul penambahan dan penolakan pula, sub-unit *fulladder* dan *XOR* akan diintegrasikan oleh unit teratas modul itu iaitu *Adder.*

Dalam bab seterusnya akan membincangkan bagaimana pengujian dilaksanakan terhadap unit, modul untuk mengintegrasikan seluruh sistem.

BAB ENAM

PENGUJIAN SISTEM

6.0 Pengujian Sistem

6.1 Pengenalan

Dalam bab ini perbincangan mengenai bagaimana untuk menguji model VHDL bagi *Pipelined* MAC akan diketengahkan. Untuk melaksanakan pengujian, penggunaa *testbench* adalah perlu. *Testbench* merupakan model struktur dengan melibatkan dua komponen iaitu *tester* dan *model under test*. *Model under test*s adalah model struktur atau *behavioral* model VHDL bagi sistem digital.

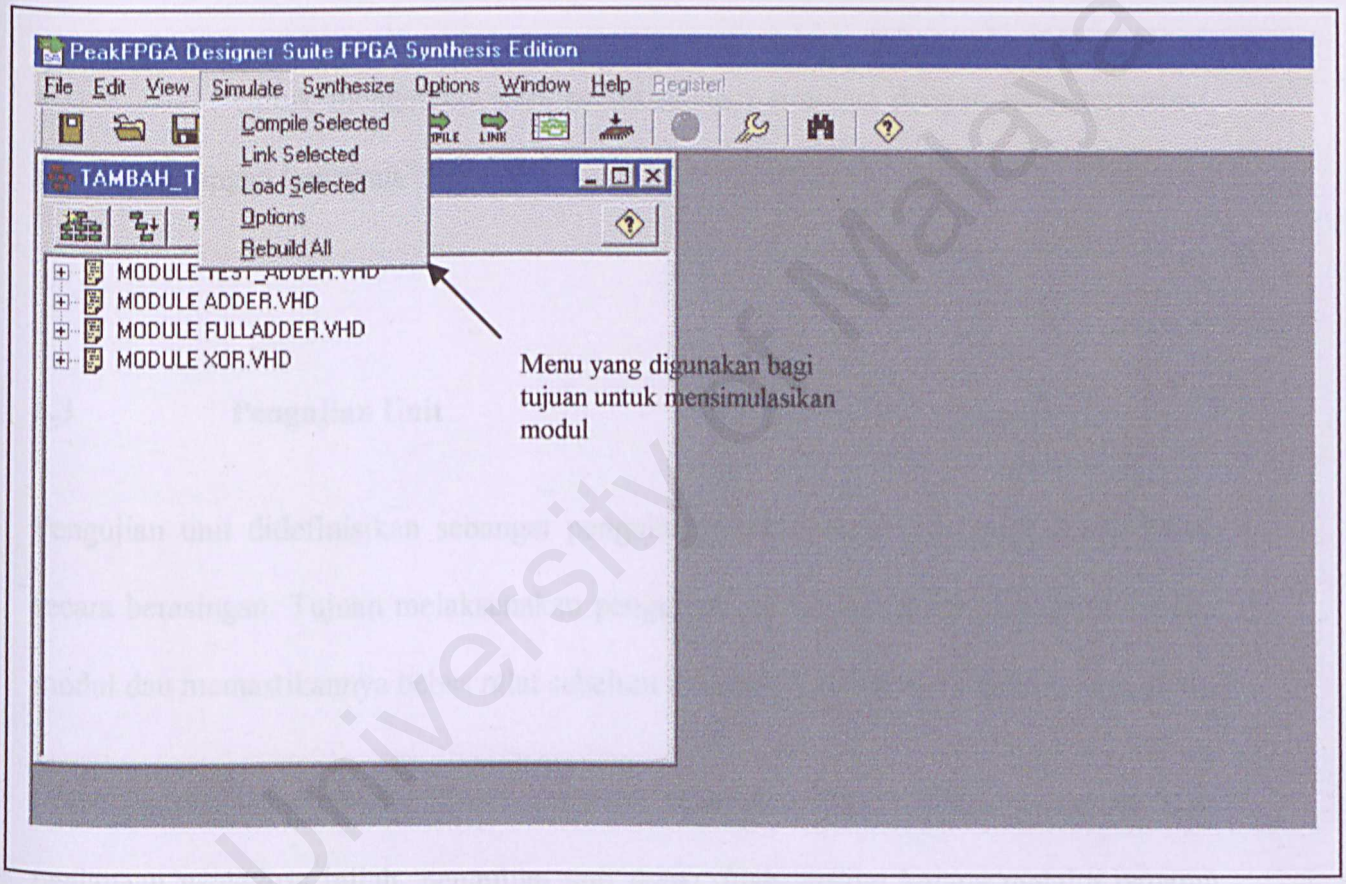
Dalam system *Pipelined* MAC, setiap modul mempunyai *testbench* yang tersendiri bagi menguji input dan output bagi setiap modul.

Pengujian terbahagi kepada dua bahagian iaitu pengujian unit dan pengujian integrasi sistem yang menggunakan cara pengujian yang sama iaitu *testbench.* Pengujian unit dilaksanakan sebelum pengujian integrasi sistem dilakukan.

Penerangan yang lebih lanjut mengenai pengujian unit dan pengujian integrasi sistem akan didapati dalam bahagian-bahagian yang seterusnya dalam bab ini.

6.2 Simulasi Menggunakan PeakFPGA

Bagi membuat pengujian, *PeakFPGA* boleh digunakan. Ada empat pilihan untuk mensimulasikan *testbench* iaitu c*ompiled selected, link selected, load selected* dan *options.*



Rajah 6.0 : Tetingkap yang menunjukkan menu pilihan bagi simulasi

6.2.1 Menu Pilihan bagi Simulasi

Bagi menu pilihan yang digunakan simulasi, ianya perlu untuk menjana serta menguji hasil kod bagi aturcara VHDL. Ikon-ikon serta penerangan mengenainya telah dijelaskan dalam bahagian 5.2 mengenai penggunaan *PeakFPGA* secara terperinci.

- Untuk mengetahui cara melaksanakan simulasi dengan lebih terperinci, sila rujuk ruangan lampiran.

6.3 Pengujian Unit

Pengujian unit didefinisikan sebangai pengujian modul-modul dalam *Pipelined* MAC secara berasingan. Tujuan melaksanakan pengujian unit adalah untuk memeriksa setiap modul dan memastikannya bebas ralat sebelum dihimpunkan untuk menjadi sistem akhir.

