DISEDIAKAN OLEH:

LEORNA SHAZERINE BT ABD. MUTALIB

WEK000005

WXES 3182 : PROJEK ILMIAH TAHAP AKHIR II

TAJUK: LITAR ARITMETIK REJA

PENYELIA:

EN. NOORZAILY B. MOHAMED NOR

LITAR ARITMETIK REJA

ABSTRAK

Projek Ilmiah Tahap Akhir II, iaitu WXES 3182 adalah suatu latihan akademik didalam penyelidikan, rekabentuk, pembangunan dan komunikasi yang melibatkan prinsip-prinsip sains komputer. Projek WXES 3182 ini merupakan kursus pelengkap bagi pelajar-pelajar tahun akhir untuk mendapatkan penganugerahan Sarjana Muda Sains Komputer. Projek WXES 3182 ini adalah kesinambungan daripada projek WXES 3181. Kajian penyelidikan yang telah dijalankan adalah berkaitan dengan konsep sistem nombor reja dan litar aritmetik reja. Laporan latihan ilmiah tahap akhir II ini, merangkumi pembangunan projek WXES 3181 dan WXES 3182. laporan ini terdiri daripada lapan bahagian dan diantaranya ialah pengenalan kepada projek, kajian literasi, metodologi yang digunakan, dan rekabentuk litar . Setiap daripada bahagian-bahagian tersebut mengandungi huraian yang lebih terperinci berkenaan konsep sistem nombor reja dan litar aritmetik reja. Ini termasuklah beberapa huraian pengkodan VHDL yang mewakili sebahagian daripada komponen-komponen yang terdapat didalam litar aritmetik reja. Selain itu juga, terdapat beberapa rajah simulasi yang telah berjaya dilakukan bagi menyokong penyelidikan yang telah dijalankan. Kajian penyelidikan litar aritmetik reja ini juga telah dihurai didalam bentuk jurnal, iaitu salah satu daripada keperluan didalam pembangunan projek WXES 3182 ini. Hasil daripada kajian penyelidikan yang telah dijalankan, diharap dapat meningkatkan pemahaman kita tentang teknologi dan sains komputer.

PENGHARGAAN

Assalamualaikum w.r.b.t dan salam sejahtera.

Syukur kehadrat Allah s.w.t., kerana dengan limpah rahmatnya, dapat saya menyiapkan projek latihan ilmiah tahap akhir II ini. Terlebih dahulu, saya ingin mengambil kesempatan ini untuk mengucapkan rasa terima kasih dan penghargaan saya, kepada semua pihak yang terlibat, sama ada secara langsung atau tidak, bagi membantu saya menyempurnakan projek WXES 3182 ini. Penghargaan ini ditujukan khas kepada penyelia iaitu En. Noorzaily dan moderator, iaitu En.Zaidi yang telah bertanggungjawab didalam menyelia dan memantau setiap aktiviti yang saya lakukan disepanjang proses pembangunan projek WXES 3182 ini. Segala nasihat dan tunjuk ajar yang diberikan amat saya sanjung tinggi dan ribuan terima kasih. Penghargaan ini juga saya tujukan buat ibu yang tersayang, yang selama ini telah banyak mendorong saya untuk mencapai kejayaan sehingga ke hari ini. Tidak lupa, ucapan terima kasih ini juga ditujukan kepada rakan-rakan seperjuangan yang sama-sama mengharungi kepayahan untuk menjayakan pembangunan projek sehingga selesai. Diharap segala ilmu yang saya perolehi semasa membangunkan projek WXES 3181 & WXES 3182 ini, dapat digunakan dikemudian hari. Sebagai mengakhiri teks penghargaan ini, saya sudahi dengan lafaz alhamdulillah.

"Pisang Emas Di Bawa Belayar,

Masak Sebiji Diatas Peti,

Hutang Emas Boleh Dibayar,

Hutang Budi Dibawa Ke Mati".

ISI KANDUNGAN **MUKA SURAT**

BAB 4 : REKABENTUK LITAR ARITMETIK REJA

BAB 5 : PEMBANGUNAN LITAR ARITMETIK REJA

SENARAI JADUAL **MUKA SURAT**

SENARAI RAJAH **MUKA SURAT**

BAB 1 : PENGENALAN

1.1.0 Pengenalan Kepada Perkakasan Digital

Litar logik digunakan untuk membina perkakasan komputer, sepertimana kebanyakan produk elektronik yang lain. Produk-produk ini diklasifikasikan sebagai perkakasan digital. Teknologi yang digunakan untuk membina perkakasan digital telah meningkat dengan mendadak dan mengagumkan semenjak beberapa abad lepas. Sehingga tahun 1960-an, litar logik telah dibina dan pembinaannya memerlukan banyak komponen-komponen tunggal dan ruang seperti transistor dan perintang. Kemunculan litar bersepadu membolehkan penempatan beberapa transistor dan litar keseluruhannya, dibuat pada satu cip tunggal.

Pada peringkat awal, litar ini mempunyai beberapa transistor sahaja, tetapi dengan adanya perkembangan teknologi tinggi, bilangan transistor semakin bertambah. Cip litar bersepadu dibuat pada silicon wafer. Wafer ini dipotong untuk menghasilkan cip-cip tunggal dan kemudiannya ditempatkan didalam pakej cip tertentu. Pada tahun 1970-an, adalah menjadi suatu kenyataan untuk mengimplemenkan semua litar yang diperlukan bagi menghasilkan mikropemproses pada satu cip tunggal. Walaupun pada mulanya, mikropemproses telah mempunyai keupayaan pengkomputeran termoden dan dianggap memenuhi piawai sepertimana piawai pada masa kini, tetapi revolusi pemprosesan maklumat terus berkembang dan terbuka luas iaitu dengan terhasilnya komputer peribadi.

Kira-kira 30 tahun yang lalu, Gordon Moore, iaitu pengerusi Intel Corporation telah menyatakan bahawa teknologi litar bersepadu akan berkembang pada kadar yang mengagumkan, dimana terdapat pertambahan dua kali ganda bilangan transistor yang boleh ditempatkan pada satu cip bagi jangkamasa pada setiap satu setengah tahun ke dua tahun. Fenomena ini dikenali sebagai Moore's Law, dan ianya berterusan sehingga kini.

Diawal tahun 1990-an, mikropemproses boleh dihasilkan dengan beberapa juta transistor didalamnya dan dipenghujung tahun 90-an, adalah suatu kenyataan untuk membina cip yang mengandungi lebih daripada 10 juta transistor. Moore's Law dijangka akan berterusan sehingga ke dekad seterusnya. Konsortium pengilang litar bersepadu yang dikenali sebagai SIA telah membuat anggaran awal tentang perkembangan teknologi pada masa depan. Contoh kepada SIA Roadmap ini ditunjukkan seperti didalam jadual 1.1.

Jadual 1.1: SIA Roadmap

	Tahun					
	1999	2001	2003	2006	2009	2012
Panjang get transistor	0.14 μm	0.12 μm	0.10 μm	0.07 μm	0.05 μm	0.035 μm
Transistor per cm^2	14 juta	16 juta	24 juta	40 juta	64 juta	100 juta
Saiz cip	800 mm^2	850 mm^2	900 mm^2	1000 mm^2	1100 mm^2	1300 mm^2

1.1.1 Pengenalan Kepada Peranti Logik Teraturcarakan (PLD)

Cip ini mempunyai struktur biasa dan mengandungi koleksi suis teraturcarakan yang membenarkan litar dalaman didalam cip dikonfigurasi didalam kaedah yang berbeza-beza. Perekabentuk boleh mengimplementasikan segala fungsi aplikasi tertentu dengan memilih suis yang bersesuaian. Suis-suis ini diprogramkan oleh pengguna akhir. Kebanyakan PLD boleh diprogram beberapa kali. Ciri-ciri ini merupakan satu kelebihan kerana perekabentuk yang membangunkan prototaip produk boleh mengaturcara semula PLD tersebut. PLD wujud didalam julat saiz yang besar. Ianya boleh digunakan untuk

menghasilkan litar logik yang lebih besar berbanding cip biasa. Diantara kebanyakan PLD yang canggih ialah seperti FPGA (Field Programmable Gate Array). FPGA mengandungi lebih daripada 100 juta transistor. Cip ini mengandungi banyak elemen litar logik kecil yang boleh disambungkan bersama iaitu dengan menggunakan suis teraturcarakan. Elemen-elemen litar logi disusun didalam struktur dua dimensi.

1.1.2 Pengenalan Kepada FPGA

FPGA (Field Programmable Gate Array) merupakan suatu peranti teraturcarakan yang boleh dikonfigurasi pada pelbagai aplikasi. Peranti teraturcarakan yang digunakan dengan meluas adalah PROM(Programmable Read Only Memory). PROM wujud didalam dua versi asas iaitu Mask-Programmable Chip yang hanya boleh diprogram oleh pengeluar. Manakala Field-Programmable Chip pula boleh diprogram oleh pengguna akhir. Field Programmable PROM dibangunkan didalam dua jenis iaitu EPROM-Erasable Programmable Read Only Memory dan EEPROM-Electrically Erasable Programmable Read Only Memory. EEPROM mempunyai kelebihan boleh padam dan program semula berulang kali. MPGA-Mask Programmable Gate Array, dibangunkan untuk mengendalikan litar logik yang lebih besar. Didalam tahun 1985, Xilinx Inc. telah memperkenalkan FPGA. Saling hubungan antara setiap elemen direkabentuk untuk menjadikannya teraturcarakan pengguna.

1.1.3 Pengenalan Kepada Litar Aritmetik Reja

Litar aritmetik reja ini terdiri daripada gabungan litar darab-tambah dan litar penyemak ralat. Litar darab-tambah terdiri daripada komponen pendarab 8 x 8 bit dan penambah CLA. Manakala litar penyemak ralat pula terdiri daripada komponen pendarab aritmetik reja, komponen penambah aritmetik reja, komponen penukar dan komponen pembanding.. Didalam implementasi konvensional litar aritmetik reja, aritmetik perduaan dilaksanakan dan ini menyebabkan perambatan bawaan semasa penambahan dan akan menghadkan kelajuan operasi aritmetik reja. Oleh itu, untuk mengatasi masalah ini, satu konsep baru digunakan pada aritmetik reja iaitu perwakilan reja lewahan yang menggunakan sistem nombor digit bertanda, digit-P asas-2. Ia menggunakan integer 2^P, 2^{P+1} dan 2^{P-1} sebagai moduli bagi sistem nombor reja.

Bagi operasi penambahan modulo m, ianya dilaksanakan oleh komponen MSDA (penambah digit bertanda tanpa bawaan), iaitu penambah CLA. Manakala operasi pendaraban modulo m (MSDM) pula dilaksanakan oleh PPG (penjana darab separa) dan pokok penambah perduaan MSDA. Operasi pendaraban dan penambahan di litar darab-tambah dilakukan dengan menukarkan nombor integer kepada perwakilan nombor perduaan digit bertanda. Manakala pada litar penyemak ralat , operasi pendaraban modulo m dan penambahan modulo m dilakukan, iaitu dengan menukarkan nombor integer kepada perwakilan nombor reja digit bertanda. Hasil operasi pendaraban dan penambahan di litar darab-tambah (nombor perduaan digit bertanda) akan ditukarkan kepada perwakilan nombor reja digit bertanda.

Kedua-dua hasil darab-tambah ini akan disemak oleh komponen pembanding di litar penyemak ralat iaitu dengan melakukan perbandingan pada kedua-dua hasil tersebut. Berasaskan pada konsep ini, maka semakan ralat pengiraan pada litar aritmetik reja dapat dilakukan dengan pantas dan mengatasi kelemahan pada litar aritmetik reja yang terdahulu

yang menggunakan perwakilan nombor perduaan. Perwakilan nombor digit bertanda asas-2, algoritma penambahan modulo m, algoritma pendaraban modulo m ini dihuraikan didalam kod VHDL untuk merekabentuk litar aritmetik reja 8-digit.

1.2 Definasi Masalah

Sepertimana yang kita ketahui, masalah biasanya dikaitkan dengan kegagalan untuk mencapai objektif pembangunan sesuatu projek Oleh itu, adalah sangat penting untuk mengenalpasti masalah-masalah, sama ada masalah terdahulu atau masalah yang dijangkakan akan berlaku. Ini bertujuan supaya objektif pembangunan sesuatu projek itu dapat dicapai terutamanya apabila melibatkan pembangunan projek baru dengan menggunakan kaedah atau konsep yang berlainan dengan tujuan untuk mengatasi kelemahan projek sebelumnya. Fenomena ini boleh dilihat pada pembangunan litar aritmetik reja sedia ada, dimana perwakilan nombor perduaan telah digunakan. Hasil daripada pembangunan litar aritmetik konvensional tersebut, beberapa masalah telah dikenalpasti dan perlu diselesaikan, diantaranya seperti berikut:-

i. Perambatan bawaan

Masalah perambatan bawaan ini berlaku semasa operasi penambahan dijalankan, dimana ia mempengaruhi operasi pada litar aritmetik reja secara keseluruhannya. Perambatan bawaan ini telah menyebabkan kelajuan operasi aritmetik dan operasi penyemakan ralat menjadi perlahan.

ii. Kepadatan rekabentuk litar

Proses pembangunan cip mengambilkira rekabentuk sesuatu litar. Oleh itu, kita perlu memastikan bahawa rekabentuk litar aritmetik reja mestilah ringkas dan padat supaya penggunaan ruang dapat dijimatkan dan meningkatkan kelajuan operasi.

iii. Meminimakan penggunaan get-get logik

Litar yang baik adalah litar yang kurang menggunakan get-get logik tetapi dapat melaksanakan fungsi sesuatu litar itu dengan berkesan.

iv. Memastikan pembangunan projek disempurnakan mengikut perancangan

Pembangunan litar aritmetik reja ini mestilah dapat disempurnakan mengikut jadual proses kerja yang telah ditetapkan.

1.3 Skop Projek

Bahagian skop projek ini adalah untuk mengkhususkan permasalahan projek kepada sub-sub bahagian supaya ianya lebih terperinci dan jelas. Skop permasalahan yang perlu diselesaikan di dalam Latihan Ilmiah II-WXES 3182 ini terbahagi kepada dua, iaitu:-

1.3.1 Skop A : Skop Permasalahan Litar Aritmetik Reja Digit Bertanda.

i. Menghasilkan litar aritmetik pengesan ralat menggunakan penyemak reja dengan sistem nombor digit bertanda, 8-digit.

ii. Mengimplementasikan sistem nombor digit bertanda pada litar aritmetik reja menggunakan moduli 2^P, 2^{P+1} atau 2^{P-1}.

iii. Menggabungkan komponen pendarab 8x8 bit, komponen penambah CLA, komponen penambah modulo m (MSDA), komponen pendarab modulo m (MSDM), komponen penukar dan komponen pembanding menjadi satu litar aritmetik reja digit bertanda.

iv. Memastikan litar darab-tambah dan litar penyemak ralat dapat menjalankan fungsi masing-masing.

v. Memastikan integrasi wujud diantara nombor perduaan, nombor digit bertanda dan nombor reja.

vi. Menghasilkan litar aritmetik reja yang berupaya melaksanakan operasi aritmetik dan semakan ralat pada kadar kelajuan yang tinggi.

vii. Memastikan kelakuan setiap komponen dihurai didalam kod VHDL dengan tepat dan mencapai objektif pembangunan projek.

viii. Memastikan proses kompil, simulasi dan ujian bangku setiap komponen berjaya dilakukan dan bebas ralat.

ix. Memastikan bentuk gelombang yang terhasil adalah tepat dan bebas ralat.

1.3.2 Skop B : Skop Permasalahan Pendarab Aritmetik Reja Digit Bertanda.

i. Membangunkan litar pendaraban dengan menggunakan modulo m SD multiplier.

ii. Memastikan litar darab-tambah dan litar penyemak ralat berupaya untuk melaksanakan operasi pendaraban yang melibatkan nombor perduaan, digit bertanda dan nombor reja.

iii. Mengimplementasikan algoritma pendaraban aritmetik reja digit bertanda kepada operasi pendaraban reja pada litar sebenar.

iv. Melaksanakan operasi penjana darab separa (PPG) dan pokok penambah perduaan digit bertanda pada litar penyemak ralat.

v. Kenalpasti dan menentukan jenis moduli $m=\{2^P, 2^{P+1}$ atau $2^{P-1}\}$ yang sesuai bagi algoritma pendaraban aritmetik reja digit bertanda.

vi. Kenalpasti dan menentukan konsep pendaraban yang sesuai bagi menghasilkan litar pendaraban yang berkelajuan tinggi.

vii. Memastikan perisian aplikasi VHDL yang hendak digunakan boleh memenuhi segala keperluan untuk merekabentuk litar aritmetik reja.

viii. Membangunkan rekabentuk litar pendaraban aritmetik reja digit bertanda yang lebih padat dan menggunakan get-get logik pada kadar yang paling minima.

ix. Mendapatkan masa operasi pendaraban yang laju dan mengurangkan masa lengah semasa proses simulasi dijalankan.

x. Memastikan proses kompil, simulasi dan ujian bangku berjaya dilakukan dan bebas dari sebarang ralat.

xi. Memastikan bentuk gelombang berjaya dihasilkan dan bebas ralat.

xii. Memastikan bentuk gelombang yang terhasil mencapai objektif pembangunan komponen.

1.3.3 Skop C : Skop Permasalahan Pendarab 8 × 8 bit.

i. Membangunkan litar pendarab 8 × 8 bit dengan menggunakan konsep pendaraban langsung.

ii. Memastikan litar darab-tambah berupaya untuk melaksanakan operasi pendaraban yang melibatkan nombor perduaan.

iii. Mengimplementasi algoritma pendaraban nombor perduaan kepada operasi pendaraban perduaan pada litar sebenar.

iv. Mendapatkan masa operasi pendaraban perduaan yang laju dan masa lengah yang kurang semasa proses simulasi dijalankan.

v. Memastikan proses kompil, simulasi dan ujian bangku berjaya dilakukan dan bebas dari sebarang ralat.

vi. Memastikan bentuk gelombang berjaya dihasilkan dan bebas ralat.

vii. Memastikan bentuk gelombang yang terhasil mencapai objektif pembangunan komponen.