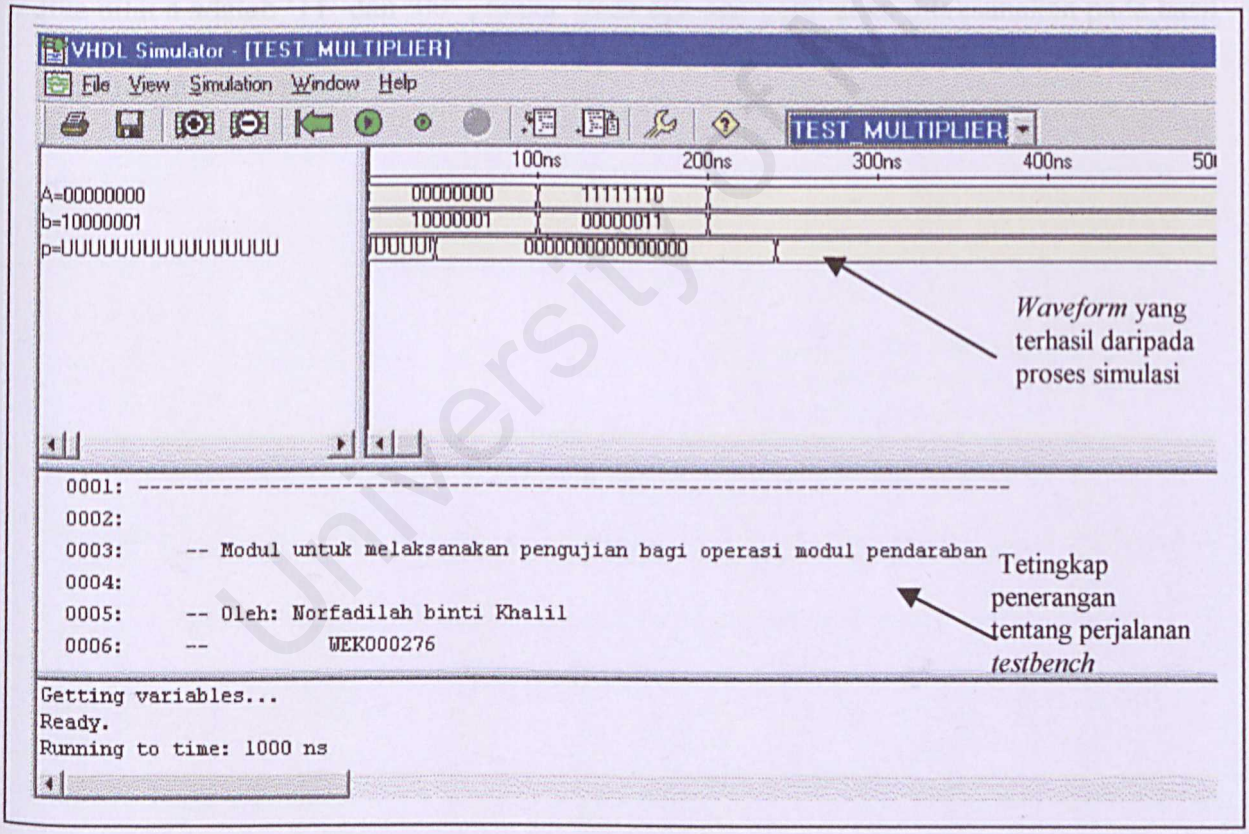
Untuk melaksanakan pengujian unit, *testbench* diperlukan untuk menjana *waveform*. Dari penjanaan *waveform* inilah, pengujian unit dapat dilaksanakan kerana melalui paparan *waveform* tersebut, dapat diperiksa setiap input masukan dan perbandingan output yang didapati dengan output yang terhasil.

Dalam sub-bahagian yang seterusnya, pengujian unit bagi operasi pendaraban serta operasi penambahan dan penolakan dapat diperjelaskan.

6.3.1 Operasi Pendaraban

6.3.1.1 Pendaraban Biasa Menggunakan Fail *Multiplier.vhd*

Bagi proses pendaraban menggunakan fail *multiplier.vhd,* seperti yang telah dinyatakan, proses adalah sensitive terhadap sebarang perubahan input a dan b. Sekiranya terdapat perubahan dalam input a dan b, maka akan mengubah output yang bakal dijana. Dari output yang terhasil, dapatlah ditentukan kesahihan teori yang digunakan.



Rajah 6.1 : Waveform daripada testbench bagi modul pendaraban Pipelined MAC

Bagi input yang bernilai negatif kepada kedua-dua masukan, hasil pendarabannya ialah dalam pelengkap duaan.

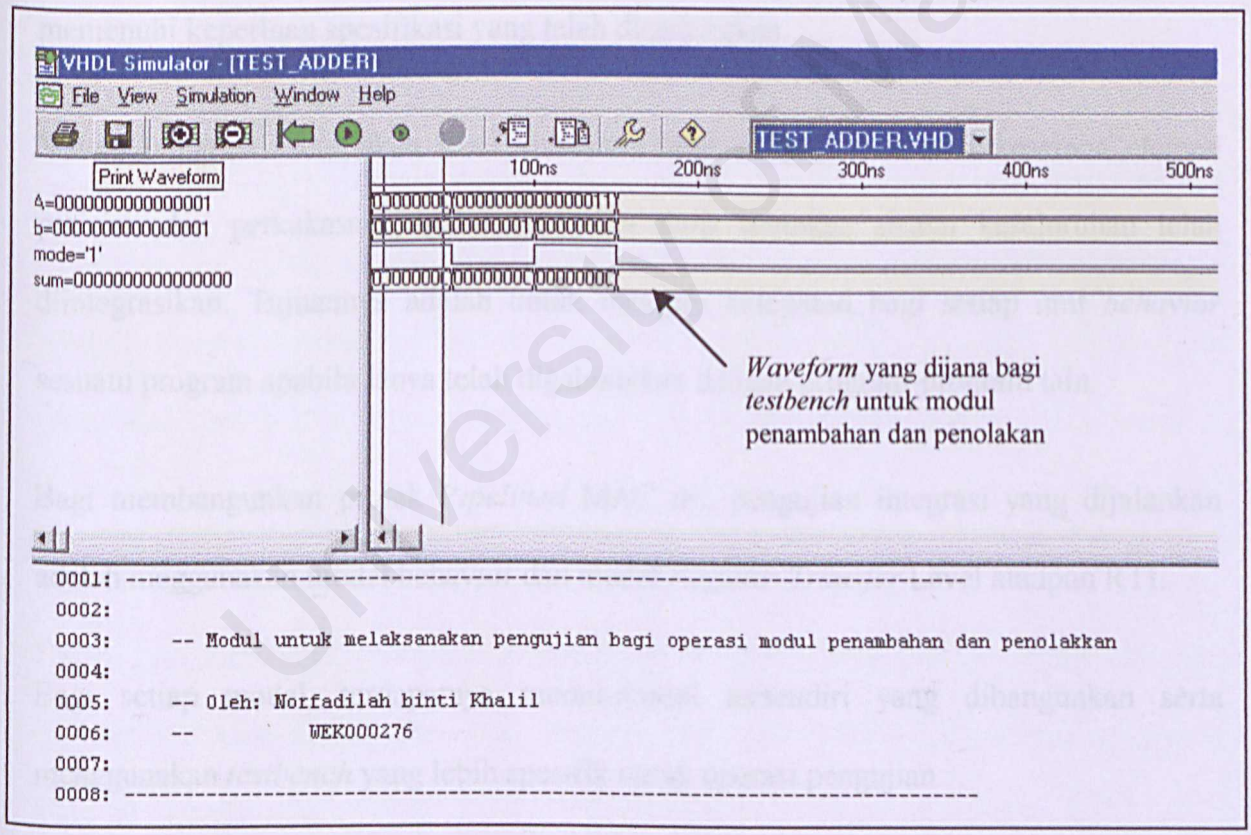
6.3.1.2 Pendaraban Menggunakan Pendekatan Algorithma Booth

Seperti pendaraban menggunakan kaedah biasa, modul ini juga menggunakan *testbench* dan *waveform* untuk melaksanakan pengujian unit. Anjakan bagi input a diperiksa dimana sekiranya nilai bagi input a ialah '01' maka, *multiplicand* akan dianjak sekali dan ditambah kepada hasil pendaraban sebelumnya. Manakala sekiranya input bagi a adalah '10', maka, *multiplicand* akan ditolak daripada hasil penambahan sebelumnya.

Jika nilai a adalah '11' dan '00' , maka, tiada apa-apa perubahan dilaksanakan pada hasil pendaraban sebelumnya.

6.3.2 Operasi Penambahan dan Penolakan

Input sensitif bagi modul ini ialah *mode*. *Mode* merupakan kawalan perlaksanaan operasi bagi penambahan atau penolakan. Sekiranya nilai *mode* adalah '1', maka, modul akan melaksanakan operasi penolakan sementara sekiranya nilai *mode* adalah '0', maka operasi penambahan akan dilaksanakan terhadap kedua-dua input.



Rajah 6.2 : Waveform yang dihasilkan daripada testbench bagi Modul Penambahan dan Penolakan

6.4 Integrasi dan Pengujian Sistem

Pengujian sistem ialah siri pengujian yang merangkumi ujian-ujian yang berbeza dimana objektif utamanya adalah untuk mengenalpasti factor-faktor penghad sistem dan mengoptimakan kebolehan kebolehan penggunakan sistem. Objektif utama pengujian system ialah untuk mengintegrasikan seluruh sistem bagi mengesahkannya telah memenuhi keperluan spesifikasi yang telah dicadangkan.

Sistem integrasi merupakan hasil kemajuan bagi pengujian dimana kesemua elemen perisian dan perkakasan digabungkan dan diuji sehingga sistem keseluruhan telah diintegrasikan. Tujuannya adalah untuk mengira ketepatan bagi setiap unit *behavior* sesuatu program apabila ianya telah digabungkan dengan program-program lain.

Bagi membangunkan projek *Pipelined* MAC ini, pengujian integrasi yang dijalankan adalah meggunakan model behavior dan model *Register Transfer* Level ataupun RTL.

Bagi setiap model, terdapatnya modul-modul tersendiri yang dibangunkan serta menggunakan *testbench* yang lebih spesifik untuk operasi pengujian.

Keputusan bagi model-model tersebut hasil penggunaan *testbench* akan di bandingkan. Walaubagaimanapun, pendekatan yang terbaik ialah menggunakan *testbench* yang berbeza yang memenuhi keperluan spesifik bagi model-model tersebut. Dengan cara ini, lebih mudah untuk membandingkan keputusan model *behavior* dan model RTL.

Kesimpulan yang dapat dibuat ialah kaedah pengujian yang digunakan dalam projek *Pipelined* MAC ialah *testbench. Testbench* ialah model struktur yang mempunyai dua komponen penting *model under test* dan *tester. Model under test* mungkin terdiri daripada *behavioral* atau struktur model VHDL bagi sistem digital.

PeakFPGA digunakan untuk membina *testbench* bagi setiap modul. Kegunaan utama *testbench* ialah untuk menguji input dan output yang dieberikan untuk pengujian. Simulasi dilaksanakan keatas *testbench* untuk memastikan output proses adalah betul.

Sementara itu, pengujian sebenarnya dilakukan dalam dua peringkat. Pertama ialah pengujian unit yang menguji unit-unit dalam modul secara individu. Kedua ialah integrasi dan pengujian sistem yang dilaksanakan selepas proses penggabungan antara program-program kecil yang akan membentuk keseluruhan sistem. Cara yang lebih mudah untuk mengesan kesalahan ialah melalui pengujian unit berbanding pengujian integrasi sistem.

Dalam bab seterusnya, penilaian terhadap sistem akan dibuat dimana segala masalah, kekangan, kekuatan dan kelemahan system ini serta untuk kajian dimasa-masa hadapan akan dibincangkan.

BAB TUJUH

PENILAIAN SISTEM

7.0 Penilaian Sistem

7.1 Pengenalan

Penilaian sistem merupakan salah satu cara untuk membuat menilai sistem secara keseluruhan, mengenalpasti semua masalah yang telah dihadapi sepanjang melaksanakan projek *Pipelined* MAC.

Dalam bab ini juga turut dibincangkan tentang semua kekangan yang harus dipertimbangkan dalam melaksanakan projek ini. Kelebihan serta kekurangan yang ada pada sistem ini dibincangkan bagi membolehkan penyelidikan serta kajian yang lebih lanjut pada masa-masa hadapan.

7.2 Penakrifan Masalah dan Penyelesaiannya

Pelbagai masalah dihadapi bagi merealisasikan projek ini menjadi kenyataan. Antara masalah yang pernah dihadapi ialah penggunaan bahasa pengaturcaraan VHDL yang merupakan agak asing. Tetapi, hasil daripada kesabaran dan ketekunan semua pihak, langkah awal mengatasinya ialah dengan membina model mudah VHDL seperti *register, fulladder, d flip-flop* dan lain-lain lagi.

Pemahaman untuk menguasai algorithma yang dicadangan dalam projek seperti algorithma Booth yang memerlukan kefahaman secara menyeluruh. Jadi, pelbagai cara pemahaman digunakan antaranya ialah mencari sumber atau tajuk yang berkenaan di halaman-halaman web serta memperbanyakkan pembacaan tentang subjek tersebut daan merujuk pensyarah bagi mendapatkan penerangan mengenainya dengan lebih jelas lagi.

Dalam penguasaan penggunaan *PeakFPGA* , ianya mengambil masa untuk dikuasai sepenuhnya. Setiap langkah harus dikuasai menguasai pemahaman terhadap perisian tersebut serta bahasa pengaturcaraan VHDL. Penambahan mengenai bahasa pengaturcaraan VHDL semakin bertambah setiap kali pendekatan lain dicuba dalam perisian *PeakFPGA*.

Namun, apa yang menyedihkan ialah projek ini tidak dapat disentisiskan kerana ianya tidak lengkap.

7.3 Kekuatan Sistem

Pipelined MAC boleh meningkatkan kelajuan bagi pemprosesan isyarat digital dengan mengimplementasikan beberapa teknik seperti :

- *Booth algorithm* dalam unit pendaraban

- *Carry LookAhead adder* dalam unit *accumulator*
- *Pipelining* bagi *Multiplier Accumulator*

Pipelined MAC membawa pelbagai pembaharuan berbanding *Conventional* MAC. Perbandingan kedua-dua system ini pernah dibincangkan pada bab 2. Bagi penerangan yang lebih terperinci, rujuk bahagian 2.4.1 Perbandingan Antara *Conventional* MAC dengan *Pipelined* MAC.

Pipeline MAC menggunakan banyak pendekatan dan pembaharuanberbanding *Conventional MAC. Pipeline MAC* dipercayai boleh meningkatkan kelajuan operasi dengan melaksanakan beberapa kerja dalam suatu masa menggunakan pendaftar sebagai tapak simpanan sementara.

7.4 Kekangan Sistem

Secara teorinya, system ini dapat dibangunkan sepertimana yang telah dicadangkan tetapi, kerana kekompleksan dalam membina modul pendaraban menggunakan pendekatan algorithma Booth VHDL, sistem tidak dapat dilengkapkan. Namun begitu, sistem berjaya dilaksanakan sekiranya masa yang diperuntukkan bagi menyiapkan projek ini dipanjangkan agar dapat menyiapkan projek ini dengan lebih sempurna lagi dan dapat membuat kajian dengan lebih mendalam.