1.4 Objektif

Untuk mencapai objektif pembangunan litar aritmetik reja digit bertanda ini, beberapa langkah-langkah penyelesaian telah dikenalpasti untuk menangani masalah yang dinyatakan didalam definasi masalah dan skop projek, antaranya seperti berikut:-

i. Masalah perambatan bawaan pada litar aritmetik reja yang menggunakan perwakilan nombor perduaan boleh diatasi dengan membangunakan litar aritmetik reja yang menggunakan konsep sistem nombor digit bertanda. Diketahui bahawa, perambatan bawaan adalah terhad kepada satu kedudukan semasa proses penambahan nombor digit bertanda. Dengan ini, kadar kelajuan operasi litar aritmetik reja secara keseluruhannya dapat ditingkatkan.

ii. Litar aritmetik reja digit bertanda boleh direkabentuk dengan menggunakan pendekatan metodologi VHDL-Very High Description Language. Bahasa huraian HDL ini, sangat efisyen untuk mengimplementasikan FPGA-Field Programmable Gate Array Architecture ke teknologi ASIC-Application Specific Integrated Circuit. Ini kerana VHDL merupakan bahasa yang dicipta untuk menghuraikan kelakuan bagi sistem dan litar elektronik digital.

iii. Pembangunan Litar aritmetik reja digit bertanda ini boleh direalisasikan dengan menggunakan perisian aplikasi VHDL, iaitu PeakFPGA Design Suite (Accolade). Aplikasi perisian ini dipilih kerana PeakFPGA Design Suite merupakan tool yang menyediakan kombinasi FPGA synthesis yang terbaik dan mempunyai ciri-ciri penggunaan yang mudah. PeakFPGA Design Suite adalah tool perisian yang

sempurna dan suatu sistem yang berkuasa tinggi bagi pembangunan aplikasi FPGA.

iv. Pelaksanaan operasi penjana darab separa (PPG) dan pokok penambah perduaan digit bertanda pada litar darab-tambah dan litar penyemak ralat boleh dilakukan dengan mengimplementasikan fungsi matematik dan menghuraikannya kepada kod VHDL.

v. Melakukan pengubahsuaian pada algoritma pendaraban aritmetik reja digit bertanda dan penjana darab separa (PPG) untuk diimplementasikan kepada bahasa VHDL.

vi. Modulo, $m = 2^{P-1}$ dicadangkan didalam pelaksanaan algoritma pendaraban litar aritmetik digit bertanda. Modulo 2^{P-1} ini dipilih kerana ia dapat menghasilkan rekabentuk litar pendarab aritmetik reja yang lebih padat dan kurang menggunakan get-get logik. Ianya bertujuan untuk mencapai objektif pembangunan cip dan keberkesanan litar, dimana ia dapat menjimatkan penggunaan ruang, kos dan meningkatkan kelajuan.

1.5 Kekangan

Kegagalan sesuatu pembangunan projek mencapai objektifnya boleh juga dikaitkan dengan wujudnya masalah kekangan. Masalah kekangan ini, disebabkan oleh faktor luaran seperti masa, kos, tahap keupayaan aplikasi perisian yang digunakan dan lain-lain lagi. Selain itu, masalah kekangan juga disebabkan oleh tahap keupayaan sesuatu operasi projek adalah terhad untuk mencapai objektif kefungsiannya secara keseluruhannya. Oleh itu, didalam pembangunan litar aritmetik reja digit bertanda ini juga terdapat beberapa masalah kekangan yang dijangka akan berlaku semasa proses pembangunannya diantaranya seperti berikut:-.

i. Keupayaan kapasiti ingatan yang terhad

Perlaksanaan operasi litar aritmetik reja digit bertanda didalam perisian aplikasi VHDL memerlukan ruang ingatan yang besar.. Oleh itu, bilangan bit yang digunakan adalah terhad kepada 8 bit, supaya proses simulasi dapat dilakukan dengan lancar dan bebas ralat.

ii. Kekangan aplikasi perisian VHDL – PeakFPGA Design Suite (Accolade)

Terdapat ralat kod sumber VHDL semasa kompil. Keadaan ini mungkin disebabkan oleh keupayaan library pada perisian aplikasi VHDL yang terhad dan tidak dapat memenuhi sintaks yang digunakan didalam pengekodan litar aritmetik reja digit bertanda. Selain daripada itu berlaku masalah pengemaskinian fail objek semasa kompil dan simulasi dan menyebabkan ralat pada paparan bentuk gelombang .

iii. Kekangan kegagalan saling hubungan komponen

Komponen pendarab modulo m menggunakan penjana darab separa dan penambah CLA untuk melaksanakan operasi pendaraban nombor reja. Oleh itu, ia memerlukan fungsi

penukaran nombor bagi membolehkannya mencapai keselarian panjang input dan ouput dengan komponen penambah CLA. Oleh kerana kegagalan kefungsian komponen penukar, maka komponen pendarab modulo m tidak dapat melaksanakan operasi penambahan hasil darab separa.

iv. Kekangan masa

Untuk membangunkan litar aritmetik reja digit bertanda ini sehingga ke peringkat rekabentuk cip adalah sukar kerana melibatkan banyak proses yang perlu dilakukan dan mengambil masa yang panjang untuk disiapkan. Oleh itu, pembangunan litar aritmetik reja ini dijangka hanya dapat dilakukan sehingga ke proses simulasi.

1.6 Rancangan Perlaksanaan Projek WXES 3181 & WXES 3182

Jangkamasa dan aktiviti-aktiviti bagi pembangunan projek WXES 3181 & WXES 3182 digambarkan didalam jadual di bahagian Apendiks A dan Apendiks B.

BAB 2 : KAJIAN LITERASI

Kajian literasi ini melibatkan beberapa analisis yang dilakukan keatas pembangunan litar aritmetik konvensional yang terdahulu. Kajian literasi ini dilakukan dengan merujuk kepada beberapa jurnal dan buku yang berkaitan dengan konsep nombor reja, nombor digit bertanda dan nombor perduaan. Kajian ini juga meliputi aplikasi perisian sedia ada yang melaksanakan fungsi matematik.

2.1 Analisis Dari Jurnal Dan Buku

i. N.S.Szabo and R.I.Tanaka [1], telah menyatakan bahawa aritmetik reja boleh melakukan operasi tambah, tolak dan darab didalam satu langkah tanpa mengambilkira panjang bagi sesuatu nombor serta tidak memerlukan perantaraan digit bawaan.

ii. N.S.Szabo and R.I.Tanaka dan D.P.Agrawal and T.R.N.Rao [1,2], hasil analisis daripada kedua-dua jurnal ini, mereka telah menyatakan bahawa integer 2_P, 2_{P+1} dan 2_{P-1} biasanya digunakan sebagai moduli bagi sistem nombor reja kerana penambahan modulo 2_P,2_{P+1} atau 2_{P-1} boleh diimplementasikan melalui penambah perduaan P-bit.

iii. F.J.Taylor dan A.Hiasat [3,4], telah mengesyorkan pendarab modulo 2_{P+1}, 2_{P-1}.

Walau bagaimanapun, oleh kerana penambah dan pendarab modulo 2_{P+1}, 2_{P-1} ini direkabentuk berasaskan pada sistem aritmetik perduaan biasa, maka perambatan bawaan akan berlaku semasa penambahan dan menghadkan kelajuan operasi aritmetik dalam modul reja. Untuk mengatasi masalah ini satu konsep baru telah dicadangkan.

iv. Menurut A.Anizienis [5], beliau telah menyatakan didalam jurnalnya bahawa penggunaan perwakilan nombor digit bertanda boleh menghadkan perambatan bawaan kepada satu kedudukan semasa penambahan.

v. Manakala S.Wei and K.Shimizu [6], pula telah mencadangkan satu konsep baru pada aritmetik reja dengan perwakilan reja lewahan yang menggunakan sistem nombor digit bertanda, p-digit asas-2 untuk melaksanakan aritmetik reja berkelajuan tinggi.

vi. Selain itu terdapat 3 penulis jurnal yang memberikan pandangan yang sama dimana mereka telah menyatakan bahawa litar modulo 2_{P+1} dan 2_{P-1} boleh digunakan didalam sistem nombor reja atau semakan reja bagi mendapatkan operasi berkelajuan tinggi [1,3,4].

vii. S.Wei and K.Shimizu [6], telah menganggarkan prestasi yang boleh dicapai oleh litar aritmetik yang menggunakan penyemak ralat reja sistem nombor digit bertanda, seperti didalam jadual 2.1.

Jadual 2.1 : Prestasi litar penyemak ralat

Litar	Get	Masa lengah (ns)
Darab-tambah	14,629	293.68
MSDA	190	5.45
MSDM	1552	28.64
Penukar(64)	1453	25.35
Penukar(32)	669	16.93

2.2 Analisis Aplikasi Perisian – MATLAB

MATLAB-Matrix Laboratory adalah perisian visual dan pengkomputeran berprestasi tinggi yang melaksanakan operasi aritmetik berasaskan pada operasi matrix dan array. Operasi aritmetik matrix ditakrifkan menggunakan peraturan matrix algebra. Manakala operasi array pula dilakukan dari satu elemen ke satu elemen. Namun begitu, aplikasi perisian ini juga mempunyai beberapa kelemahan, diantaranya seperti berikut:-

i. Operasi penambahan dan pendaraban perlu mematuhi beberapa prinsip matrix, dimana;

   i.i. Penambahan/penolakan - hanya ditakrifkan jika dua matrix yang terlibat mempunyai dimensi yang sama.

   i.ii. Pendaraban - hanya berfungsi jika matrix mempunyai nilai dalaman dimensi yang sama.

ii. Operasi aplikasi perisian ini lebih tertumpu kepada konsep sains komputer dan tidak menggunakan konsep matematik keseluruhannya.

iii. Kelajuan dan keupayaan operasi aritmetik aplikasi perisian ini bergantung pada prestasi CPU dan ALU komputer yang digunakan.

iv. Pengguna perlu mempunyai kemahiran didalam bidang sains komputer dan matematik untuk menggunakan aplikasi perisian ini.

v. Jenis nombor yang boleh digunakan untuk melaksanakan operasi aritmetik adalah terhad.

vi. Aplikasi perisian ini tidak boleh melakukan pengesanan dan penyemakan ralat pada hasil operasi aritmetik yang dijalankan.

2.3 Analisis Konsep Pendaraban Langsung

Pada kebiasaannya, kita melakukan operasi pendaraban dengan menggunakan konsep pendaraban langsung yang panjang menggunakan nombor asas sepuluh (decimal).

Cth 1:

				3	1	8	4	×
				6	2	0	7	
			2	2	2	8	8	(3184×7)
							0	(3184×00)
		6	3	6	8	0	0	(3184×200)
1	9	1	0	4	0	0	0	(3184×6000)
1	9	7	6	3	0	8	8	

Untuk mengimplementasikan fungsi ini didalam logik perduaan, langkah utama yang perlu dilakukan adalah memisahkan operasi pendaraban panjang ini kepada bahagian-bahagian unsur. Hasil darab yang terakhir dibina daripada penambahan bagi setiap nombor komponen. Nombor pertama didarabkan dengan setiap komponen pada nombor kedua didalam turutan giliran dan anjakan. Biasanya, didalam konsep pendaraban panjang ini, operasi penjumlahan untuk mendapatkan hasil darab dilakukan hanya sekali, bagi keseluruhan komponen yang telah dikira.

Namun begitu, terdapat pendekatan alternatif yang boleh diiplementasi dimana ianya melibatkan pengiraan pendaraban separa pada setiap peringkat. Pendekatan ini merujuk kepada, perlunya untuk menyimpan dua nilai, iaitu nilai pendaraban separa dan

nilai komponen pendaraban tunggal bagi melakukan operasi pendaraban. Dua nilai ini ditambah bersama untuk mengira pendaraban separa yang baru. Konsep ini memudahkan operasi pendaraban iaitu dengan hanya menambah dua nombor. Pendaraban panjang asas sepuluh masih lagi memerlukan pendaraban bagi mengira setiap nilai komponen.

Cth 2:				3	1	8	4	×
				6	2	0	7	
							0	(darab separa)
			2	2	2	8	8	(3184×7)
			2	2	2	8	8	(darab separa)
							0	(3184×00)
			2	2	2	8	8	(darab separa)
		6	3	6	8	0	0	(3184×200)
		6	5	9	0	8	8	(darab separa)
1	9	1	0	4	0	0	0	(3184×6000)
1	9	7	6	3	0	8	8	

Konsep ini juga boleh diimplen bagi operasi pendaraban dua nombor perduaan. Didalam pendaraban nombor perduaan, terdapat dua nilai yang mungkin bagi menjalankan

operasi darab, iaitu bit 0 dan bit 1, yang mana akan menghasilkan nilai 0 atau nombornya sendiri.

Cth 3:

			1	0	0	1	×
			1	1	0	1	
			1	0	0	1	(1001×1)
						0	(1001×00)
	1	0	0	1	0	0	(1001×100)
1	0	0	1	0	0	0	(1001×1000)
1	1	1	0	1	0	1	

Berikut adalah contoh operasi pendaraban nombor perduaan yang menggunakan pendaraban separa;

Cth 4:			1	0	0	1	×	
			1	1	0	1		
							0	(darab separa)
			1	0	0	1		(1001×1)
			1	0	0	1		(darab separa)
							0	(1001×00)
		1	0	0	1	0	0	(1001×100)
		1	0	1	1	0	1	(darab separa)
	1	0	0	1	0	0	0	(1001×1000)
	1	1	1	0	1	0	1	

2.4 Analisis Konsep Perwakilan Nombor Reja & Konsep Perwakilan Nombor Digit Bertanda

2.4.1 Konsep Sistem Nombor Reja

Penukaran integer biasa kepada perwakilan nombor reja boleh dilakukan dengan menggunakan modulo. Ia menggunakan nombor digit bertanda; P-digit asas-2.

Persamaan penukaran nombor integer kepada nombor reja, [6]:

Pers. 1: $x_m = X - m[X/m]$ x_m = integer reja,

X = integer,

m = modulo; $m = 2^{P-1}$

2.4.2 Konsep Sistem Nombor Digit Bertanda

Melibatkan penukaran nombor integer biasa kepada nombor perduaan digit bertanda asas-2. Nombor perduaan digit bertanda asas-2 hanya melibatkan nombor -1, 0 dan 1, [6].

Fungsi aritmetik digit bertanda asas-2:

Pers. 2: $Z = A \times B + C$ A, B = n-bit nombor perduaan

C, Z = $2n$-bit nombor perduaan

Persamaan darab-tambah dalam bentuk perwakilan digit bertanda nombor reja asas-2

Pers. 3: $Z = \| |A|_m | \times |B|_m | + |C|_m |_m$;

$|A|_m$, $|B|_m$, $|C|_m$, $|Z|_m$ = nombor perduaan p-bit

2.4.3 Algoritma Pendaraban Modulo m

Dengan menggunakan 2 integer iaitu x dan y dalam perwakilan nombor digit bertanda, P-digit asas-2, operasi pendaraban modulo m boleh dilakukan dengan menggunakan konsep penjana darab separa (PPG),[6].

Pers. 4: $$\langle x \times y \rangle_m = \langle \sum_{i=0}^{p-1} \langle y_i 2^i \times (x_{p-1} 2^{p-1} + x_{p-2} 2^{p-2} + \ldots + x_0) \rangle_m \rangle_m$$

$$= \langle \sum_{i=0}^{p-1} ppi \rangle_m$$

Di mana ;

$$ppi = \langle y_i 2^i \times (x_{p-1} 2^{p-1} + x_{p-2} 2^{p-2} + \ldots + x_0) \rangle_m \rangle_m ;$$

$$\langle 2^p \rangle_m = -\mu \ldots\ldots \mu \in \{-1,1\};$$

$$\langle x_{p-1} 2^p \rangle_m = -\mu \times x_{p-1} = x_{-1};$$

$$yi \in \{-1,0,1\};$$

Apabila $\langle 2^p \rangle_m$ digantikan dengan $-\mu$, maka persamaan akan menjadi;

$$ppi = yi \times (x_{p-i-1}, x_{p-i-2}, \ldots, x_0, -\mu x_{p-1}, \ldots, -\mu x_{p-i+1}, -\mu x_{p-i})_{SD}$$

2.4.4 Implementasi Algoritma Modulo m Kepada VHDL

Untuk menentukan perwakilan nombor digit betanda asas-2, x_i ditunjukan seperti didalam jadual 2.2, [6] :-

Jadual 2.2: Perwakilan perduaan bagi RSD asas-2

x_i	x_i (1)	x_i (0)
-1	1	1
0	0	0
1	0	1

x_i (1) = tanda ;

x_i (0) = nilai mutlak (abs) bagi x

Dengan mengubahsuai algoritma 1; maka abs(x_i) ≠ abs(y_i) adalah seperti berikut :-

$$z_i = \begin{cases} -(x_i + y_i) & \text{jika } T = TL \\ x_i + y_i & \text{sebaliknya} \end{cases}$$

dan

$$c_i = \begin{cases} x_i + y_i & \text{jika } T = TL \\ 0 & \text{sebaliknya} \end{cases}$$

dimana

$T = x_i\,(1)$ or $y_i\,(1)$

$TL = x_{i-1}(1)$ or $y_{i-1}\,(1)$

BAB 3: METODOLOGI

3.1 Latar Belakang VHDL

Pada tahun 1981, satu bengkel tentang HDL telah dianjurkan oleh United States Department of Defense (DoD) di Woods Hole, Massachusetts untuk melakukan penyelidikan tentang tool dokumentasi dan rekabentuk piawai bagi program litar bersepadu kelajuan tinggi, VHSIC. Bengkel ini dikendalikan oleh Institute for Defense Analysis (IDA) untuk mengkaji pelbagai kaedah huraian perkakasan, keperluan satu bahasa piawai dan ciri-ciri yang mungkin perlu oleh sesuatu piawai tersebut. Oleh kerana program VHSIC ini dibawah pengawasan United States International Traffic and Arms Regulations (ITAR), maka komponen VHDL bagi program ini pada mulanya tertakluk dibawah pengawasan ITAR.