Disamping itu, kekangan kemahiran dalam penggunaan bahasa pengaturcaraan VHDL. Dalam mendalami bahasa tersebut, penggunaan sintaks dan penjanaan fungsi perlu diketahui bagi mengelakkan sebarang kesilapan.

7.5 Future Enhancements

Sepertimana yang dimaklumkan pada bahagian sebelum ini, projek ini tidak dapat direalisasikan sepenuhnya kerana peruntukkan masa yang diberikan adalah singkat. Diharap, agar penyelidikan dan kajian mengenai pendekatan algortihma Booth dapat diteruskan pada masa-masa hadapan bagi membolehkan sistem ini dilaksanakan sepenuhnya.

Pengiraan secara manual bagi algoritma Booth ditunjukkan dalam bahagian lampiran. Rujuk lampiran 3. Kegagalan algoritma ini mungkin berpunca daripada kesilapan dalam pembinaan modulnya dan teknik serta pendekatan yang dikaji agak tidak menepati algoritma tersebut.

Projek ini mungkin dapat diadaptasikan untuk menjadi *filter* atau penapis. Ini kerana, fungsi penapis adalah memisahkan isyarat yang dikehendaki dengan isyarat yang tidak dikehendaki. Penapis berfungsi sebagai penghalang isyarat gangguan. Penapis digunakan dalam menghasilakan imej yang lebih baik dan jelas dan untuk tujuan merekod data-data analog seperti suara dan lain-lain.

Perkaitan *filter* yang dapat dilihat ialah dengan menggunakan *Pipelined MAC*, isyarat gangguan dapat diasingkan kerana menggunakan nombor-nombor kompleks sebagai input.

Pipelined MAC juga boleh digunakan dalam *equalization* iaitu untuk menyamakan isyarat bagi menghasilkan isyarat penghantar yang lebih baik dan juda dalam *demodulation.*

Bagi mengimplimentasi system ini secara menyeluruh, pendekatan pemprosesan isyarat digital harus digunakan. Isyarat digital digunakan untuk mengintegrasikan seluruh sistem iaitu penggabungan semua modul; pendaraban, penambahan dan *pipeline*. Bagi melaksanakannya, suatu unit kawalan diperlukan sebagai pemproses pusat yang mengawal kesegerakkan serta perjalanan pengambilan input bagi menjana output.

7.6 Pengetahuan dan Pengalaman yang Diperolehi

Pengalaman yang terpenting yang diperolehi sepanjang melaksanakan sistem ini ialah pendedahan dalam melaksanakan dan mengaturcara program terutamanya melaksanakan bahasa pengaturcaraan VHDL dalam membangunkan sistem ini.

Bagi menguasai bahasa pengaturcaraan tersebut, seharusnya penggunaan perisian seperti *PeakFPGA* diperlukan untuk tujuan menkompil dan mensimulasikan bahasa pengaturcaraan tersebut.

Pengetahuan yang paling bermakna yang telah ditimba ialah semasa proses merekabentuk sistem digital yang boleh dijadikan pengalaman bagi diimplementasikan pada dunia kerjaya sebenar.

Pengalaman yang tidak kurang pentingnya ialah berkerja secara berkumpulan dimana perbincangan, bantuan serta toleransi diperlukan bagi menghasilkan projek yang lebih kreatif dan menyelesaikan sesuatu masalah dengan lebih baik.

Pengalaman yang dapat dipelajari ialah bagaimana mengentalkan hati dan terus tabah dalam menyiapkan projek ini. Semasa menghadapi kegagalan, khidmat nasihat penyelia telah berjaya menguatkan semangat untuk meneruskan projek ini hingga ke titisan terakhir.

Dalam penjanaan laporan pula, pengalaman yang ditimba ialah dapat menyediakan cadangan projek untuk masa-masa hadapan. Cara dan langkah-langkah untuk menjana laporan ini sedikit sebanyak membantu serta mengajar tentang cara-cara menyiapkan projek dengan lengkap, mengikut spesifikasi yang diberikan dan menyiapkan dalam masa yang sesuai.

7.7 Kesimpulan dan Perbincangan

Secara globalnya, projek *Pipelined MAC* ini dirangka bagi mengelakkan lengahan yang dihadapi oleh sistem *Conventional MAC*. Algoritma Booth, teknik penambahan *Carry Look Ahead* dan penggunaan *pipelined* digunakan untuk diimplimentasikan dan dilaksanakan dalam operasi penambahan dan pendaraban dalam MAC. Dengan pendekatan-pendekatan tersebut, objektif pembinaan sistem ini dapat dicapai.

Dalam melaksanakan sistem ini, banyak masalah telah timbul. Masalah utama yang dihadapi ialah peruntukkan masa yang diberikan tidak mencukupi disamping kesukaran dalam menguasai bahasa pengaturcaraan VHDL serta pemahaman berkenaan algoritma-algoritma yang terlibat.

Diharap, pada masa-masa hadapan, kajian terhadap penggunaan serta implementasi sistem ini dalam kehidupan dapat diteruskan. Kajian terhadap penggunaan *Pipelined MAC* dalam penapis, *demodulizer* dan *equalizer* patut diteruskan. Disamping itu, penggunaan pemprosesan isyarat digital patut diambil-kira bagi menghasilkan sistem yang lebih efisyen.

Banyak pengalaman serta pengetahuan yang didapati sepanjang melaksanakan projek ini. Diantaranya ialah pembahagian masa, pengetahuan menggunakan perisian bagi

melaksanakan modul serta mengetahui sintaks dan aturcara menggunakan VHDL dan cara-cara merialisasikan algoritma-algoritma yang terlibat kedalam bahasa pengaturcaraan tersebut.

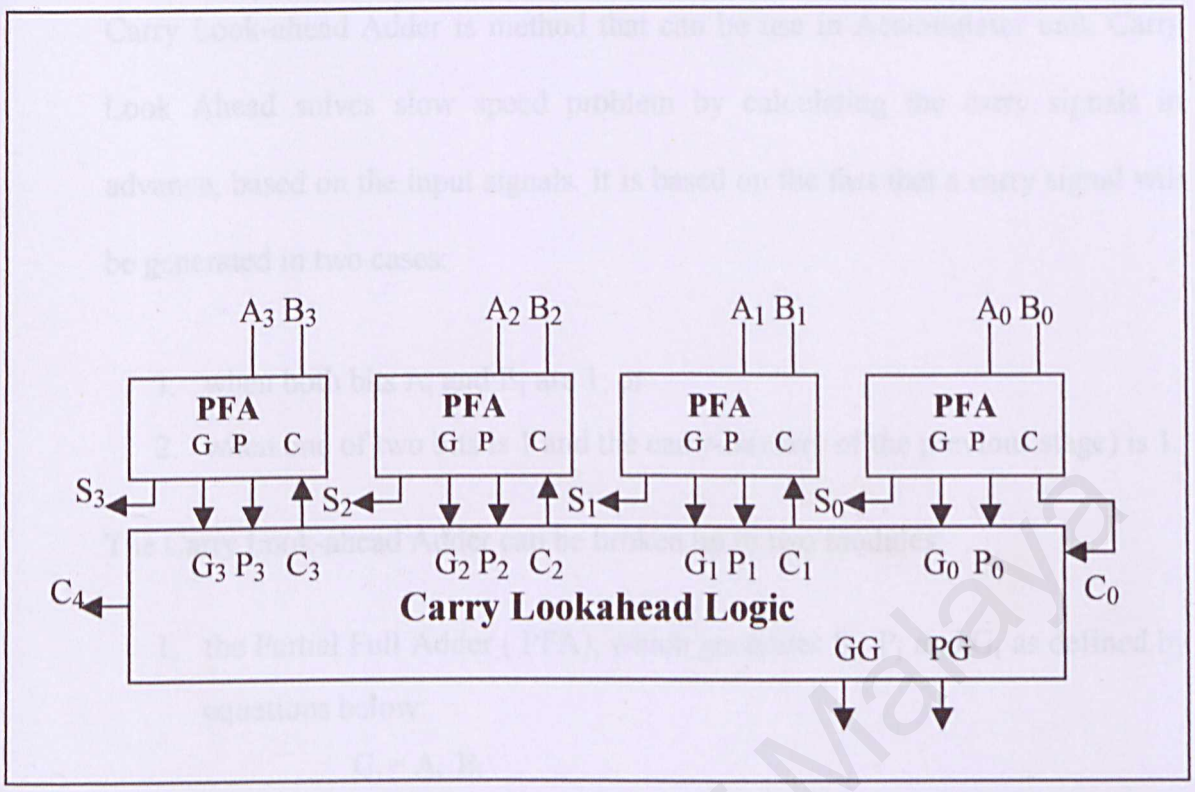
Kesimpulannya, projek ini mempunyai beberapa peningkatan berbanding sistem sebelumnya. Pembangunan algoritma yang telah dilaksanakan amat penting bagi merealisasikan sistem ini.

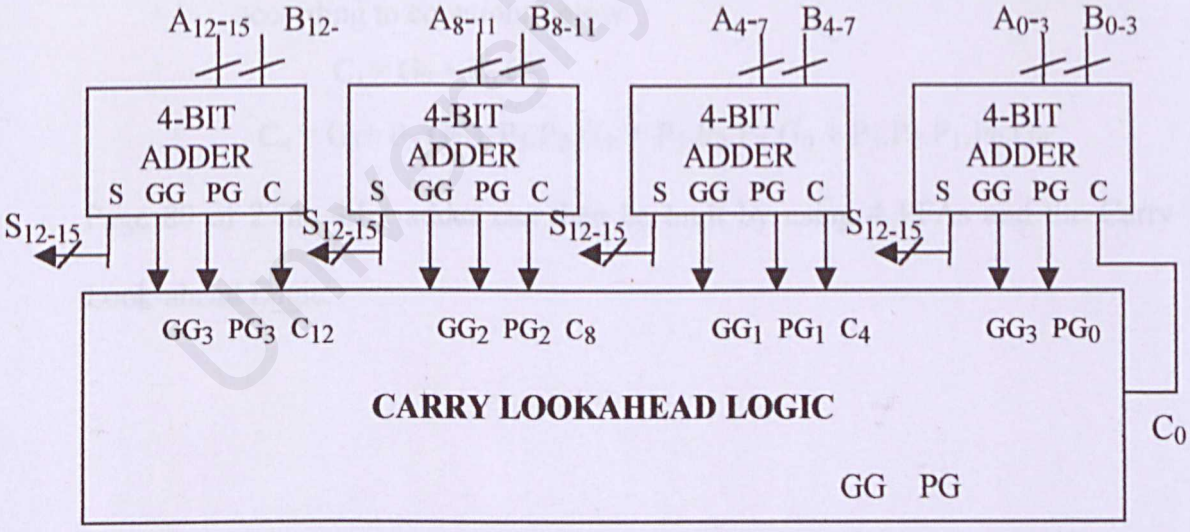
Disamping itu, pelbagai kekangan dikenalpasti dan sistem ini juga dapat membantu dalam memperluaskan pengalaman serta pengetahuan dalam membina sistem dan penggunaan bahasa pengaturcaraan VHDL.

LAMPIRAN

LAMPIRAN 1

Gambarajah Logik Penambahan *Carry Look Ahead*





Logik Penambah *Carry Look Ahead*

Carry Look-ahead Adder is method that can be use in Accumulator unit. Carry Look Ahead solves slow speed problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases:

1. when both bits A_i and B_i are 1, or
2. when one of two bits is 1 and the carry-in(carry of the previous stage) is 1.

The Carry Look-ahead Adder can be broken up in two modules:

1. the Partial Full Adder (PFA), which generates S_i, P_i and G_i as defined by equations below:

$$G_i = A_i . B_i$$

$$P_i = (A_i \$ B_i)$$

$$S_i = A_i \$ B_i \$ C_i = P_i \$ C_i$$

2. the Carry Look ahead Logic, which generates the carry-out bits according to equations below:

$$C_1 = G_0 + P_0 C_0$$

$$C_4 = G_3 + P_3 G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.G_0$$

Page 80 of 2The 4-bit adder can then be built by using 4 PFAs and the Carry Look-ahead Logic.

LAMPIRAN 2

Langkah-Langkah Simulasi Menggunakan

PeakFPGA

Menu Option for Simulation Using PeakFPGA

Compiled Selected

To compile selected VHDL modules, do the following :

- Select the module to be compiled by clicking on it once in the Hierarchy Browser.
- Select Options / Compile... from the menu bar to bring up the Compile Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button. Set compile options as needed. Click on the Close button to close the dialog.
- Select the Simulate / Compile Selected option from the menu bar or click on the Compile Selected Module toolbar button. The selected module is then compiled.

Link Selected

To link modules, do the following :

- Select the module, entity, or architecture representing the top level for the link operation by clicking on the appropriate item once in the Hierarchy Browser.
- Select Options / Link... from the menu bar to bring up the Link Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Link folder tab..Once

the dialog is displayed, set link options as needed. Click on the Close button to close the dialog.

- Select the Simulate / Link Selected option from the menu bar or click on the Link Selected Module toolbar button. The link operation then takes place.

Load selected

To load a selected simulation executable, do the following:

- Select the module, entity, or architecture you wish to load by clicking on the appropriate item once in the Hierarchy Browser.
- Select Options / Simulation... from the menu bar to bring up the Simulation Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Simulation folder tab..Once the dialog is displayed, set simulation options as needed. Click on the Close button to close the dialog.
- Select the Simulate / Load Selected option from the menu bar or click on the Load Selected Simulation Executable toolbar button. The PeakSIM application is then invoked and the selected simulation executable is loaded.

Options

To set Simulation options, select Options / Simulation... from the menu bar to bring up the Simulation Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Simulation folder tab..Once the dialog is displayed, set simulation options as needed. The various simulation options are discussed below :

- Update simulation executable before loading - If this option is checked, the Link process will be invoked if the simulation executable is out of date (as determined by checking the date and time stamps of the object files).
- Vector display format - This pull-down list allows you to specify the vector data display format for the waveform. Use the list to select binary, octal, decimal, or hexadecimal.
- Run to time - This field shows the default duration for the simulation run. You can reset this value by clicking on the Run to Time field and typing in a new value. This value can be overridden for individual simulation runs as needed by changing the value in the GO field in the Waveform Display.
- Step value - This field shows the default step time interval for a step simulation run. You can reset this value by clicking on the Step Value field and typing in a new value. This value can be overridden for individual step simulation runs as needed by changing the value in the Step field in the Waveform Display.
- Unit - This field shows the unit of time to be used during simulation. To select a different unit of time, click on the Unit field to display the various options. Then click on the desired unit to select it. Valid units of time are those units defined by

the VHDL language are fs (femtosecond), ps (picosecond), ns (nanosecond), us (microsecond), ms (millisecond), sec (second), min (minute) and hr (hour).