Dalam tahun 1983, DoD memperkenalkan keperluan bagi piawai bahasa huraian perkakasan VHSIC berdasarkan pada cadangan dari bengkel "Woods Hole". Kontrak untuk pembangunan bahasa VHDL dan persekitarannya serta perisiannya telah diberikan kepada IBM, Texas Instruments dan Intermetrics. Kerja pembangunan VHDL ini telah dimulakan pada tahun 1983. Pada masa tersebut, spesifikasi bahasa ini tidak lagi dibawah pengawasan ITAR, tetapi pengawasan masih lagi diteruskan pada pembangunan perisian kerajaan. VHDL 2.0, telah dikeluarkan hanya selepas 6 bulan projek bermula. Walaubagaimanapun, versi ini hanya membenarkan pernyataan serentak dan tidak berupaya untuk menghuraikan perkakasan dalam perisian berjujukan. Kelemahan ini boleh mengganggu keupayaan bagi bahasa huraian kelakuan tahap tinggi. Bahasa ini telah dipertingkatkan dan kelemahan ini telah diperbaiki apabila VHDL 6.0 dikeluarkan, pada Disember, 1984. Pada tahun yang sama, pembangunan VHDL berasaskan tool juga bermula. Dalam tahun 1985, VHDL dan perisian-perisian yang berkaitan, tidak lagi

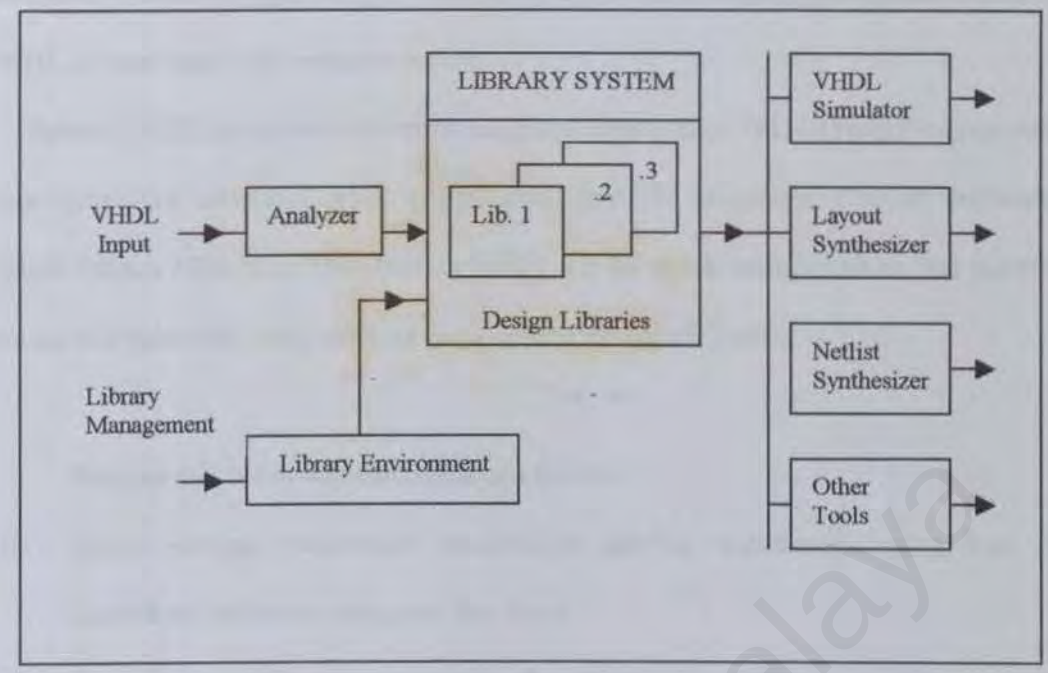
tertakluk dibawah pengawasan ITAR, manakala hakcipta VHDL 7.2 Language Reference Manual (LRM) telah dipindahmilik kepada IEEE untuk pembangunan dan piawaian yang lebih lanjut. Ini telah mendorong kepada pembangunan IEEE 1076/ A VHDL Language Reference Manual (LRM), dimana ia telah dikeluarkan pada Mei, 1987. Setahun kemudian, versi B bagi LRM telah dibangunkan dan diluluskan oleh REVCOM (Ahli Jawatan Kuasa Lembaga piawaian IEEE). VHDL 1076-1987, secara rasminya menjadi piawai HDL IEEE pada Disember, 1987. Usaha untuk mendefinisikan versi baru bagi VHDL bermula pada tahun 1990, oleh sekumpulan sukarelawan yang bekerja dibawah IEEE DASC (Design Automation Standards Committee).

Pada Oktober, 1992, VHDL yang baru dirujuk sebagai VHDL'93 telah disempurnakan dan dikeluarkan. Selepas pengubahsuaian kecil dilakukan, versi baru ini mendapat kelulusan dari ahli kumpulan pengundi VHDL dan menjadi bahasa piawai VHDL yang baru. Piawai VHDL secara rasminya dirujuk sebagai VHDL 1076-1993. Dalam pertengahan tahun 1994,

aktiviti mendapatkan piawai bagi versi ini selesai dilakukan.

3.2 Bahasa VHDL

Pada asalnya VHDL merupakan bahasa HDL yang lebih menekankan pada keserentakan. Bahasa ini menyokong huraian berhierarki bagi perkakasan daripada sistem kepada get malah sehingga ke peringkat suis. VHDL menyokong pada semua peringkat spesifikasi pemasaan dan pengesanan perlanggaran. Seperti yang dijangkakan, VHDL menyediakan binaan bagi konfigurasi dan spesifikasi rekabentuk generik. Design entity VHDL ditakrifkan sebagai pengisytiharan entity dan gabungannya iaitu architecture body. Pengisytiharan entity menentukan antaramukanya dan ia digunakan oleh architecture body pada design entity di paras paling atas hierarki. Architecture body menghuraikan operasi design entity iaitu, melalui penentuan salinghubungannya dengan design entity yang lain, melalui kelakuannya atau gabungan kedua-duanya. Bahasa VHDL ini, menghimpunkan subprogram atau design entity dengan menggunakan packages. Untuk menghuraikan generic bagi design entity , configuration digunakan. VHDL juga menyokong libraries dan mengandungi binaan untuk capaian packages, design entity atau configuration daripada pelbagai libraries. Persekitaran rekabentuk VHDL digambarkan seperti dalam rajah 3.1.



Rajah 3.1 : Persekitaran rekabentuk VHDL

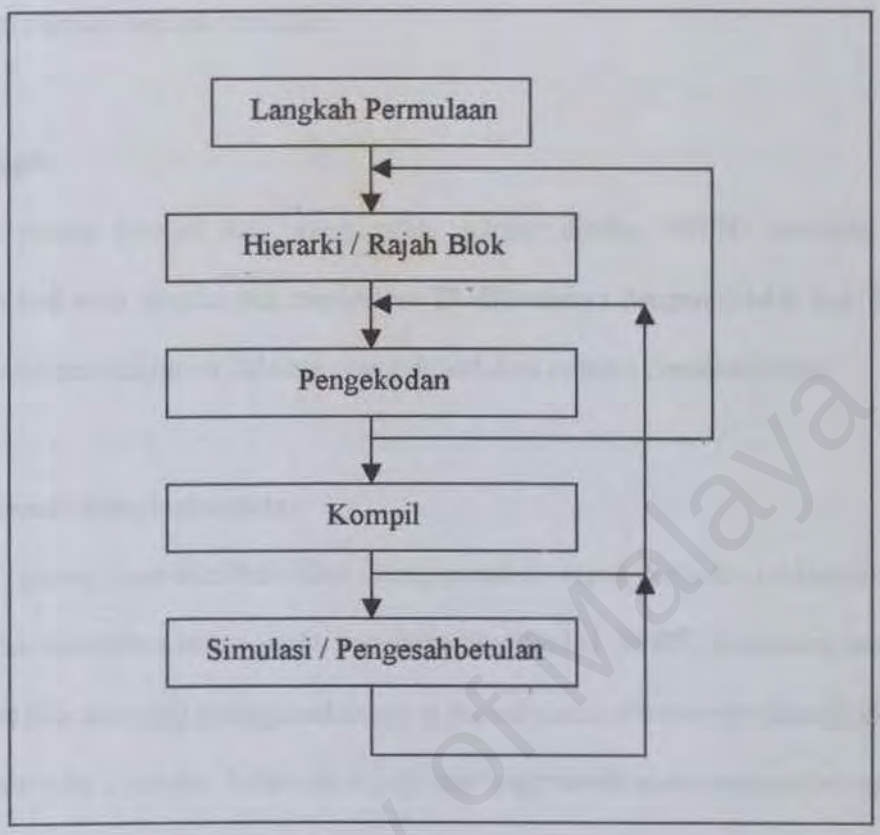
Rajah 3.1 menunjukkan analyzer program menterjemahkan huraian VHDL kepada bentuk perantaraan dan menempatkannya didalam libraries rekabentuk. Analyzer berfungsi sebagai penganalisis leksikal dan penyemak sintaks. Operasi pada libraries rekabentuk, seperti mencipta libraries baru, menghapuskan libraries lama atau menghapuskan packages atau design entity dari libraries dilakukan melalui persekitaran rekabentuk. VHDL berasaskan tools menggunakan bentuk perantaraan daripada libraries rekabentuk. Satu tool pembangunan yang baik ialah VHDL simulator, yang mana ia melakukan simulasi design entity daripada libraries rekabentuk dan menghasilkan laporan simulasi. Manakala, synthesizer pula menguji vector generator, dan tools rekabentuk fizikal merupakan contoh lain bagi VHDL berasaskan tools. VHDL berasaskan tools sedia untuk kebanyakkan jenis platforms dari komputer sehinggalah ker Unix multipengguna. Simulators untuk keseluruhan VHDL IEEE 1076 juga sedia untuk pelbagai jenis

platforms. Selain itu juga, terdapat beberapa program sintesis yang merupakan subset dari VHDL sebagai input dan menjana netlist.

Bahasa VHDL amat berkesan untuk megimplementasikan FPGA (Field Programmable Gate Array) ke teknologi ASIC (Application Specific Integrated Circuit) berbanding dengan bahasa HDL yang lain. Bahasa VHDL dipilih untuk menghuraikan litar aritmetik reja kerana kelebihan yang terdapat pada bahasa ini seperti berikut:-

i. Rekabentuk boleh dipecahkan secara hierarki.
ii. Setiap elemen rekabentuk mempunyai takrifan antaramuka yang baik dan spesifikasi kelakuan yang jelas dan tepat.
iii. Spesifikasi boleh menggunakan algoritma atau struktur perkakasan untuk mentakrifkan operasi elemen-elemen.
iv. Keserentakan, pemasaan dan clock boleh dimodelkan. VHDL mengendalikan struktur litar-berjujukan segerak dan tak segerak.
v. Kod sumber rekabentuk boleh digunakan untuk semua jenis teknologi tanpa perlu mengubahnya.
vi. Operasi logikal dan kelakuan pemasaan bagi rekabentuk boleh disimulasikan

3.3 Aliran Rekabentuk



Rajah 3.2 : Aliran rekabentuk

3.3.1 Hierarki / Rajah Blok

Bermula dengan memberi gambaran tentang pendekatan asas dan pembinaan blok pada tahap rajah blok. VHDL boleh memberi rangka kerja yang baik, iaitu dengan mentakrifkan modul-modul dan antaramukanya dan kemudian memasukkan butir-butir kedalamnya.

3.3.2 Pengekodan

Melibatkan proses menulis kod VHDL bagi modul-modul, antaramuka dan butir-butir didalamnya. Penulisan kod dilakukan dengan menggunakan VHDL text editor, dan

sesetengah editor mempunyai ciri-ciri seperti highlight pada katakunci VHDL dan indenting secara automatik, built-in templates bagi struktur program, pemeriksaan sintaks dalaman dan capaian kepada compiler.

3.3.3 Kompil

Melibatkan proses kompil kod yang telah selesai ditulis. VHDL compiler akan menganalisa kod ralat sintaks dan memeriksa kesesuaiannya dengan modul lain. Ia juga mencipta mencipta maklumat dalaman yang diperlukan semasa proses simulasi.

3.3.4 Simulasi / Pengesahbetulan

Melibatkan proses mentakrifkan dan menggunakan input kepada rekabentuk dan memerhatikan outputnya tanpa perlu membina litar fizikal. VHDL Simulator berupaya mencipta test benches yang menggunakan input secara automatik dan membandingkannya dengan output yang dijangka. Selain itu ia juga berfungsi untuk mengesahkan bahawa litar beroperasi sepertimana yang dijangkakan.

3.4 Bahasa AHPL, CDL, CONLAN, IDL, ISPS, TEGAS, TI-HDL

Pada peringkat awal program VHSIC diperkenalkan, didapati tiada satu pun bahasa HDL sedia ada yang sesuai dan boleh digunakan sebagai tool untuk rekabentuk, pengeluaran dan dokumentasi bagi litar digital dari peringkat IC sehingga sistem yang lengkap. Sebahagian daripada kajian yang dilakukan untuk pembangunan keperluan bahasa VHSIC, adalah aktiviti menganalisa keupayaan, kelemahan dan ciri-ciri lain yang terdapat pada 8 jenis bahasa HDL yang sedia ada pada masa tersebut, diantaranya seperti berikut:-

i. AHPL- A Hardware Programming Language

AHPL adalah bahasa HDL yang berfungsi menghuraikan perkakasan pada peringkat aliran data bagi pengabstrakan. Bahasa ini menggunakan implicit clock untuk menyegerakkan data kepada daftar dan flip-flop, tetapi ia tidak menyokong huraian litar tak segerak. Huraian bahasa ini mengandungi modul keserentakan interaksi dan tidak menyokong hierarki modul. Jenis data didalam AHPL adalah tetap dan terhad kepada bit, bit vector dan bit array. Prosedur dan fungsi hanya boleh dilakukan didalam konteks unit logik bergabungan. Spesifikasi lengah dan had tidak boleh dilakukan dalam AHPL dan umpukan nilai kepada bas dan daftar berlaku pada masa yang sama tanpa adanya masa lengah, kerana ianya disegerakkan dengan implicit clock.

ii. CDL-Computer Design Language

CDL adalah bahasa HDL yang dibangunkan didalam persekitaran akademik yang mempunyai arahan didalam sistem digital. Bahasa ini adalah terhad kepada bahasa aliran data dan tidak menyokong rekabentuk hierarki. Didalam CDL, microstatements digunakan untuk memindahkan data kepada daftar. Microstatements ini menggunakan pernyataan berbentuk if-then-else dan boleh disarangkan.

iii. CONLAN-Consensus Language

Projek CONLAN bermula sebagai percubaan untuk memperkenalkan piawai HDL. Platform ini mengandungi bahasa untuk menghuraikan perkakasan pada pelbagai peringkat pengabstrakan. Semua operasi didalam CONLAN dilaksanakan secara serentak. CONLAN berupaya menghuraikan perkakasan secara hierarki tetapi penggunaan diluar adalah terhad.

iv. IDL-Interactive Description Language

IDL merupakan bahasa IBM dalaman dan kegunaan luarannya adalah terhad. IDL pada asalnya direkabentuk untuk janaan automatik bagi struktur PLA, tetapi kemudiannya telah diperluaskan merangkumi huraian litar berhierarki. Perkakasan didalam HDL boleh dihuraikan dalam struktur berhierarki. Bahasa ini adalah bahasa HDL serentak.

v. ISPS-Instruction Set Processor Specification

ISPS adalah bahasa kelakuan tahap tinggi dan ia direkabentuk untuk mencipta persekitaran bagi perisian rekabentuk yang berasaskan pada perkakasan. Walaupun tujuan bahasa ini disasarkan kepada arkitektur CPU, sistem digital lain boleh dihuraikan dengan mudah didalamnya. Kawalan pemasaan didalam ISPS adalah terhad. Rekabentuk NEXT membenarkan kawalan pemasaan antara pernyataan huraian kelakuan, tetapi ia tidak boleh menentukan pemasaan pada paras get dan maklumat berstruktur.

vi. TEGAS-Test Generation and Simulation

TEGAS merupakan sistem untuk janaan ujian dan simulasi litar digital. Walaupun sesetengah versi lanjutan bahasa ini mempunyai ciri-ciri kelakuan, tetapi bahasa utama (seperti TEGAS atau TDL) adalah berstruktur. Perkakasan digital boleh dihuraikan secara

hierarki didalam bahasa ini. Spesifikasi pemasaan yang lebih terperinci boleh ditentukan didalam TDL.

vii. TI-HDL-Texas Instruments Hardware Description Language

TI-HDL ialah bahasa multi-tahap untuk rekabentuk dan huraian perkakasan. Ia membolehkan spesifikasi berhierarki bagi perkakasan dan menyokong huraian bagi litar logik bergabungan segerak dan tak segerak. Huraian kelakuan didalam TI-HDL adalah secara berjujukan dan ianya menggunakan pernyataan if-then-else, case, for dan binaan untuk kawalan aliran program. Bahasa ini mempunyai jenis data yang tetap dengan tiada syarat atau keperluan untuk penambahan jenis takrifan-pengguna.