- Max signal depth - This field specifies the depth of signals to be loaded for into the Available Signals list in the Waveform Display. The depth of a signal is determined by its position in the design hierarchy. For example, a signal DUT.Clk has a signal depth of 2, while signal DUT.U1.ControlSM.Var1 has a depth of 4. You can use this option to reduce the number of signals and speed simulation loading when simulating large structural models.
- When you are finished setting options, click on the Close button to close the dialog.

LAMPIRAN 3

Pendaraban Manual Menggunakan Algoritma Booth

Algoritma Booth

Algoritma Booth adalah algoritma pendaraban yang melibatkan nombor-nombor pelengkap duaan. Ia adalah sama dengan kaedah pendaraban biasa, tetapi, pendaraban dengan menggunakan pendekatan algoritma ini merujuk kepada bit terkini dan bit terdahulu untuk menentukan apa yang harus dilaksanakan.

Syarat-syarat dalam melaksanakan Algoritma Booth ialah:

- Sekiranya bit semasa adalah satu dan bit sebelumnya ialah 0 (10) laksanakan anjakan dan bit tanda bagi *multiplicand* di *extend* kemudian ditolak dengan keputusan sebelumnya.
- Sekiranya dalam pasangan 01, tambah kepada keputusan sebelumnya.
- Sekiranya jujukan 11 atau 00, tiada apa-apa yang akan dilaksanakan.

Contohnya:

Pendaraban 8 bit:

```
          00000110      <- 6
x         00000010      <- 2
-----------------------------------
      0000000000000000
-          00000110
-----------------------------------
      1111111111110100
+         00000110
-----------------------------------
(1)   0000000000001100  <- 12 (bit limpahan diabaikan)
```

Nilai tersebut didapati kerana:

0000110 x 00000010 = 00000110 x (-00000010 + 00000100) = -000001100 + 00000011000 = 00001100

Dalam algoritma Booth, sekiranya *multiplicand* dan pendarab dalam nombor n-bit pelengkap duaan, keputusan atau hasil pendaraban di andaikan dalam nilai nombor pelengkap duaan 2n-bit. Nilai limpahan (diluar 2n bit) diabaikan.

Pendaraban yang melibatkan nombor negatif

(-5) x (-3) :

```
        10000011      -> -5
x       10000101      -> -3
-----------------------------------
    0000000000000000
-           11111011
-----------------------------------
    0000000000000101
+   1111111111111011
-----------------------------------
    1111111111111011
-   11111111111011
-----------------------------------
    1111111111111111      -> +15
```

Contoh Pendaraban Panjang

```
          10011100      <- -100
x         01100011      <- 99
-----------------------------------
     0000000000000000
-    1111111110011100
-----------------------------------
     0000000001100100
+    11111110011100
-----------------------------------
     1111111011010100
-    11110011100
-----------------------------------
     0000101101010100
+    110011100
-----------------------------------
     1101100101010100      <- -9900
```

Dapat dilihat pada contoh diatas, hasil pendaraban melibatkan 16 bit nombor dalam pelengkap duaan.

LAMPIRAN 4

Pengkodan Sistem

Contoh Pengekodan untuk Modul Pendaraban biasa.

```
--------------------------------------------------------------------------------

    -- Modul untuk melaksanakan pengujian bagi operasi pendaraban

    -- Oleh: Norfadilah binti Khalil
    --           WEK000276

--------------------------------------------------------------------------------
-- modul ini akan melaksanakan operasi pendaraban
-- melibatkan proses yang sensitif kepada dua input

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;


entity MULTIPLIER is
    port (
        a: in std_ulogic_vector(7 downto 0);  -- input masukkan yang mempunyai 8 bit
        b: in std_ulogic_vector(7 downto 0);
        p: out std_ulogic_vector(15 downto 0) -- keluaran yang menjana 16 bit (hasil darab
8X8)
    );

end MULTIPLIER;

architecture BEHAVIORAL of MULTIPLIER is
begin

    behavior : process(a,b) is
    -- proses yang sensitif pada kedua-dua masukan iaitu a dan b
    -- bila ada perubahan pada input, proses akan melaksanakan operasi pendaraban

    constant Tpd_in_out : time := 40ns;
    -- tempoh yang perlu ditunggu untuk menjana output

    variable negative_result : boolean;
    variable op1 : std_ulogic_vector ( 7 downto 0);
    variable op2 : std_ulogic_vector ( 7 downto 0);
    variable result : std_ulogic_vector (15 downto 0);
    variable carry_in, carry : std_ulogic;

   begin

    op1 := a;         -- mewakili pendarab
    op2 := b;         -- mewakili multiplicand


    -- make both operands positive, remembering sign of result

        negative_result := (op1(7) = '1') xor (op2(7) = '1');      -- untuk digunakan
kemudian dalam menentukan bit tanda

    if (op1(7) = '1') then        -- pengujian bagi memastikan nombor bagi multiplicand
tersebut adalah negatif atau tidak
            carry := '1';         -- anggap carry sebagai 1
            for index in 0 to 7 loop
                carry_in := carry;
                carry := carry_in and op1(index);
                op1(index) := not op1(index) xor carry_in;
            end loop;             -- gelung bertujuan untuk melaksanakan pelengkap duaan
bagi menukarkan nombor negatif bagi input a
        end if;


        if (op2(7) = '1') then
        carry := '1';
```

```
            for index in 0 to 7 loop
                carry_in := carry;
                carry := carry_in and op2(index);
                op2(index) := not op2(index) xor carry_in;
            end loop;        -- gelung bertujuan untuk melaksanakan pelengkap duaan bagi
menukarkan nombor negatif bagi input b
        end if;

        -- melaksanakan operasi pendaraban yang panjang

        result := (others => '0');    -- sekiranya nilai yang lain adalah sifar; nombor
adalah positif dan tidak perlu melaksanakan apa-apa perubahan keatas nombor tersebut

        for count in 0 to 7 loop
            carry := '0';
            if (op2(count) = '1') then
                for index in 0 to 7 loop
                    carry_in := carry;
                    carry := (result(index+count) and op1(index))
                             or (carry_in and (result(index+count) xor op1(index)));
                    result(index+count) := result(index+count) xor op1(index) xor
carry_in;
                end loop;    -- gelung melaksanakan operasi penambahan bagi melaksanakan
operasi pendaraban.
                             -- menambahkan setiap kali masukan bit untuk menghasilkan
partial product
                    result(count+8) := carry;
            end if;
        end loop;



        if negative_result then
            carry := '1';
            for index in 0 to 15 loop
                carry_in := carry;
                carry := carry_in and not result(index);
                result(index) := not result(index) xor carry_in;
            end loop;    -- gelung untuk melaksanakan operasi penukaran semula nombor
negatik kepada bentuk yang asal.

        end if;

        p <= result after Tpd_in_out;

        -- product diperolehi selepas 40ns.

    end process behavior;


end BEHAVIORAL;
```