3.5 Aplikasi Perisian – PeakFPGA Design Suite

PeakFPGA Design Suite merupakan tool yang menyediakan kombinasi FPGA synthesis yang terbaik dan mempunyai ciri-ciri penggunaan yang mudah. PeakFPGA Design Suite adalah tool perisian yang sempurna dan suatu sistem yang berkuasa tinggi bagi pembangunan aplikasi FPGA. Perisian ini dipilih untuk melaksanakan proses pengekodan, pengkompilan dan simulasi bagi litar aritmetik reja kerana ciri-cirinya, seperti berikut:-

i. Ciri-ciri umum

PeakFPGA Design Suite menerima huraian rekabentuk yang dinyatakan didalam bahasa VHDL (sama ada fail sumber tunggal atau beberapa fail sumber VHDL) dan menganalisis huraian rekabentuk ini supaya ia berfungsi sepertimana yang dikehendaki, dimana ia akan mengoptimakan versi rekabentuk ke format yang sesuai dengan jenis peranti FPGA yang telah ditentukan. Disepanjang proses ini, PeakFPGA Design Suite akan memeriksa rekabentuk untuk memastikan ianya telah disintesis, mengesan dan menjana elemen-elemen tertentu seperti flip-flop, latches dan rangkaian logik gabungan, dan menjalankan pengoptimuman logik yang bersesuaian bagi peranti FPGA tertentu. Synthesis option membolehkan pengguna untuk mengawal paras pengoptimuman yang dijalankan.

ii. Ciri-ciri pengurusan rekabentuk

PeakFPGA Design Suite mempunyai ciri-ciri pengurusan rekabentuk yang sangat berguna untuk proses simulasi dan sintesis. Dengan adanya ciri-ciri ini, ia dapat membantu pengguna untuk mencipta, ubahsuai dan memproses projek VHDL. Hierarchy Browser pula akan menunjukkan paparan terkini bagi struktur rekabentuk. Ciri-ciri ini sangat penting bagi projek yang melibatkan banyak fail sumber VHDL atau melibatkan banyak tahap jika ianya adalah hierarki. Entity Wizard pula berupaya membantu pengguna untuk

mencipta rekabentuk modul baru, melakukan sintesis huraian rekabentuk VHDL iaitu melalui beberapa siri pertanyaan tentang keperluan rekabentuk pengguna dan menjana fail sumber VHDL berasaskan pada keperluan tersebut. Disamping itu,Test Bench Wizard yang terdapat didalam tool ini dapat membantu pengguna mencipta test bench dan menjana modul yang telah disintesis. Dengan ini, secara tak langsung dapat menjimatkan masa

iii. Ciri-ciri simulasi

Dengan adanya VHDL simulator bersepadu didalam tool ini, pengguna boleh mengesahkan rekabentuk FPGA sebelum proses place-and-route FPGA, serta berupaya kenalpasti masalah rekabentuk dengan lebih pantas. Simulator bersepadu PeakFPGA Design Suite mempunyai ciri-ciri seperti berikut;

a. Menyokong IEEE 1076-1987 dan piawai bahasa 1076-1993.

b. Menyokong piawai IEEE 1164 (standard logic), 1076.3 (piawai numerik) dan 1076.4 (piawai VITAL).

c. Paparan bentuk gelombang dengan paparan isyarat boleh pilih, bercirikan zoom dan pan serta kursor pengukuran boleh pilih.

d. Penyahpepijat paras-sumber bersepadu dengan titik putus dan ciri-ciri langkah-tunggal.

e. Pantas, VHDL analyzer yang efisien, penghurai dan penjana kod dengan x86 native compilation.

f. Pemeriksaan secara automatik, termasuklah pemeriksaan tarikh dan masa untuk mempercepatkan pembangunan projek.

g. Konsisten, antaramuka simulasi yang mudah digunakan.

iv. Ciri-ciri sintesis

Algoritma sintesis lanjutan bagi PeakFPGA Design Suite direkabentuk khusus untuk aplikasi dan peranti FPGA dan membolehkan pengguna merekabentuk huraian dengan lebih pantas dan memproses netlists optimized dengan lebih mudah bagi famili peranti FPGA tertentu. Ia mempunyai ciri-ciri seperti berikut;

a. Menyokong IEEE 1076-1987, piawai 1076-1993 serta standard 1164 dan 1076.3.

b. Mempunyai pakej bagi fail sumber yang sesuai dengan synopsys dan produk sintesis yang lain.

c. Menyokong kebanyakan famili FPGA, serta peranti daripada Xilinx, Actel, Altera, Lucent, Lattice dan Quicklogic.

d. Inferens automatik bagi paras-tertinggi, FPGA-primitif khusus

e. Pengkompilasi separa, membenarkan rekabentuk VHDL digabungkan dengan f. bentuk masukan lain dengan lebih mudah.

f. Penyisipan I/O pads dan buffer bagi modul paras-atas.

g. Janaan terus bagi FPGA-netlists khusus didalam format EDIF yang bersesuaian.

h. Pilihan output bahasa CUPL untuk penukaran kepada famili PLD dengan mudah.

i. Atribut sintesis untuk kawalan paras-sumber bagi perisian place-and-route.

j. PeakFPGA Design Suite menyokong 64-bit time

k. Mengandungi context-sensitive VHDL code editor dan tetingkap penyahpepijat paras-sumber bagi masukan rekabentuk yang lebih mudah.

l. Menyokong ciri-ciri text I/O, termasuklah synopsys extensions.

3.6 Aplikasi Perisian VHDL-Xilinx Foundation Series 2.1i

Perisian Foundation Series 2.1i ini merupakan tool yang menyokong rekabentuk bagi semua Xilinx programmable logic device families, termasuklah Spartan/XL, Virtex, XC 4000X, XC 4000XV, XC 3100A/L, XC 5200 FPGA dan XC 9000.

Ciri-ciri aplikasi perisian Xilins Foundation Series 2.1i adalah seperti berikut:

i. Proses simulasi kod sumber VHDL dilakukan baris demi baris.

ii. Boleh mengesan design bottleneck sebelum proses sintesis

iii. VHDL test benches dijana secara automatic bagi kelakuan atau simulasi pemasaan.

iv. Foundation Series test vectors boleh diubah kepada VHDL test bench dan boleh digunakan pada semua peringkat pembangunan rekabentuk.

v. Semua Xilinx libraries adalah binaan dalam dan merupakan pra-kompil bagi mendapatkan prestasi simulasi yang optimum.

vi. Menyediakan Xilinx libraries yang lengkap untuk kegunaan pengguna.

vii. Foundation Project Manager mempunyai antaramuka butang yang membolehkan simulasi rekabentuk VHDL pada paras kelakuan.

viii. Membolehkan simulasi Foundation Series Project pada 3 peringkat;

ix. viii.i Sebelum sintesis (kelakuan)

x. viii.ii Selepas sintesis (kefungsian / paras get)

xi. viii.iii Selepas place & route (pemasaan-RTL, VITAL 95, EDIF, SDF simulation)

xii. Masa pengkompilan yang sangat laju

xiii. Mesra pengguna, antaramuka mudah digunakan

xiv. Sistem pembangunan Xilinx boleh digunakan pada kebanyakan jenis komputer, workstation platform dan sistem pengoperasian.

3.7 Aplikasi Synopsys, Scuba Dan Hand-Made ORCA

Berikut adalah diantara aplikasi-aplikasi perisian VHDL yang dijustifikasikan mengikut rekabentuk:-

3.7.1 Rekabentuk Penambah

i. Synopsys

i.i Ia perlu menggunakan ripple carry adder method yang mempunyai penghalaan yang sangat laju dan rekabentuk yang lebih padat.

3.7.2 Rekabentuk Pendarab

i. Synopsys

i.i Jika saiz operands meningkat, jumlah logic cells yang diperlukan adalah sangat banyak.

i.ii Hanya digunakan untuk signed multipliers.

ii. Scuba generated multipliers

ii.i Menjimatkan ruang

ii.ii Boleh menghasilkan pendarab pelbagai saiz

ii.iii Mengurangkan masa pembangunan

ii.iv Hanya boleh menjana unsigned multipliers sahaja dan tidak boleh digunakan untuk signed multipliers

iii. Hand-made ORCA multiplier

iii.i Mempunyai kelajuan yang tinggi

iii.ii Menggunakan ruang yang kecil

3.7.3 Rekabentuk Pembanding

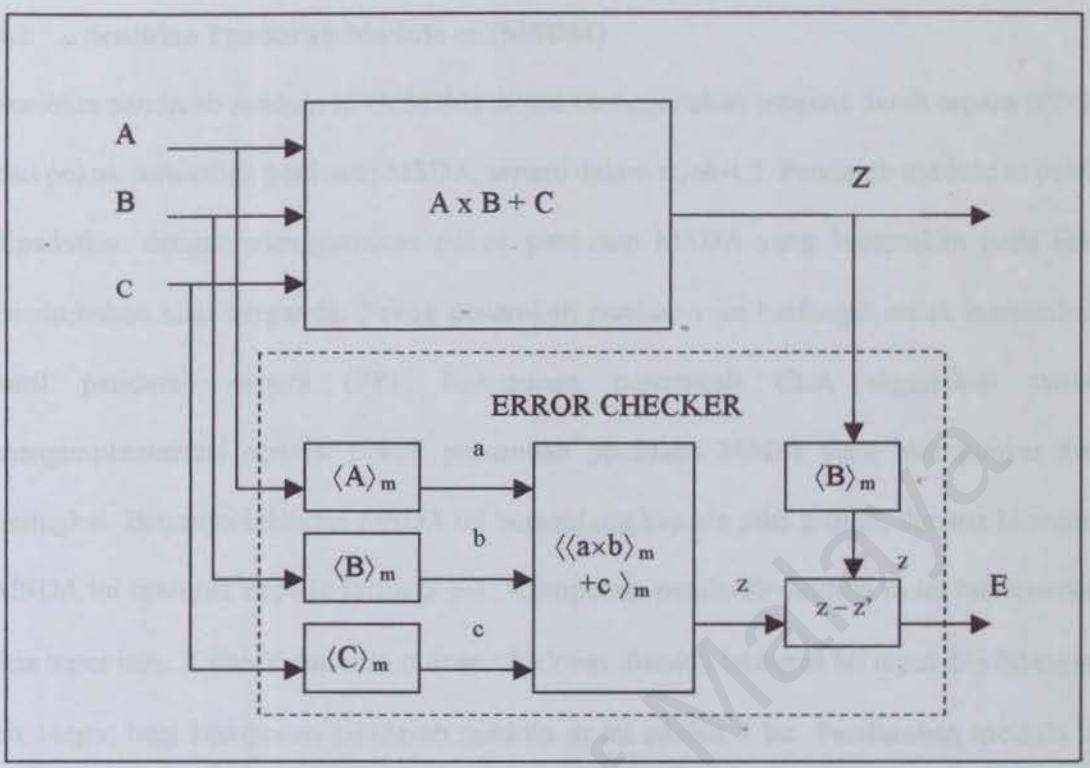
Jadual 3.1 : Ringkasan pembanding hasil dari tiga kaedah

Tools	Keluasan (PFUs)	Kelajuan (MHz)	Lengah Perambatan (ns)
Synopsys	15	70	14.3
SCUBA	19	56	17.8
Hand-made	19	87	11.4

BAB 4 : REKABENTUK LITAR ARITMETIK REJA

4.1 Senibina Litar Aritmetik Reja

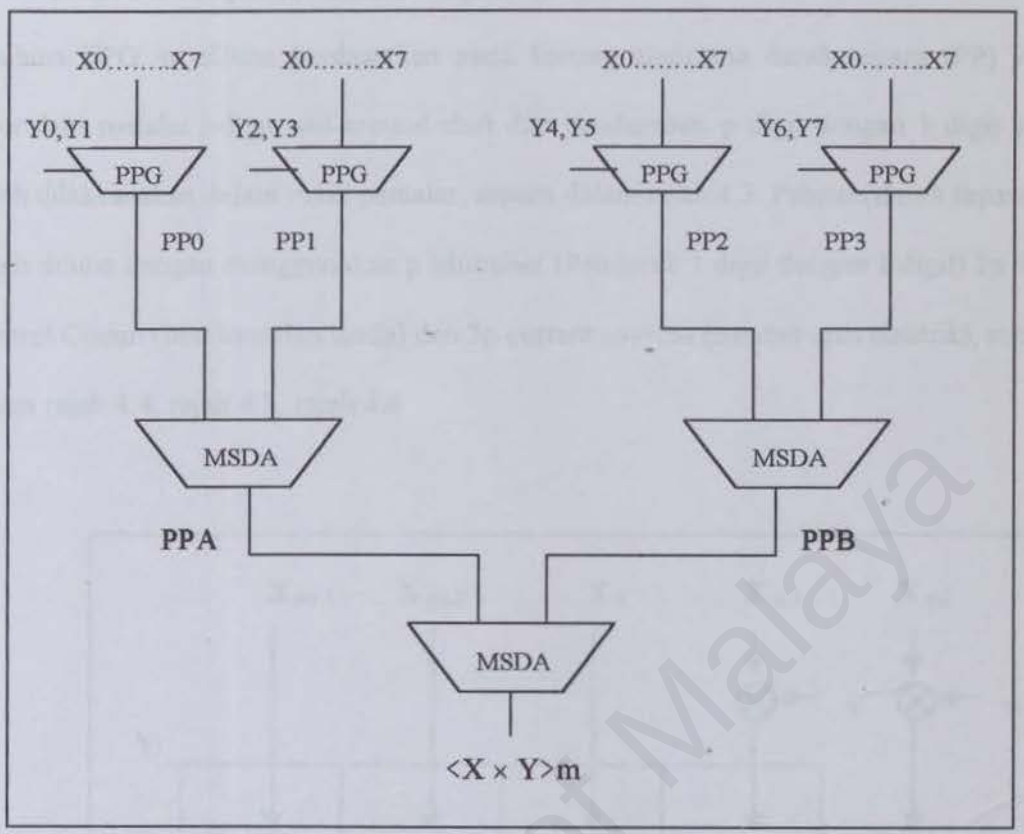
Senibina litar aritmetik reja ini terdiri daripada kombinasi litar darab-tambah dan litar penyemak ralat, seperti dalam rajah 4.1. Litar darab-tambah ini melibatkan operasi pendaraban dan penambahan yang menggunakan perwakilan nombor perduaan digit bertanda. Litar darab-tambah mengandungi komponen pendarab 8×8 bit dan penambah CLA. Manakala litar penyemak ralat pula, mengandungi komponen pendarab modulo m (MSDM), penambah modulo m (MSDA), empat penukar nombor perduaan ke nombor reja digit bertanda dan satu komponen pembanding. Komponen penambah modulo m(bebas-bawaan) menggunakan penambah jenis CLA. Oleh itu, masa penambahan modulo m bebas daripada panjang perkataan operand. Komponen penukar perduaan ke reja melaksanakan operasi menukar A, B, C dan Z dari nombor perduaan n-bit atau 2n-bit kepada $|A|_m$, $|B|_m$, $|C|_m$, dan $|Z|_m$ dengan nombor perduaan p-bit, dimana $n \rangle\rangle p$. Apabila $E \neq 0$, ia menunjukkan berlakunya ralat pengiraan yang dikesan mungkin berlaku dalam litar darab-tambah atau didalam litar penyemak ralat. Penyemak ralat menggunakan perwakilan digit bertanda untuk mengesan ralat pengiraan sama ada $E = 0$ atau $E \neq 0$, dengan mudah dan lebih cepat berbanding dengan perwakilan nombor perduaan. Namun begitu, terdapat beberapa kesukaran yang mungkin dihadapi untuk membina litar aritmetik reja ini. Masalah utama adalah untuk menggabungkan komponen pada litar darab-tambah dan komponen pada litar penyemak ralat serta mewujudkan integrasi antara komponen-komponen ini supaya menjadi satu litar aritmetik pengesan ralat. Masalah kedua, adalah untuk membina litar darab-tambah dan litar penyemak ralat yang berupaya menjalankan operasi darab dan tambah menggunakan perwakilan nombor perduaan digit bertanda dan nombor reja digit bertanda.



Rajah 4.1 : Litar aritmetik dengan penyemak reja

4.2 Senibina Pendarab Modulo m (MSDM)

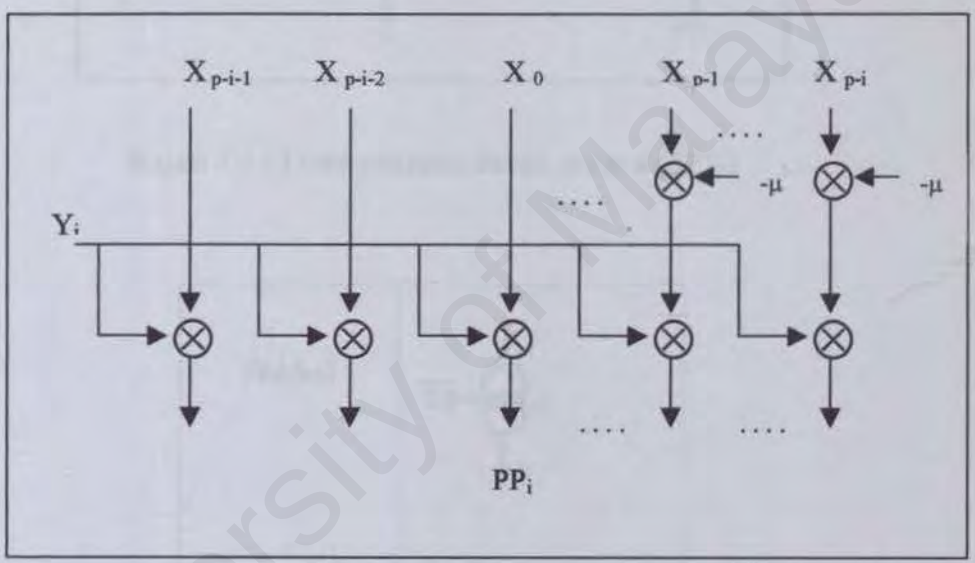
Senibina pendarab modulo m (MSDM) dibina menggunakan penjana darab separa (PPG) dan pokok penambah perduaan MSDA, seperti dalam rajah 4.2. Pendarab modulo m boleh dipadatkan dengan menggunakan pokok perduaan MSDA yang berasaskan pada litar penambahan nilai berganda. Pokok penambah perduaan ini berfungsi untuk menambah hasil pendarab separa (PP). Komponen penambah CLA digunakan untuk mengimplementasi operasi pokok penambah perduaan MSDA yang mempunyai dua peringkat. Biasanya bilangan MSDA ini bergantung kepada nilai p-digit, dimana bilangan MSDA ini merujuk kepada formula p-1. Komponen pendarab modulo m ini mempunyai dua input iaitu X dan Y dan satu output $<X \times Y>m$. Jumlah bilangan bit input dan bilangan bit output bagi komponen pendarab modulo m ini adalah 8 bit Pendaraban modulo m dilaksanakan dalam kadar masa $\log_2 p$. Konsep pendaraban separa dan carta alir bagi operasi pendaraban ditunjukkan seperti didalam bahagian Apendiks I.



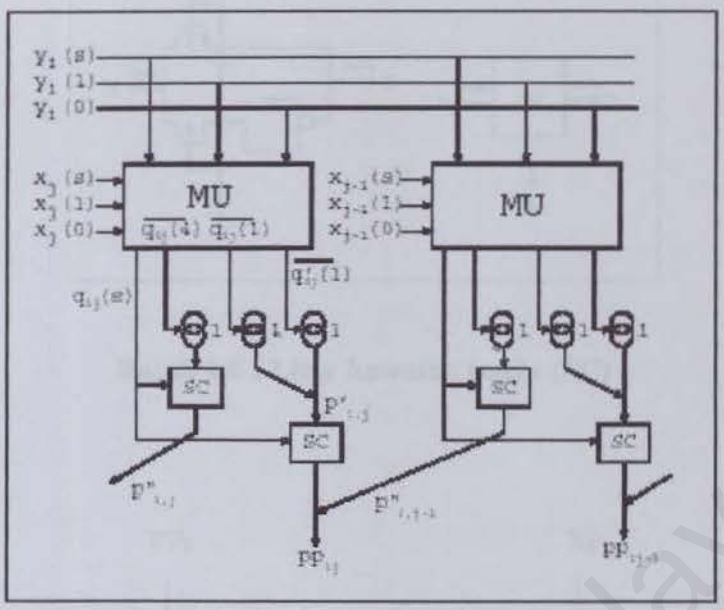
Rajah 4.2 : MSDM dengan pokok perduaan MSDA

4.3 Senibina Penjana Pendarab Separa (PPG)

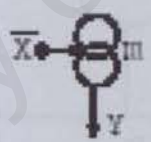
Senibina PPG ini dibina berdasarkan pada konsep algoritma darab separa (PP) yang diperolehi melalui i-digit end-around-shift dan pendaraban p digit dengan 1 digit yang boleh dilaksanakan dalam masa pemalar, seperti dalam rajah 4.3. Penjana darab separa ini boleh dibina dengan menggunakan p Multplier (Pendarab 1 digit dengan 1 digit) 2p Sign Control Circuit (litar kawalan tanda) dan 3p current sources (sumber arus elektrik), seperti dalam rajah 4.4, rajah 4.5, rajah 4.6.



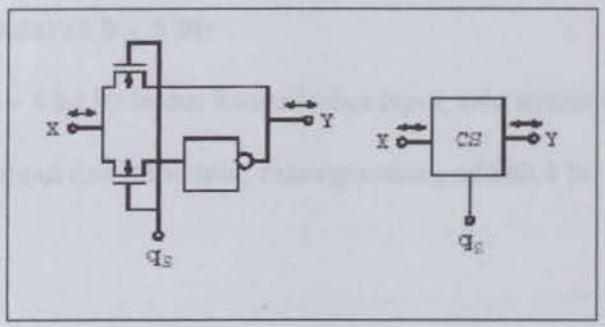
Rajah 4.3 : Litar Penjana Darab Separa (PPG)



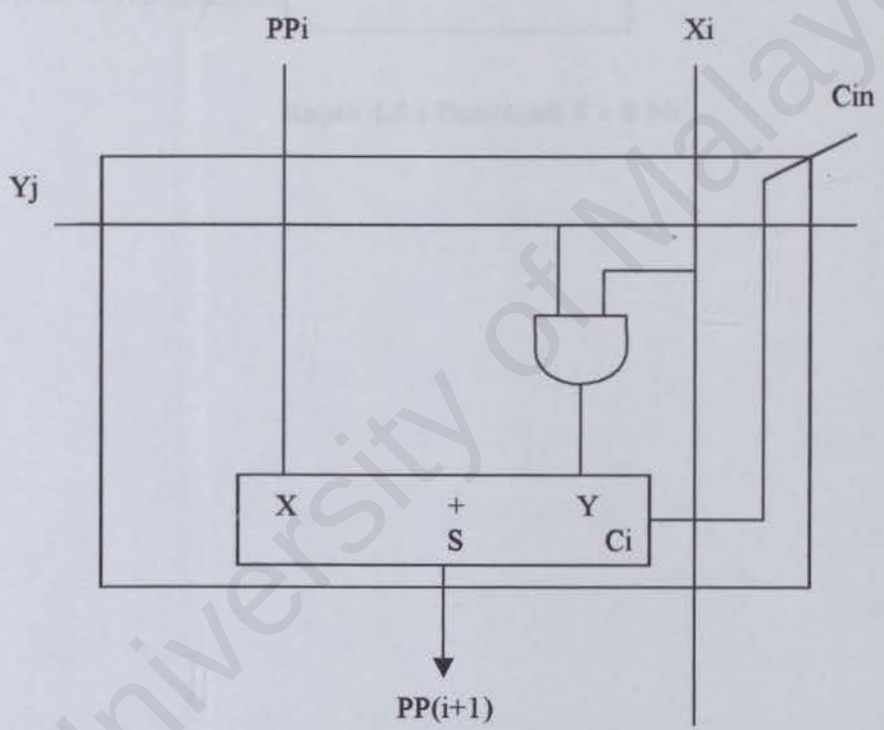
Rajah 4.4 : Litar penjana darab separa (PPG)

Simbol	
Fungsi	Y = 0 if X = "1" Y = m if X = "0"

Rajah 4.5 : Simbol dan fungsi litar mode-arus



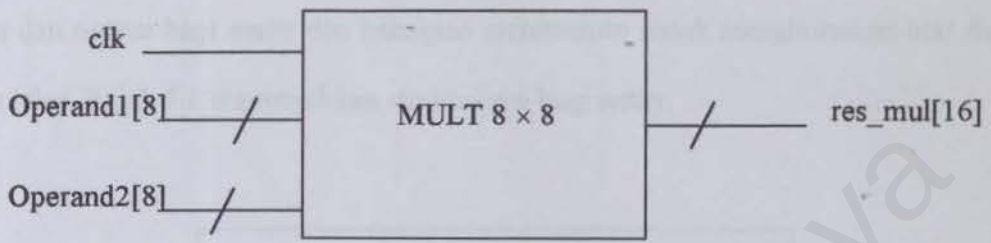
Rajah 4.6 : Litar kawalan tanda (SC)



Rajah 4.7 : Blok binaan asas PP

4.4 Senibina Pendarab 8 × 8 Bit

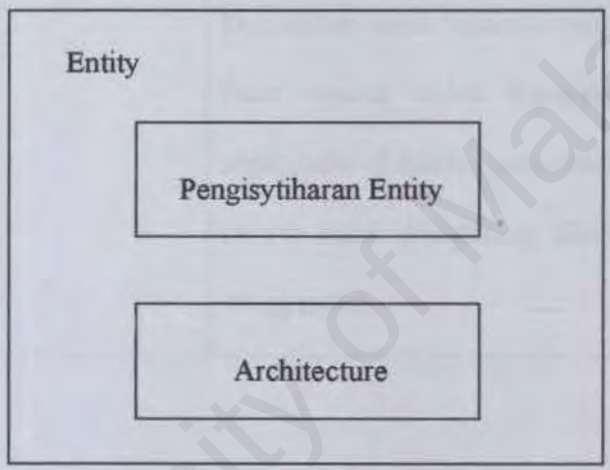
Senibina pendarab 8 × 8 bit ini terdiri daripada dua input, satu isyarat klok dan satu output. Jumlah bilangan bit input dan bit output, masing-masing adalah 8 bit dan 16 bit.



Rajah 4.8 : Pendarab 8 × 8 bit

5.1.1 Struktur Entity VHDL

Litar atau sublitar yang dihuraikan dengan kod VHDL dikenali sebagai entity. Ia mempunyai dua bahagian utama, iaitu pengisytiharan entity untuk menentukan isyarat input dan output bagi entity dan bahagian architecture untuk menghuraikan litar dengan lebih jelas. Rajah 5.1 menunjukkan struktur am bagi entity.



Rajah 5.1: Struktur am bagi entity.

5.1.2 Pengisytiharan Entity

Isyarat input dan output didalam entity ditentukan menggunakan pengisytiharan entity seperti didalam jadual 5.1.

Jadual 5.1 : Mod isyarat bagi port entity

Mod	Fungsi
IN	Digunakan untuk isyarat input kepada entity
OUT	Digunakan untuk isyarat output daripada entity. Nilai isyarat tidak boleh digunakan didalam entity.
INOUT	Digunakan untuk isyarat input dan output entity.
BUFFER	Digunakan untuk isyarat output dari entity. Nilai isyarat boleh digunakan didalam entity,iaitu didalam pernyataan umpukan, isyarat boleh dipaparkan dikiri dan kanan <= operator.

5.2 Piawai IEEE 1164

Piawai IEEE 1164 telah dikeluarkan dipenghujung tahun 1980-an, dan membantu mengatasi kekangan VHDL dan implementasinya. Walaupun VHDL mempunyai banyak jenis data, tetapi ia tidak mengandungi jenis piawai yang membenarkan pelbagai nilai (seperti high-impedance, unknown dan lain-lain) digunakan sebagai perwakilan bagi wayar. Nilai metalogik ini adalah sangat penting bagi mencapai ketepatan simulasi. IEEE 1164 telah membantu mengatasi kekangan VHDL ini dengan mewujudkan sembilan jenis data piawai, seperti berikut:

	Nilai	Huraian
i.	'U'	-uninitialized (tiada nilai-nilai awal)
ii.	X	-unknown (tidak diketahui)
iii.	'0'	-logik 0 (pandu)
iv.	'1'	-logik 1 (pandu)
v.	'Z'	-high impedance (galangan tinggi)
vi.	'L'	-logik 0 (baca)
vii.	'H'	-logik 1 (baca)
viii.	'-'	-don't care (tidak peduli)

Dengan adanya sembilan nilai ini, ia membolehkan pengguna mendapatkan model kelakuan litar digital dengan tepat disepanjang proses simulasi.

5.2.1 Kelebihan Piawai IEEE 1164

Terdapat banyak alasan kukuh mengapa perlunya untuk mengimplen piawai IEEE 1164 bagi semua rekabentuk dan menggunakannya sebagai jenis data piawai bagi semua antaramuka sistem. Bagi proses simulasi, standard logic data type membenarkan pengguna menggunakan nilai selain daripada '0' atau '1' sebagai input dan paparan output. Diantara faktor utama perlunya standard logic data type adalah kerana sifatnya yang mudah alih dimana ia merupakan satu piawai yang boleh diimplemen pada persekitaran simulasi yang berbeza-beza.

5.3 Kod VHDL

Berikut adalah diantara sebahagian daripada kod VHDL yang digunakan untuk penulisan pengekodan litar aritmetik reja.

i. STD_LOGIC dan STD_LOGIC_VECTOR

Jenis STD_LOGIC telah ditambah kepada piawai VHDL didalam IEEE 1164. Ianya lebih fleksibel berbanding dengan jenis BIT. Untuk menggunakan jenis ini, dua pernyataan perlu dimasukkan iaitu;

```
Library ieee;
Use ieee.std_logic_1164.all;
```

Pernyataan ini membolehkan capaian kepada pakej std_logic_1164 yang mentakrifkan jenis std_logic. Nilai yang boleh digunakan bagi objek data std_logic ialah: 0,1,Z,-,L,H,U,X,W. STD_LOGIC_VECTOR mewakili tatasusunan bagi objek STD_LOGIC. Isyarat STD_LOGIC_VECTOR boleh digunakan sebagai nombor perduaan didalam litar aritmetik iaitu dengan memasukkan pernyataan kod;

```
Use ieee.std_logic_signed.all;
```

STD_LOGIC membenarkan isyarat mempunyai banyak sumber, manakala STD_ULOGIC adalah sebaliknya.

ii. Jenis SIGNED dan UNSIGNED

Pakej ini mentakrifkan jenis litar yang hendak digunakan untuk mengimplemen operator aritmetik seperti '+'. SIGNED dan UNSIGNED ialah untuk membenarkan pengguna menunjukkan kod VHDL.

iii. Objek Data

Maklumat diwakilkan didalam kod VHDL sebagai objek data. Terdapat tiga jenis objek data iaitu SIGNAL, CONSTANT dan VARIABLE. Bagi menggambarkan litar logic, objek data yang terpenting sekali adalah isyarat (SIGNAL). Isyarat ini mewakili isyarat logic (wayar) didalam litar.

iii.i Objek Data SIGNAL

Objek data SIGNAL mewakili isyarat logik atau wayar didalam litar. SIGNAL merupakan objek yang digunakan untuk menyambungkan elemen-elemen keserentakan seperti komponen, proses dan umpukan keserentakan iaitu sama seperti mana fungsi wayar yang digunakan untuk menyambungkan komponen pada papan litar atau didalam skematik. Terdapat tiga bahagian dimana isyarat boleh diisytiharkan didalam kod VHDL iaitu didalam pengisytiharan entity, architecture dan didalam pengisytiharan pakej.

iii.ii Objek Data CONSTANT

CONSTANT ialah objek data dimana nilainya tidak boleh diubah. Tidak seperti SIGNAL, CONSTANT tidak mewakili wayar didalam litar. CONSTANT digunakan untuk mencipta lebih banyak huraian rekabentuk boleh baca dan ianya memudahkan pengguna untuk mengubah rekabentuk bila-bila masa.

iii.iii Objek Data VARIABLE

VARIABLE tidak seperti SIGNAL, dimana ia tidak diperlukan untuk mewakili wayar didalam litar. Objek data VARIABLE digunakan untuk memegang hasil keputusan bagi pengiraan dan sebagai pembolehubah indeks didalam gelung (loop).

iv. Jenis INTEGER

Piawai VHDL mentakrifkan jenis INTEGER untuk digunakan bersama operator aritmetik. Isyarat INTEGER mewakili nombor perduaan. Kod tidak memberi nombor bit didalam isyarat secara khusus sepertimana ia lakukan bagi isyarat STD_LOGIC_VECTOR.

v. Operator

VHDL menyediakan operator Boolean, operator aritmetik dan operator hubungan. Ianya dikategorikan mengikut konsep duluan. Bagaimanapun operator daripada kategori yang sama tidak mempunyai konsep duluan keatas satu sama lain. Manakala, didalam operator Boolean pula, tiada konsep duluan diantara setiap operatornya. Klasifikasi Operator VHDL ditunjukkan seperti didalam jadual 5.2.

Jadual 5.2 : Operator VHDL

	Kelas Operator	Operator
Duluan tertinggi	Lain-lain	**,ABS,NOT
	Pendaraban	*, / , MOD, REM
	Tanda	+ , -
	Penambahan	+, -, &
	Hubungan	=, /=, <, <=, >, >=
Duluan terendah	Logikal	AND,OR,NAND,NOR,XOR,XNOR

vi. Pernyataan LOOP

VHDL menyediakan dua jenis pernyataan gelung iaitu pernyataan FOR-LOOP dan pernyataan WHILE-LOOP. Pernyataan ini digunakan untuk pengulangan satu atau lebih pernyataan umpukan berjujukan.

vii. Pernyataan PROCESS

Apabila nilai bagi isyarat didalam senarai sensitive berubah, PROCESS menjadi aktif. Setiap kali ianya aktif, maka pernyataan didalam PROCESS tersebut dinilai dalam turutan berjujukan. Sebarang umpukan isyarat yang diletakkan didalam PROCESS hanya berfungsi selepas semua pernyataan didalam PROCESS tersebut telah dinilai. PROCESS menggambarkan litar logic dan menterjemahkannya kepada persamaan logic. Konsep pernyataan PROCESS ini memberi pemahaman kepada pengguna tentang semantik bagi kod didalam PROCESS.

viii. Penggunaan Component

Entity VHDL yang ditakrifkan didalam satu fail kod sumber boleh digunakan sebagai sublitar didalam fail kod sumber lain. Didalam istilah bahasa VHDL, sublitar dikenali sebagai component. Sublitar mestilah diisytiharkan menggunakan pengisytiharan component. Pernyataan ini mengkhususkan nama bagi sublitar dan memberikan nama bagi port input dan outputnya. Pengisytiharan component boleh dihuraikan didalam bahagian pengisytiharan bagi architecture atau didalam pengisytiharan pakej. Apabila pengisytiharan component diberikan, maka component serta-merta menjadi sublitar.

5.4 Penulisan Kod VHDL – Pendarab Modulo m (MSDM)

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

entity msdmly is

port ( x,y: in std_logic_vector(7 downto 0);
     mly: out std_logic_vector(7 downto 0)
);

end msdmly;

architecture rtl of msdmly is

component CLA_adder

port(

a_in    : in std_logic_vector(7 downto 0);
b_in    : in std_logic_vector(7 downto 0);
c_out   : out std_logic_vector(7 downto 0)

);

end component;

constant h: std_logic_vector(1 downto 0):="01";
subtype word is std_logic_vector(7 downto 0);
type prod is array(0 to 3) of word;

signal pp:prod;
signal ss:prod;
begin


ppg:process(x,y)
variable t: word;

begin

for i in 0 to 3 loop

    t(7 downto 2*i):=x(2*(4-i)-1 downto 0);
```

```
    for j in 0 to i-1 loop
        t(2*j):=x(2*(4-i+j)) and h(0);

        t(2*j+1):=(x(2*(4-i+j)+1) xor h(1)) and t(2*j);

    end loop;
        for j in 0 to 3 loop
      t(2*j):=t(2*j) and y(2*i);
      t(2*j+1):=t(2*j+1) and y(2*i-1) and t(2*j);

        end loop;

        pp(i) <= t;

    end loop;

end process;

mu1: CLA_adder port map (pp(0),pp(1),ss(0));
mu2: CLA_adder port map (pp(2),pp(3),ss(1));
mu3: CLA_adder port map (ss(0),ss(1),mly);
end rtl;
```

5.5 Penulisan Kod VHDL – Ujian Bangku Pendarab Modulo m (MSDM)

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

entity TB_msdmly is
end TB_msdmly;

architecture beh of TB_msdmly is

component msdmly

port (x,y     : in std_logic_vector(7 downto 0);
      mly     : out std_logic_vector(7 downto 0)
   );

end component;

constant period: time := 10 ns;

signal w_x      : std_logic_vector(7 downto 0);
signal w_y      : std_logic_vector(7 downto 0);
signal w_mly    : std_logic_vector(7 downto 0);

begin

dut: msdmly

port map (
            x       =>  w_x,
            y       => w_y,
            mly     => w_mly);

stimuli:process
begin

   w_x <= (others => '0');
   w_y <= (others => '0');

wait for period;

   w_x<= "00000001";
   w_y<= "00000011";

wait for period;
wait;
end process stimuli;
end beh;
```

5.6 Penulisan Kod VHDL – Penambah CLA (MSDA)

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

entity  CLA_adder is

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

end CLA_adder;

architecture fast_carry_behavior OF CLA_adder is

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

BEGIN
sum <= a_in XOR b_in;

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

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

-- Membolehkan penambah untuk 'look ahead'
-- menentukan jika isyarat pembawa (carry signals) diperlukan atau tidak

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

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

END PROCESS;

-- Assign final values to c_out signal

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

END fast_carry_behavior;
```

5.7 Penulisan Kod VHDL – Pendarab 8 x 8 Bit

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

entity multiplier is
            port(
    clk         : in    std_logic  ;
    operand1    : in    std_logic_vector (7 downto 0);
    operand2    : in    std_logic_vector (7 downto 0);
    res_mul     : out   std_logic_vector (15 downto 0)
                );
end multiplier ;

architecture multiplication of multiplier is

        signal res_mul2 : std_logic_vector(15 downto 0);
begin

    process (clk)

        begin

            if clk'event and clk = '1' then

                    res_mul2    <= operand1 * operand2;
                    res_mul     <= res_mul2;

        end if;
   end process;
end multiplication;
```

5.8 Penulisan Kod VHDL – Ujian Bangku Pendarab 8 × 8 Bit

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

entity TB_multiplier is
end TB_multiplier;

architecture behave of TB_multiplier is
component multiplier

  port (

        clk          : in std_logic;
        operand1     : in std_logic_vector (7 downto 0);
        operand2     : in std_logic_vector (7 downto 0);
        res_mul      : out std_logic_vector (15 downto 0)