Contoh Pengekodan untuk Modul Pendaraban menggunakan pendekatan Algoritma Booth.

```
--------------------------------------------------------------------------

    -- Modul untuk melaksanakan algorithma Booth

    -- Oleh: Norfadilah binti Khalil
    --          WEK000276

--------------------------------------------------------------------------
-- modul untuk melaksanakan algorithma Booth dalam merekabentuk modul pendaraban
    pipelined MAC
-- algorithma Booth merupakan algoruthma pendaraban yang boleh melaksanakan pendaraban
keatas nombor2 pelengkap duaan
-- akan lihat bit semasa dengan bit terdahulu untuk menentukan apa yang perlu dilakukan
-- langkah-langkah perlaksanaanya:
--       sekiranya jujukan bit semasa dan bit terdahulu ialah 10, anjak dan setkan
multiplicand untuk penolakkan dan penambahan bit tanda di depan untuk dapatkan 16 bit
output
--       sekiranya jujukan bit semasa dan bit terdahulu ialah 01, tambah dengan keputusan
yang diperolehi
--       sekiranya jujukan 00 dan 11 diperolehi, tiada perubahan.

-- dalam algorithma ini, sekiranya multiplicand dan pendarab adalah dalam pelengkap
duaan, maka produk juga dalam nilai
-- pelengkap duaan. Bit limpahan diabaikan.

-- pendarab mengikut algorithma Booth
--      a  anjak    bb
--  i i-1        pendarab, anjak produk separa satu tempat bagi setiap stage.
--  0  0   0     tidak lakukan apa2
--  0  1   +b    penambahan
--  1  0   -b    penolakkan
--  1  1   0     tidak lakukan apa2


library ieee;
use ieee.std_logic_1164.all;

entity BOOTH_ALGO is
    port (
        a: in std_logic_vector(1 downto 0);          --booth multiplier
        b: in std_logic_vector(7 downto 0);          -- multiplicand
        sum_in: in std_logic_vector(7 downto 0);     -- sum input
        sum_out: out std_logic_vector(7 downto 0);   -- sum output
        prod: out std_logic_vector ( 1 downto 0)                          -- satu bit
produk
    );

end BOOTH_ALGO;

architecture circuit of BOOTH_ALGO is

subtype word is std_logic_vector(7 downto 0);
signal bb           : word;
signal psum         : word;
signal sum          : word;
signal b_bar     : word;
signal two_b     : word;          -- untuk melaksanakan anjakan
signal cout         : std_logic;
signal cin          : std_logic;
signal topbit       : std_logic; -- bit teratas
signal topout       : std_logic; -- bit teratas sebagai output
signal ncl          : std_logic; -- untuk temporary/ dummy

begin
two_b <= b(6 downto 0) & '0';-- anjakan 1 tempat
b_bar <= not b;      -- 1's complements for multiplicand
bb <= b when a= "01"             -- umpukkan nilai b kepada bb apabila a adalah bernilai
01(penambahan)
```

```
        else b_bar when a="10"    -- umpukkan nilai b_bar kepada bb apabila a adalah
bernilai 10(penolakan)
        else x"00";               -- umpukkan nilai x'00' kepada bb apabila a adalah
bernilai 11 atau 00 (tiada perubahan)

cin <= '1' when a="10"            -- untuk operasi tolak; perlukan penambahan bit 1;
umpukkan nilai 1 pada cin bila operasi penolakkan
            else '0';             -- tiada perubahan untuk nilai cin bila a adalah
01,00,11

topbit <= b(7) when a = "01"                    -- bit teratas b diumpukkan kepada topbit
bila a = 01
                else b_bar(7) when a = "10" -- bit teratas b_bar diumpukkan kepada
topbit bila a = 10
                else '0';                       -- bit teratas 0 diumpukkan kepada
topbit bila a = 11 atau 00

-- untuk laksanakan operasi penambahan; gunakan add8 dan fulladder
a1 : entity work.add8 port map(sum_in,bb,cin,psum,cout);                              --
petakan add8 kepada booth_algo
a2 : entity work.fulladder port map(sum_in(7), topbit, cout, topout, nc1);     -- petakan
fulladder kepada booth_algo


sum_out (5 downto 0) <= psum(7 downto 2);
sum_out (7) <= topout;
sum_out (6) <= topout;
prod <= psum ( 1 downto 0);

end circuit;
```

```
--------------------------------------------------------------------------

    -- Modul untuk melaksanakan pendaraban yang menggunakan
    -- pendekatan algorithma Booth

    -- Oleh: Norfadilah binti Khalil
    --          WEK000276

--------------------------------------------------------------------------

-- modul untuk melaksanakan operasi pendaraban ke atas
-- multiplicand dan multiplier
-- menggunakan sub modul booth_algo untuk petakan dengan
-- hasil pendaraban 16 bit

library ieee;
use ieee.std_logic_1164.all;


entity BOOTH_MUL is
    port (

        a: in std_logic_vector(7 downto 0);   -- multiplier
        b: in std_logic_vector(7 downto 0);   -- multiplicand
        p: out std_logic_vector(15 downto 0)  -- product

    );

end BOOTH_MUL;

architecture CIRCUITS of BOOTH_MUL is

    signal zero : std_logic_vector(7 downto 0) := x"00";  -- kosong
    signal mul0 : std_logic_vector(2 downto 0);
    subtype word is  std_logic_vector(7 downto 0);
    type ary is array (0 to 7) of word;
    signal s : ary;       -- temp sum
begin

    mul0 <= a(0) & '0';  -- append sifar pada awalan input a

    a0 : entity work.BOOTH_ALGO port map (
                mul0, b, zero, s(0), p(1 downto 0));
    a1 : entity work.BOOTH_ALGO port map (
                a(2 downto 1), b,s(0),s(1), p(3 downto 2));
    a2 : entity work.BOOTH_ALGO port map (
                a(3 downto 2), b,s(1),s(2), p(5 downto 4));
    a3 : entity work.BOOTH_ALGO port map (
                a(4 downto 3), b,s(2),s(3), p(7 downto 6));
    a4 : entity work.BOOTH_ALGO port map (
                a(5 downto 4), b,s(3),s(4), p(9 downto 8));
    a5 : entity work.BOOTH_ALGO port map (
                a(6 downto 5), b,s(4),s(5), p(11 downto 10));
    a6 : entity work.BOOTH_ALGO port map (
                a(7 downto 6), b,s(5),s(6), p(13 downto 12));
    -- a7 : entity work.BOOTH_ALGO port map (
    --             a(15 downto 13), b,s(6),s(7), p(15 downto 14));
    -- a8 : entity work.BOOTH_ALGO port map (
    --             a(17 downto 15), b,s(7),s(8), p(17 downto 16));
    -- a9 : entity work.BOOTH_ALGO port map (
    --             a(19 downto 17), b,s(8),s(9), p(19 downto 18));
    -- a10 : entity work.BOOTH_ALGO port map (
    --             a(21 downto 19), b,s(9),s(10), p(21 downto 20));
    -- a11 : entity work.BOOTH_ALGO port map (
    --             a(23 downto 21), b,s(10),s(11), p(23 downto 22));
    -- a12 : entity work.BOOTH_ALGO port map (
    --             a(25 downto 23), b,s(11),s(12), p(25 downto 24));
    -- a13 : entity work.BOOTH_ALGO port map (
    --             a(27 downto 25), b,s(12),s(13), p(27 downto 26));
    -- a14 : entity work.BOOTH_ALGO port map (
```

```
    --              a(29 downto 27), b,s(13),s(14), p(29 downto 28));
    -- a15 : entity work.BOOTH_ALGO port map (
    --              a(31 downto 29), b,s(14),p(63 downto 32),p(31 downto 30));


end CIRCUITS;
```