         );
end component;

constant period: time := 100 ns;

signal clk           : std_logic  ;
signal w_operand1    : std_logic_vector (7 downto 0) ;
signal w_operand2    : std_logic_vector (7 downto 0) ;
signal w_res_mul     : std_logic_vector (15 downto 0);
signal clck          : natural := 0;

begin
  dut: multiplier port map (

            clk          => clk,
            operand1     => w_operand1,
            operand2     => w_operand2,
            res_mul      => w_res_mul
  );

  clock: process

    begin
            clck <= clck + 1;
            clk  <= '1';
            wait for 20 ns;
            clk <= '0';
            wait for 20 ns;
   end process;
  stimuli: process
```

```
  begin

        w_operand1 <= "00100110";
        w_operand2 <= "01010101";
        wait for 40 ns;

        w_operand1 <= "01101011";
        w_operand2 <= "00010011";
        wait for 40 ns;

      wait;

   end process stimuli;

end behave;
```

5.9 Penulisan Kod VHDL – Penambah CLA 16 Bit

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.all;
entity  CLA_adder is

PORT (a_in, b_in    : IN std_logic_vector(15 downto 0);
      c_out         : OUT std_logic_vector(15 downto 0));

end CLA_adder;


architecture fast_carry_behavior OF CLA_adder is

SIGNAL  sum         :  STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL  g           :  STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL  p           :  STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL  c_internal  :  STD_LOGIC_VECTOR(15 DOWNTO 1);
SIGNAL  carry_in    :  STD_LOGIC;

BEGIN
sum <= a_in XOR b_in;

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

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

-- Membolehkan penambah untuk 'look ahead'
-- menentukan jika isyarat pembawa (carry signals) diperlukan atau tidak

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

   FOR i IN 1 TO 14 LOOP
     c_internal(i+1) <= g(i) OR (p(i) AND c_internal(i));
   END LOOP;

END PROCESS;

-- Assign final values to c_out signal

c_out(0) <= sum(0) XOR carry_in;
c_out(15 DOWNTO 1) <= sum(15 DOWNTO 1) XOR c_internal(15 DOWNTO 1);

END fast_carry_behavior;
```

5.10 Penulisan Kod VHDL – Darab-Tambah

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.all;
use work.multiplier;
use work.CLA_adder;



entity prod_sum is
  port (
       A1_in : in std_logic_vector(7 downto 0);
       B1_in : in std_logic_vector(7 downto 0);
       C1_in : in std_logic_vector(15 downto 0);
       E_out : out std_logic_vector(15 downto 0)
      );

end prod_sum;

architecture behavior of prod_sum is


    component multiplier
    port (
       clk          : in std_logic;
         operand1   : in std_logic_vector(7 downto 0);
         operand2   : in std_logic_vector(7 downto 0);
         res_mul    : out std_logic_vector(15 downto 0)
      );
    end component;


  component CLA_adder
    port (
            a_in     : in std_logic_vector(15 downto 0);
            b_in     : in std_logic_vector(15 downto 0);
            c_out    : out std_logic_vector(15 downto 0)
     );

    end component;

  signal result: std_logic_vector(15 downto 0);
  signal clk: std_logic;
```

```
begin

 U1 :multiplier
    port map (
                                clk=>clk,
                                operand1=>A1_in,
                                operand2=>B1_in,
                                res_mul=>result

        );

 U2 :CLA_adder
    port map (  a_in => result,
                b_in => C1_in,
                c_out=> E_out

            );

 end behavior;
```

5.11 Penulisan Kod VHDL – Ujian Bangku Darab-Tambah

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.all;
use work.multiplier;
use work.CLA_adder;

entity TB_prod_sum is
end TB_prod_sum;

architecture BEH of TB_prod_sum is

  component prod_sum
    port(   A1_in  : in std_logic_vector (7 downto 0);
            B1_in  : in std_logic_vector(7 downto 0);
            C1_in  : in std_logic_vector (15 downto 0);
            E_out  : out std_logic_vector (15 downto 0) );

  end component;


  constant period  : time := 100 ns;
  signal W_A1_IN  : std_logic_vector (7 downto 0);
  signal W_B1_IN  : std_logic_vector (7 downto 0);
  signal W_C1_IN  : std_logic_vector (15 downto 0);
  signal W_E_OUT : std_logic_vector (15 downto 0);

begin

  DUT : prod_sum
    port map(
          A1_in => W_A1_IN,
          B1_in => W_B1_IN,
          C1_in => W_C1_IN,
          E_out => W_E_OUT );


  STIMULI : process

begin
        W_A1_in       <= (others => '0');
        W_B1_in       <= (others => '0');
            wait for 40 ns;

        W_C1_in       <= (others => '0');
            wait for 20 ns;
```

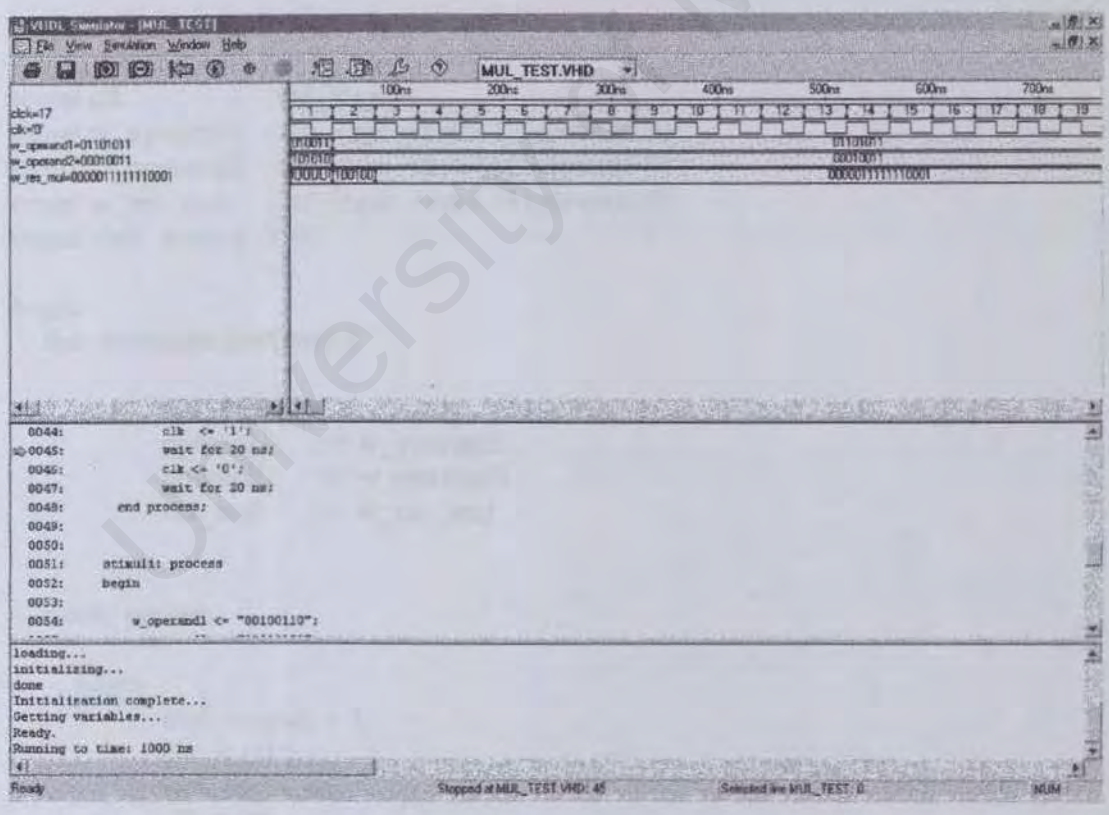
```
        W_A1_in      <= "00000011";
        W_B1_in      <= "00000101";
            wait for 40 ns;

        W_C1_in      <= "0000000000000011";
            wait for 20 ns;

            wait;

    end process STIMULI;

end BEH;
```

BAB 6 : PENGUJIAN LITAR ARITMETIK REJA

6.1 Penulisan Kod VHDL-Ujian Bangku Pendarab 8 × 8 Bit

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

entity TB_multiplier is
end TB_multiplier;

architecture behave of TB_multiplier is
component multiplier

  port (

        clk         : in std_logic;
        operand1    : in std_logic_vector (7 downto 0);
        operand2    : in std_logic_vector (7 downto 0);
        res_mul     : out std_logic_vector (15 downto 0)

         );
end component;

constant period: time := 100 ns;

signal clk            : std_logic ;
signal w_operand1     : std_logic_vector (7 downto 0) ;
signal w_operand2     : std_logic_vector (7 downto 0) ;
signal w_res_mul      : std_logic_vector (15 downto 0);
signal clck : natural := 0;

begin
  dut: multiplier port map (

            clk => clk,
            operand1    => w_operand1,
            operand2    => w_operand2,
            res_mul     => w_res_mul
  );

  clock: process

     begin
            clck <= clck + 1;
            clk  <= '1';
            wait for 20 ns;
```

```
            clk <= '0';
            wait for 20 ns;

    end process;
    stimuli: process
    begin

            w_operand1 <= "00100110";
            w_operand2 <= "01010101";
            wait for 40 ns;

            w_operand1 <= "01101011";
            w_operand2 <= "00010011";
            wait for 40 ns;

        wait;

    end process stimuli;

end behave;
```

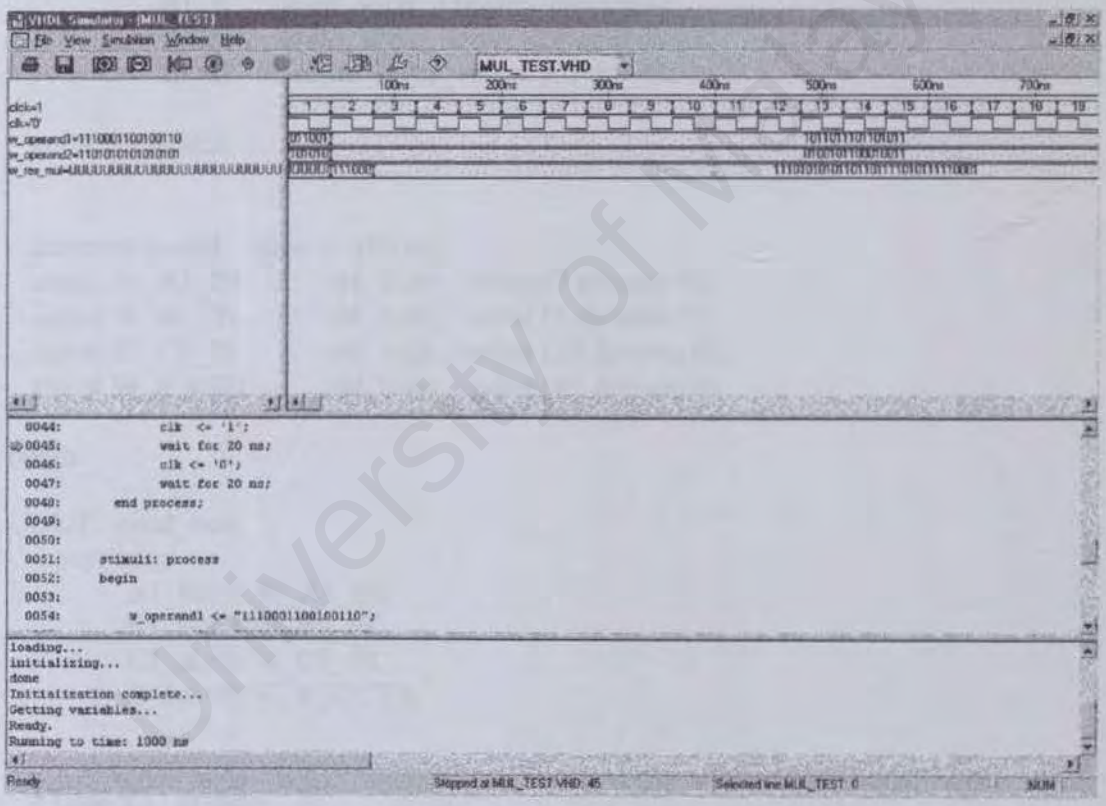


Rajah 6.1 : Bentuk gelombang pendarab 8 Bit

6.2 Penulisan Kod VHDL-Ujian Bangku Pendarab 16 × 16 Bit

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

entity TB_multiplier is
end TB_multiplier;

architecture behave of TB_multiplier is
component multiplier

  port (

        clk          : in std_logic;
        operand1     : in std_logic_vector (15 downto 0);
        operand2     : in std_logic_vector (15 downto 0);
        res_mul      : out std_logic_vector (31 downto 0)

         );
end component;

constant period: time := 100 ns;

signal clk            : std_logic  ;
signal w_operand1     : std_logic_vector (15 downto 0) ;
signal w_operand2     : std_logic_vector (15 downto 0) ;
signal w_res_mul      : std_logic_vector (31 downto 0);
signal clck : natural := 0;

begin
  dut: multiplier port map (

            clk => clk,
            operand1    => w_operand1,
            operand2    => w_operand2,
            res_mul     => w_res_mul
  );

  clock: process

    begin
            clck <= clck + 1;
            clk  <= '1';
            wait for 20 ns;
            clk <= '0';
            wait for 20 ns;
  end process;
  stimuli: process
```

```
begin

        w_operand1 <= "1110001100100110";
        w_operand2 <= "1101010101010101";
        wait for 40 ns;

        w_operand1 <= "1011011101101011";
        w_operand2 <= "0100101100010011";
        wait for 40 ns;

      wait;

    end process stimuli;

end behave;
```

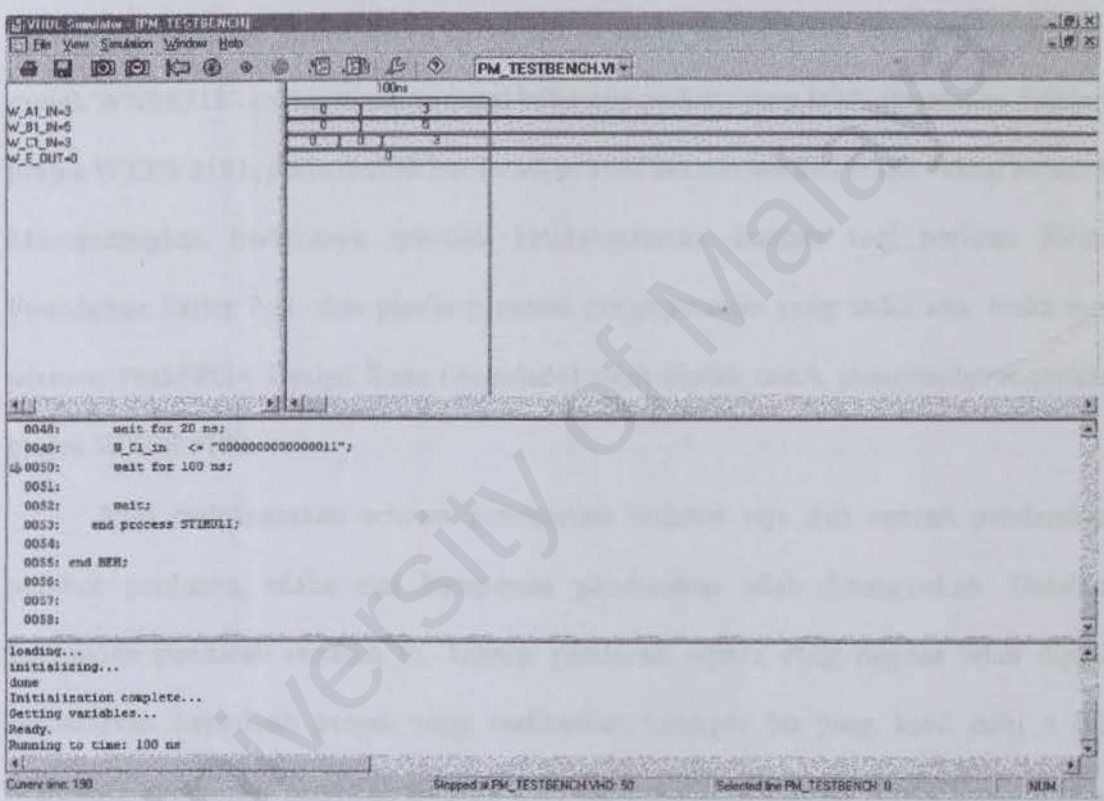


Rajah 6.2 : Bentuk gelombang pendarab 16 Bit

6.3 Penulisan Kod VHDL-Ujian Bangku Darab-Tambah

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.all;
use work.multiplier;
use work.CLA_adder;

entity TB_prod_sum is
end TB_prod_sum;

architecture BEH of TB_prod_sum is

  component prod_sum
    port(    A1_in   : in std_logic_vector (7 downto 0);
             B1_in   : in std_logic_vector(7 downto 0);
             C1_in   : in std_logic_vector (15 downto 0);
             E_out   : out std_logic_vector (15 downto 0) );

  end component;


  constant period  : time := 100 ns;
  signal W_A1_IN       : std_logic_vector (7 downto 0);
  signal W_B1_IN       : std_logic_vector (7 downto 0);
  signal W_C1_IN       : std_logic_vector (15 downto 0);
  signal W_E_OUT       : std_logic_vector (15 downto 0);

begin

  DUT : prod_sum
    port map(
          A1_in => W_A1_IN,
          B1_in => W_B1_IN,
          C1_in => W_C1_IN,
          E_out => W_E_OUT );


  STIMULI : process
  Begin
        W_A1_in        <= (others => '0');
        W_B1_in        <= (others => '0');
            wait for 20 ns;

        W_C1_in        <= (others => '0');
            wait for 100 ns;
```

```
        W_A1_in      <= "00000011";
        W_B1_in      <= "00000101";
            wait for 20 ns;

        W_C1_in      <= "0000000000000011";
            wait for 100 ns;

            wait;
    end process STIMULI;

end BEH;
```



Rajah 6.3 : Bentuk gelombang darab-tambah

BAB 7 : PERBINCANGAN

Seperti yang diketahui, pembangunan projek WXES 3182 ini merupakan kesinambungan daripada projek WXES 3181. Sebagai memenuhi keperluan yang diminta oleh moderator, iaitu En.Zaidi, saya dan 2 orang ahli kumpulan telah menyiapkan jurnal berkaitan sistem nombor reja dan litar aritmetik reja. Jurnal ini mengandungi maklumat dan konsep-konsep yang berkaitan dengan pembangungan projek WXES 3181&3182. Setelah mengadakan perbincangan bersama moderator dan penyelia bagi membangunkan projek WXES3182 maka, maka terdapat beberapa perkara yang telah dinyatakan didalam projek WXES 3181, perlu diubah berdasarkan keadaan semasa dan faktor-faktor tertentu. Memandangkan berlakunya masalah ketidakserasian diantara tool perisian Xilinx Foundation Series 2.1i dan platform sistem pengoperasian yang sedia ada, maka tool perisian PeakFPGA Design Suite (Accolade) telah dipilih untuk mengimplementasikan projek WXES 3182.

Bagi melaksanakan operasi pendaraban nombor reja dan operasi pendaraban nombor perduaan, maka dua komponen pendaraban telah dibangunkan. Didalam komponen pendarab modulo m, konsep pendarab separa yang ringkas telah dipilih berdasarkan keperluan projek yang melibatkan bilangan bit yang kecil iaitu 8 bit. Manakala komponen pendarab 8×8 bit pula menggunakan konsep pendaraban nombor perduaan secara langsung. Untuk menggabungkan komponen-komponen bagi litar darab-tambah dan litar penyemak ralat, dan mencapai fungsi komponen-komponen tertentu, maka kaedah component dan port map digunakan. Terdapat beberapa kekangan yang dihadapi semasa membangunkan komponen pendarab modulo m, dimana proses kompil dan simulasi tidak dapat dilakukan. Keadaan ini disebabkan oleh kegagalan untuk melaksanakan operasi penambahan hasil darab separa, dimana komponen penukar gagal

untuk berfungsi dan secara tak langsung menyebabkan berlakunya masalah ketidak selarian panjang input dan output diantara komponen pendarab modulo m dan komponen penambah CLA. Bagi komponen pendarab 8×8 bit pula, terdapat sedikit masa lengah semasa proses simulasi dijalankan. Oleh kerana terdapat beberapa ralat pada komponen-komponen tertentu, maka pembangunan Litar Aritmetik Reja tidak dapat mencapai beberapa objektif tertentu. Pada mulanya Penyelia telah menetapkan masa dan tarikh untuk membentangkan projek WXES 3182 ini, iaitu pada pukul 2.00 petang, 5 Januari 2002. Tetapi disebabkan oleh kesulitan yang dihadapi oleh penyelia dan moderator , maka sesi VIVA telah pinda ke pukul 9.00 pagi pada tarikh yang sama. En. Zaidi telah mengarahkan saya dan 2 orang ahli kumpulan yang lain melakukan pembentangan projek WXES 3182 ini di lokasi yang sama. Seperti mana yang diketahui, pembentangan dan penilaian projek WXES 3182 dilakukan secara individu.

Kami telah membentangkan projek Litar Aritmetik Reja ini mengikut 3 komponen yang telah ditugaskan pada setiap ahli kumpulan projek WXES 3181&3182, dimana Wan Nurul Rukiah membentangkan komponen penambah CLA, Fadzlin Noor Arbain membentangkan komponen penukar dan komponen pembanding, dan saya sendiri membentangkan komponen pendarab modulo m dan komponen pendarab 8×8 bit.. Semasa sesi VIVA ini dijalankan, kami telah berbincang dengan moderator dan penyelia, dan bertukar-tukar idea antara satu sama lain. Moderator dan penyelia juga mengajukan soalan-soalan berkaitan untuk mengetahui dengan lebih jelas lagi tentang projek kami. Moderator telah memberi beberapa cadangan untuk mengatasi masalah yang kami hadapi didalam membangunkan projek WXES 3182 ini. Antaranya mengusulkan kepada kami supaya menulis kod VHDL dengan cara yang lebih efisyen bagi mengimplem komponen-komponen dengan lebih berkesan. Sesi VIVA tamat pada pukul 10.30 pagi.

BAB 8 : KESIMPULAN

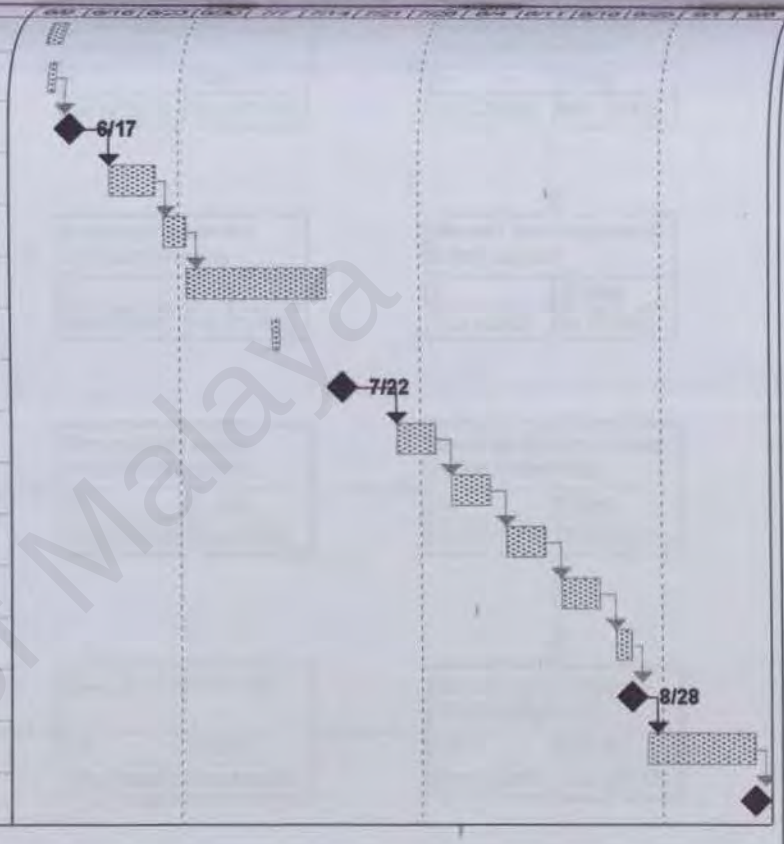
Projek WXES 3182 ini mula dibangunkan selepas pembangunan projek WXES 3181 berjaya dilakukan dan mencapai persetujuan daripada penyelia dan moderator. Seperti yang kita ketahui, maklumat dan konsep bagi Litar Aritmetik Reja telah dihuraikan dengan jelas dan terperinci di beberapa bahagian didalam laporan ini. Litar Aritmetik Reja ini merupakan salah satu daripada pendekatan baru bagi melaksanakan operasi aritmetik, dimana ia menggunakan konsep yang melibatkan perwakilan nombor perduaan digit bertanda dan nombor reja digit bertanda bagi menjalankan operasi aritmetik darab dan tambah. Hasil daripada rujukan yang telah dibuat keatas beberapa jurnal dan buku, didapati tujuan utama untuk membangunkan litar ini adalah untuk mengatasi kelemahan yang terdapat didalam litar aritmetik konvensional yang menggunakan perwakilan nombor perduaan biasa, dimana berlakunya perambatan bawaan dan operasi litar yang tidak efisien.

Selain daripada itu, litar ini juga boleh mengatasi keupayaan pada sistem perisian aritmetik sedia ada dimana litar ini berupaya untuk mengesan ralat pengiraan aritmetik dengan pantas. Hasil daripada sesi VIVA WXES 3182, beberapa idea baru telah diusulkan yang mana ia boleh meningkatkan lagi keberkesanan dan tahap keupayaan litar aritmetik reja ini. Walaupun pembangunan Litar Aritmetik Reja ini tidak dapat mencapai beberapa objektif tertentu tetapi pengalaman dan ilmu yang saya perolehi semasa membangunkan projek WXES 3181 & WXES 3182 amat bernilai kerana secara tak langsung kursus ini memberi pendedahan kepada saya didalam melaksanakan projek besar mengikut jangkamasa yang ditetapkan. Apa yang penting bagi saya adalah ilmu yang diperolehi daripada kajian yang dilakukan dan bagaimana untuk menggunakannya dengan sebaik mungkin pada masa depan kelak.

APENDIKS A

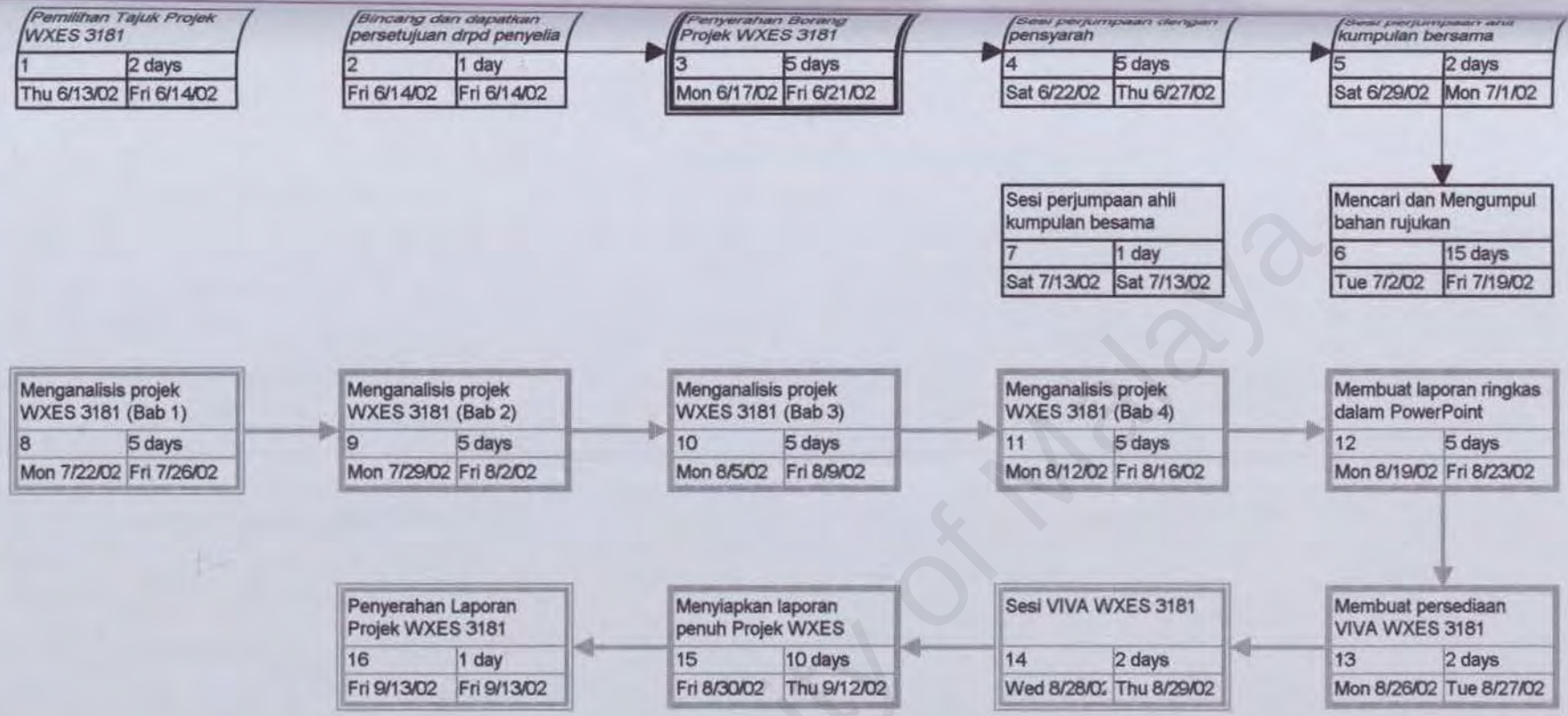
Jadual Pembangunan Projek WXES 3181

ID	Project WXES 3181 Activities	Duration	Start	Finish
1	Pemilihan Tajuk Projek WXES 3181	2 days	Thu 6/13/02	Fri 6/14/02
2	Bincang dan dapatkan persetujuan drpd penyelia	1 day	Fri 6/14/02	Fri 6/14/02
3	Penyerahan Borang Projek WXES 3181	5 days	Mon 6/17/02	Fri 6/21/02
4	Sesi perjumpaan dengan pensyarah	5 days	Sat 6/22/02	Thu 6/27/02
5	Sesi perjumpaan ahli kumpulan bersama penyelia	2 days	Sat 6/29/02	Mon 7/1/02
6	Mencari dan Mengumpul bahan rujukan	15 days	Tue 7/2/02	Fri 7/19/02
7	Sesi perjumpaan ahli kumpulan besama penyelia	1 day	Sat 7/13/02	Sat 7/13/02
8	Menganalisis projek WXES 3181 (Bab 1)	5 days	Mon 7/22/02	Fri 7/26/02
9	Menganalisis projek WXES 3181 (Bab 2)	5 days	Mon 7/29/02	Fri 8/2/02
10	Menganalisis projek WXES 3181 (Bab 3)	5 days	Mon 8/5/02	Fri 8/9/02
11	Menganalisis projek WXES 3181 (Bab 4)	5 days	Mon 8/12/02	Fri 8/16/02
12	Membuat laporan ringkas dalam PowerPoint	5 days	Mon 8/19/02	Fri 8/23/02
13	Membuat persediaan VIVA WXES 3181	2 days	Mon 8/26/02	Tue 8/27/02
14	Sesi VIVA WXES 3181	2 days	Wed 8/28/02	Thu 8/29/02
15	Menyiapkan laporan penuh Projek WXES 3181	10 days	Fri 8/30/02	Thu 9/12/02
16	Penyerahan Laporan Projek WXES 3181	1 day	Fri 9/13/02	Fri 9/13/02



WXES 3181
Litar Aritmetik Reja

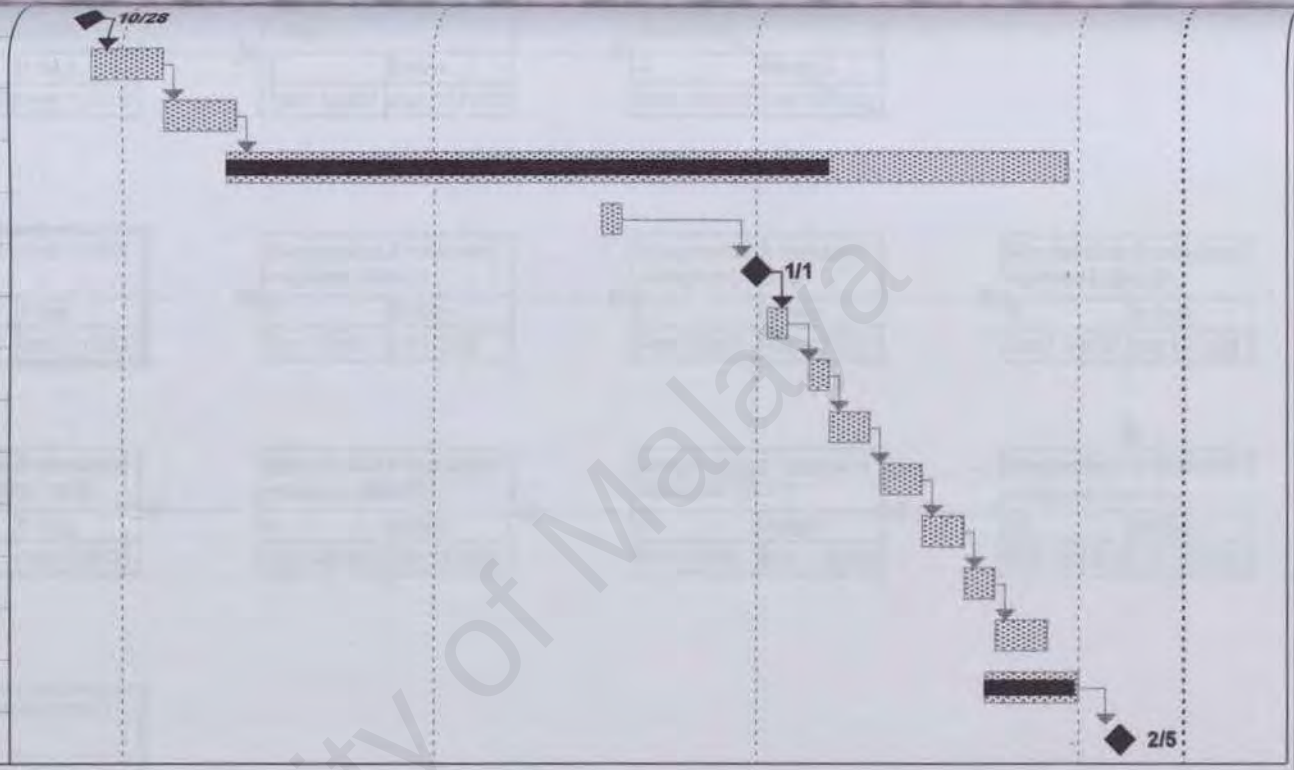
Task
Split
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Split
Rolled Up Milestone
Rolled Up Progress
External Tasks
Project Summary



APENDIKS B

Jadual Pembangunan Projek WXES 3182

1		Sesi perjumpaan bersama penyel	1 day
2		Kemaskini keperluan Projek WXE	5 days
3		Menganalisa bahasa VHDL	5 days
4		Menulis & menganalisa Kod VHD	60 days
5		Sesi perjumpaan bersama penyel	2 days
6		Penyerahan jurnal WXES 3182	1 day
7		Menganalisa & kemaskini maklun	2 days
8		Menganalisa & kemaskini maklun	2 days
9		Menganalisa & kemaskini maklun	4 days
10		Menganalisa & kemaskini maklun	4 days
11		Menganalisa & kemaskini maklun	2 days
12		Menganalisa & kemaskini maklun	3 days
13		Menganalisa & kemaskini maklun	3 days
14		Persediaan akhir Sesi VIVA WXE	7 days
15		Sesi VIVA WXES 3182	1 day



WXES 3182
Litar Aritmetik Reja

Task		Summary		Rolled Up Progress	
Split		Rolled Up Task		External Tasks	
Progress		Rolled Up Split		Project Summary	
Milestone		Rolled Up Milestone			

APENDIKS C

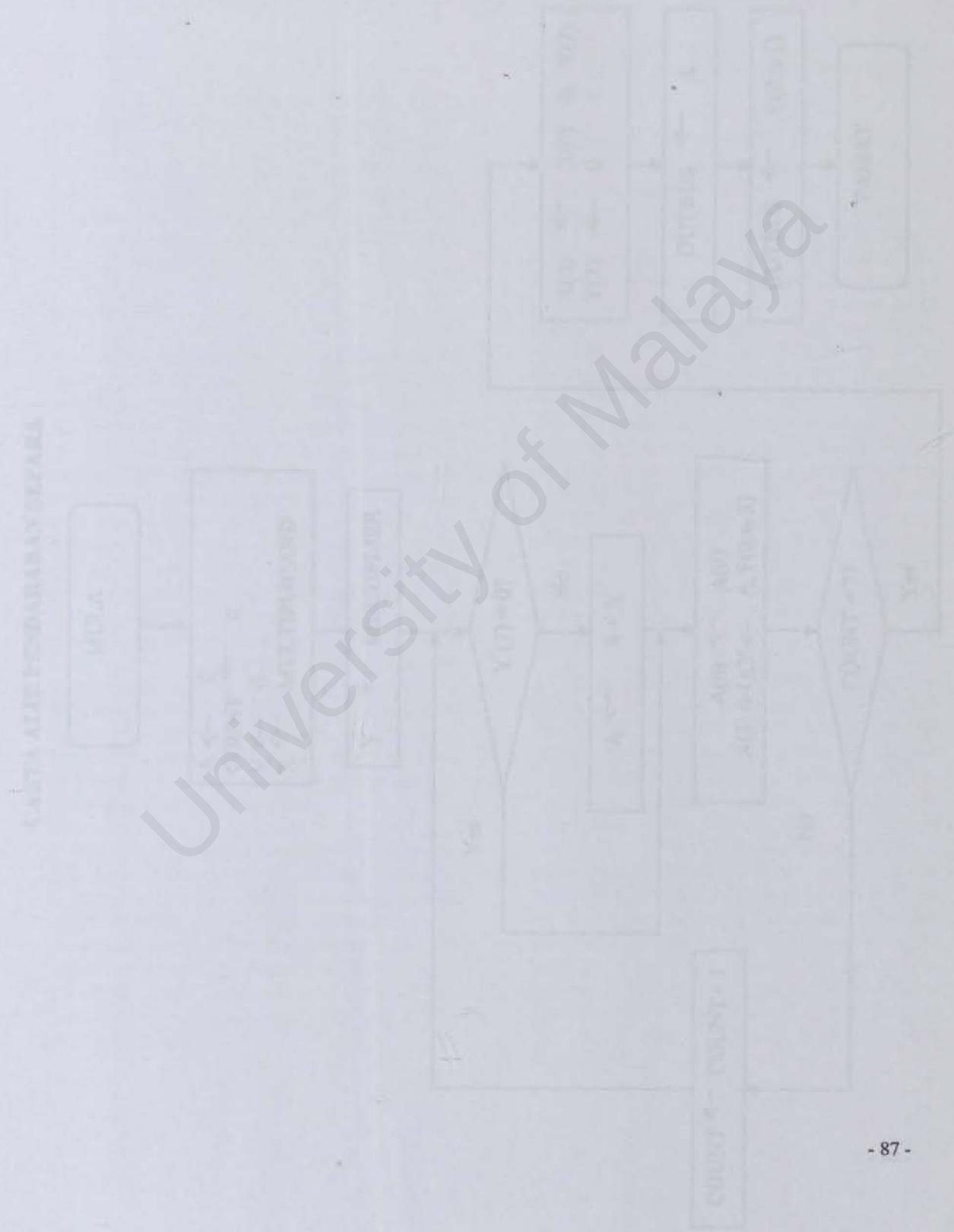
Kaedah pendaraban separa 8 bit

KEDARAB PENDARABAN SEPARA BIT

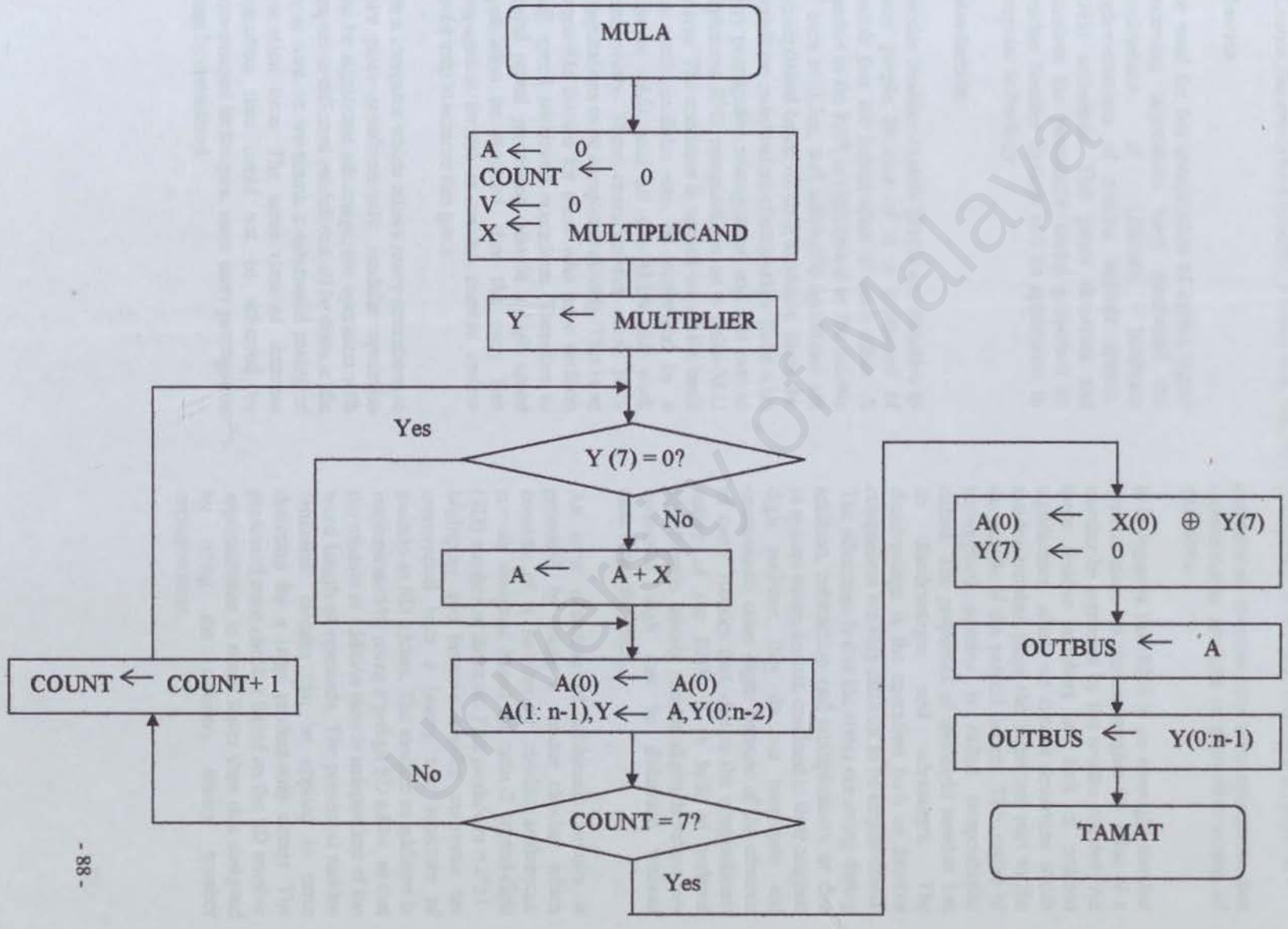
MULTIPLICAND:									x7	x6	x5	x4	x3	x2	x1	x0
MULTIPLIER:								×	y7	y6	y5	y4	y3	y2	y1	y0
DARAB SEPARA-1:									y0x7	y0x6	y0x5	y0x4	y0x3	y0x2	y0x1	y0x0
DARAB SEPARA-2:								y1x7	y1x6	y1x5	y1x4	y1x3	y1x2	y1x1	y1x0	
DARAB SEPARA-3:							y2x7	y2x6	y2x5	y2x4	y2x3	y2x2	y2x1	y2x0		
DARAB SEPARA-4:						y3x7	y3x6	y3x5	y3x4	y3x3	y3x2	y3x1	y3x0			
DARAB SEPARA-5:					y4x7	y4x6	y4x5	y4x4	y4x3	y4x2	y4x1	y4x0				
DARAB SEPARA-6:				y5x7	y5x6	y5x5	y5x4	y5x3	y5x2	y5x1	y5x0					
DARAB SEPARA-7:			y6x7	y6x6	y6x5	y6x4	y6x3	y6x2	y6x1	y6x0						
DARAB SEPARA-8:		+	y7x7	y7x6	y7x5	y7x4	y7x3	y7x2	y7x1	y7x0						
DARAB AKHIR:	p16	p15	p14	p13	p12	p11	p10	p9	p8	p7	p6	p5	p4	p3	p2	p1	p0

APENDIKS D

Carta alir pendaraban separa



CARTA ALIR PENDARABAN SEPARA



APENDIKS E

A Research about RNS and Arithmetic Circuits

Prepared by:
Leorna Shazerine(WEK000005), Fadzlin Nor Arbain(WEK000006), Wan Nurul Rukiah(WEK000176)

Abstract

The need for fast computation of digital signal processing algorithms have motivated the development of efficient hardware implementations of residue number system. (RNS) arithmetic. This paper describes and examines the potentially useful properties of Residue Number System and its application to computer technology.

Introduction

Residue Number System (RNS) are attractive to many people because of it is composed of moduli that are independent of each other. A number in the RNS is represented by the residue of each modulus, and arithmetic operations are accomplished based on each modulus. Since the moduli are independent of each other, there is no carry propagation among them, and it is easy to implement RNS computations on a multi-ALU system. The operation is carried out on the basis that each modulus can be performed by a separate ALU, and all the ALUs can work concurrently. These characteristics allow RNS computations to be completed quickly. This is an irresistible feature for those who have perform high speed arithmetic operations. Therefore in digital signal processing, should a high speed application be required, then the carry free propagation feature in residue number system could help to achieve this goals.