Contoh Pengekodan untuk Modul Penambahan dan Penolakan.

```
--------------------------------------------------------------------------

    -- Modul untuk melaksanakan operasi penambahan

    -- Oleh: Norfadilah binti Khalil
    --          WEK000276

--------------------------------------------------------------------------

-- modul akan melaksanakan operasi penambahan secara bersiri
-- menggunakan sub-modul fulladder untuk menghasilkan 16-bit keluaran.
-- modul ini juga melaksanakan operasi penolakkan binari.
-- menggunakan modul XOR untuk melaksanakan operasi pelengkap satuan(1st complement)
-- menambahkan 1 kepada pelengkap satuan untuk menghasilkan pelengkap duaan (second
complement)
-- modul ini juga merupakan top level design bagi modul adder and subtracter
-- apabila nilai mode ==1, maka, operasi tolak dilaksanakan
-- sekiranya nilai mode == 0, maka operasi tambah dilaksanakan

library ieee;
use ieee.std_logic_1164.all;

entity ADDER is
       port(
        a: in std_logic_vector(15 downto 0);  -- input
        b: in std_logic_vector(15 downto 0);
        mode : in std_logic;                  -- input yang akan menentukan perlaksanaan
operasi tambah atau tolak.
        sum: out std_logic_vector(15 downto 0);
        cout: out std_logic
    );
--  pengistiharan entiti

end ADDER;

architecture BEHAVIORAL of ADDER is

component FullAdder       -- menggunakan komponen modul fulladder untuk operasi penambahan
binari
port (  x: in std_logic;
        y: in std_logic;
        cin: in std_logic;
        cout: out std_logic;
        sum: out std_logic
        );
end component;
-- memetakan port masukan/keluaran bagi modul Adder dengan port masukan/keluaran komponen
fulladder

component hasil_XOR       -- menggunakan komponen modul hasil_XOR untuk operasi penolakkan
binari
port (
        X: in std_logic;
        Sub: in std_logic;
        XOR_out: out std_logic
    );
end component;
-- memetakan port masukan/keluaran bagi modul Adder dengan port masukan/keluaran komponen
hasil_XOR


signal C : std_logic_vector (15 downto 0);     -- menggunakan isyarat yang akan mewakili
setiap bit.
signal D : std_logic_vector (15 downto 0);

-- signal C mewakili bawaan masuk dan signal D mewakili sub(input yang bernilai negatif
atau positif)
```

```
begin

F0      : hasil_xor port map ( b(0), mode, D(0));                    -- petakan masukan
adder kepada fulladder dan modul hasil_xor
F0a     : fulladder port map ( a(0), D(0), mode, C(1),sum(0));
F1      : hasil_xor port map ( b(1), mode, D(1));
F1a     : fulladder port map ( a(1), D(1), C(1), C(2),sum(1));
F2      : hasil_xor port map ( b(2), mode, D(2));
F2a     : fulladder port map ( a(2), D(2), C(2), C(3),sum(2));
F3      : hasil_xor port map ( b(3), mode, D(3));
F3a     : fulladder port map ( a(3), D(3), C(3), C(4),sum(3));
F4      : hasil_xor port map ( b(4), mode, D(4));
F4a     : fulladder port map ( a(4), D(4), C(4), C(5),sum(4));
F5      : hasil_xor port map ( b(5), mode, D(5));
F5a     : fulladder port map ( a(5), D(5), C(5), C(6),sum(5));
F6      : hasil_xor port map ( b(6), mode, D(6));
F6a     : fulladder port map ( a(6), D(6), C(6), C(7),sum(6));
F7      : hasil_xor port map ( b(7), mode, D(7));
F7a     : fulladder port map ( a(7), D(7), C(7), C(8),sum(7));
F8      : hasil_xor port map ( b(8), mode, D(8));
F8a     : fulladder port map ( a(8), D(8), C(8), C(9),sum(8));
F9      : hasil_xor port map ( b(9), mode, D(9));
F9a     : fulladder port map ( a(9), D(9), C(9), C(10),sum(9));
F10     : hasil_xor port map ( b(10), mode, D(10));
F10a    : fulladder port map ( a(10), D(10), C(10), C(11),sum(10));
F11     : hasil_xor port map ( b(11), mode, D(11));
F11a    :fulladder port map ( a(11), D(11), C(11), C(12),sum(11));
F12     : hasil_xor port map ( b(12), mode, D(12));
F12a    : fulladder port map ( a(12), D(7), C(12), C(13),sum(12));
F13     : hasil_xor port map ( b(13), mode, D(13));
F13a    : fulladder port map ( a(13), D(13), C(13), C(14),sum(13));
F14     : hasil_xor port map ( b(14), mode, D(14));
F14a    : fulladder port map ( a(14), D(14), C(14), C(15),sum(14));
F15     : hasil_xor port map ( b(15), mode, D(15));
F15a    : fulladder port map ( a(15), D(15), C(15), cout,sum(15));

-- penggunaan komponen hasil_xor dengan fulladder sebanyak 16 kali untuk menghasilkan
output
-- perlaksaan berdasarkan bit demi bit
end BEHAVIORAL;
```

RUJUKAN

BUKU

[Cavanagh, 84] Cavanagh, Joseph J.F., *Digital Computer Arithmetic Design And Implementation*, McGraw-Hill, Inc., 1984, page 98-233.

[Ash, 96] Ashenden, Peter. J., *The Designer's Guide to VHDL*, Morgan Kaufmann Publishers, Inc., 1996, page 161-188.

[Wayne, 98] Wayne Wolf, *Modern VLSI Design: System's on Silicon*, 2nd ed. Prentice-Hall, Inc., 1998, page 302,506.

[Boo, 51] Andrew D. Booth, "A signed binary multiplication technique, "*Quart. Journal of Mech. And Appl. Math*, Vol. IV, Pt. 2, page 236-240.

[William, 2000] William Stalling, *Computer Organization And Architecture*, 5th ed. Prentice-Hall International, Inc., 2000, page 423-425.

[Barry, 91] Wilkinson, Barry, *computer architecture design and performance*, Prentice-Hall International (UK) Ltd., 1991, page 102-143.

[LKS, 96] Lee Kap Soung (1996)., *The Pipelined Multiplier Accumulator in Digital Signal Processing*. Undergraduate Thesis. University of Malaya

LAMAN WEB

[ACC, 02] http://www.accu.org/acornsig/public/caugers/volume2/fixedpoint.html

[ANG, 02] http://www.angelfire.com/in/rajesh52/verilogvhdl.html

[ATM, 02] http://www.atmel.com/atmel/acrobat/doc0467.pdf

[CHI, 02] http://www.chipcenter.com/SearchResults.jhtml

[ECE, 96] http://www-ece.rice.edu/Courses/422/1996/supafly/adder.html

[ECS, 02] http://www.ecs.umass.edu/ece/koren/arith/simulator/Add/ripple.htm

[EET, 02] http://www.eetasia.com/ART_8800132735_499481,499485.HTM

[HOW, 02] http://www.howstuffworks.com/boolean2.htm

[MAT, 97] http://www.math.toronto.edu/mathnet/answers/imaghard.html

[RES, 02] http://research.microsoft.com/~hollasch/cgindex/coding/ieeefloat.html

[SEA, 02] http://www.seas.upenn.edu/~ee201/lab/CarryLookAhead

[SYN, 02] http://www.synopsys.com/products/designware/docs/doc/dwf/datasheet

[TRA, 02] http://www.traquair.com/articles/mousetrap.pdf

[TUD, 99] http://ce.et.tudelft.nl/~robbert/mac/