On a computer which allows many operations to take place simultaneously, modular operation can be significant advantage; the operation with respect to different moduli can all be done at the same time, so we obtain a substantial saving of execution time. The same time of decrease execution time could not be achieved by conventional techniques, since carry propagation must be considered.

computation involves only these operations, this representation provides an alternative system of arithmetic.

It is apparent that RNS is an unweight number system, since we cannot imagine the value of a number be represent by the residue number. All these residue numbers are lack of ordered significance, where we cannot determine which residue number plays the important part to the magnitude of the natural integer. This nature of unweighted number is called nonpositional nature. The properties of unweight number has its disadvantages and advantages. The disadvantage is the operation such as number comparison are very difficult to be implemented. The advantage is that the errors occurring during addition, subtraction and multiplication, or due to system noise, remain confined to their original digit position; they do not propagate and contaminate other digits because of the absence of carry. Besides that, due to the nonpositional nature of the RNS, it is lack of ordered significance among residue digits,therefore any erroneous digit can be discarded without destroying the result.

An error detection of arithmetic circuits is proposed, by using a residue checker which consists of a number of residue arithmectic circuits designed based on radix-2 signed-digit (SD) number arithmectic. Fast modulo $m = 2^p \pm 1$ Multiplier and binary-to-residue converter are constructed with a binary tree structure of modulo m SD adders. The modulo m addition is implemented by using a p-digit SD adder, so that the modulo m addition time is independent of the word length of operands. The presented residue arithmetic circuits can be applied to error detection for a large product-sum circuit. The presented error checker based on the SD number representation is much faster than that designed by using the ordinary binary number representation.

Background and Terminologies

Szabo and Tanaka [1] were among the earliest researchers to conduct a detailed investigation into the properties of the RNS. Their study emphasized the inherent parallel nature of residue arithmetic and its advantages for high speed computations in digital computers.
Mandelbaum [2] from his journal later developed some basic results in error detection and correction in the RNS. Barsi and Maestrini [3] demonstrated the requirement of minimal redundancy for erro correcting procedure that operates directly on the residue representation of the numbers. Later, an alternate approach to error correction in the RNS using product codes was described by Barsi and Maestrini [4].
Complex electronic systems have increasingly relied upon fault tolerence, self- checking, and VLSI design to improve the reliability and performance. In 1982, Taylor [5] investigated a VLSI residue arithmetic multiplier.

This paper presents introduction of RNS, and analysis of some important results with applications of Residue Number System Theory to digital signal processing and defines terminology that is used throughout the paper.

Redundant Residue Number Set and Signed-digit Number Presentation

A redundant residue number system is defined as a standard residue number system with r additional moduli. All L + r moduli must be relatively prime to ensure a unique representation for each number in the system.

In symmetric RNS, a residue digit
$x = /X/_m = X-[X/m]\ x\ m$ has a value in the following number set :

$$l_m=\{-(m-1)/2, \ldots 0, \ldots (m-1)/2\} \quad (1)$$

Where [X/m] is the closest integer rounding X/m. In the paper, let $m \in \{2^p-1, 2^p+1\}$. Since the carry propagation a risen during additions in the ordinary binary arithmetic system will limit the speed of arithmetic operations, we consider a residue number x represented by a *p*-digit radix – 2 SD number representation as follows :

$$x = x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0,$$
$$x_i \in \{-1,0,1\} \quad (2)$$
$$(i=0,1, \ldots p-1)$$

which can be denoted as $x = (x_{p-1}, x_{p-2}, \ldots x_0)_{SD}$. In the SD number representation, x has a values in the range of $[-(2^p-1), 2^p-1]$. However, it is difficult to judge if x is in l_m.

To simplify the manipulation of the residue operation in SD number representation;

$$L_m = \{-(2^p-1), \ldots -(m-1)/2, \ldots -1,0,1, \ldots (m-1)/2, \ldots 2^p-1\} \quad (3)$$

Thus, x must be in Lm when it is expressed in a *p*-digit SD number representation. Obviously

$$-x = -(x_{p-1}, x_{p-2}, \ldots x_0)_{SD}$$
$$= (-x_{p-1}, -x_{p-2}, \ldots -x_0)_{SD}$$

is in L_m.

Let X be an integer and $m \in \{2^p-1, 2^p+1\}$ be a modulous. Then $x=\langle X\rangle_m$ is defined as an integer in Lm. When $/X/_m \neq 0$, x has one of two possible values given by equations

$$x = \langle X\rangle_m = /X/_m \quad (4)$$

And

$$x = \langle X\rangle_m = /X/_m - \text{sign}(/X/m) \times m \quad (5)$$

respectively, where

$$\text{sign}(x) = \begin{cases} -1 & x<0 \\ 1 & x\geq 0 \end{cases}$$

When $/X/_m=0$, in the case of $m=2^p-1$, there are three possible values for x, that is,-m,0 and m.

In the above definition , L_m has the value range represented by the p-digit radix-2 SD number representation and the value set l_m in Eq(1) is a partial set of L_m for the residue digit. (The subscript *i* is omitted). For some integer X, there may be only one possible value by Eq(4). When $m = 2^p-1$, Lm has the largest redundancy to l_m. In the case of $m = 2^{p+1}-1$, $L_m = l_m$, that is non-redundant. We can use only one even number as a modulus in the redundant symmetric RNS.

Example 1 :
Let p = 3, m = 13. Thus,

$$l_m = \{-6,-5, \ldots 0, \ldots 5,6\}$$

And

$$L_m = \{-7,-6,-5, \ldots 0, \ldots 5,6,7\}$$

Thus $|20|_{13} = -6$ and $\langle 20\rangle_{13} = -6$, by Eq(4) or $|20|_{13} = 7$ by Eq(5). However,

$\langle 18\rangle_{13} = |18|_{13} = 5$, that is the only possible value.
When $m = 2^3-1 = 7$, $|21|_7 = 0$ and $\langle 21\rangle_7$ may have one of three possible values, that is -7,0 and 7.

Therefore the addition, subtraction and multiplication of tuples $A = (A_1, A_2, \ldots A_n)$ and $B = (B_1, B_2, \ldots B_n)$ in on RNS can be represented as follows :

$$A \pm B = (\langle A_1 \pm B_1\rangle_{m1}, \ldots \langle A_n \pm B_n\rangle_{mn}) \quad (6)$$

$$A \times B = (\langle A_1 \times B_1\rangle_{m1}, \ldots \langle A_n \times B_n\rangle_{mn}) \quad (7)$$

For a given modulus m, $2^p-1 \le m \le 2^{p+1}-1$, we have the following properties.

Property 1 : Let a and b be integers. Then

a) $abs(\langle a\rangle_m) \le 2^p-1$
b) $\langle a \pm b\rangle_m \equiv \langle\langle a\rangle_m \pm \langle b\rangle_m\rangle_m$
c) $\langle axb\rangle_m \equiv \langle\langle a\rangle_m \times \langle b\rangle_m\rangle_m$
d) $\langle -a\rangle_m \equiv -\langle a\rangle_m$

Where abs(x) is the absolute value of x and $\equiv$ indicates a binary congruent relation with modulus m.

The SD number representation has redundancy; for example, 7 may be represented by $(0,1,1,1)_{SD}$, $(1,-1,1,1)_{SD}$ or $(1,0,0,-1)_{SD}$ for p=4. By using the redundant number representation, parallel arithmetic can be achieved without the carry propagation which occurs during addition in an ordinary binary system.

Error Detection Based on Residue Arithmetic

An error detection method for arithmetic circuit is proposed by using a residue checker consists of a number of residue arithmetic circuits designed based on radix-2 signed digit (SD) number arithmetic. Residue arithmetic circuits can be applied to error detection for a large product-sum circuit.

A checker with residue arithmetic be applied to detect the calculation error of an ordinary binary arithmetic circuit [6,7]. A number of modulo $2^p \pm 1$ circuits have been proposed [1,8,9], which are considered to be applied into an RNS or a residue checker for high speed computations.

Carry propagation is limited to one position during additions of signed-digit (SD) number [10]. To perform the high speed residue arithmetic, a concept on residue arithmetic with the redundant residue representation using a p-digit radix-2 SD number system [11] have proposed, in which the carry propagation is limited to one position during residue additions and the residue operation is easy to performed. Based on the concept, a fast modulo m

($m \in \{2^p-1, 2^p+1\}$) multiplier and a binary-to-residue converter with a binary tree structure of SD adders then apply the residue arithmetic circuits to design a fast error checker. Error checker based on the SD number representation is much faster than that designed by using the ordinary binary number representation.

A product-sum circuit can be used in a multiply-accumulate (MAC) unit of a processor, performs the following arithmetic function :

$$Z = A \times B + C \quad (8)$$

In the above equation, A and B are in the n-bit, C and Z are in the 2n-bit binary number representation, respectively. By converting the operands into residue number, that is, $a = |A|_m$, $b = |B|_m$ and $c = |C|_m$, we have the following residue product-sum equation.

$$Z = ||A|_m \times |B|_m + |C|_m|_m \quad (9)$$

Where

$$|X|_m = X \bmod m \quad (10)$$

And $m = 2^p-\mu$ and $\mu \in \{-1,1\}$. Since $|X|_m$ usually has a value in the following number set I_{mo} :

$$I_{mo} = \{0,1, \ldots (m-1)\} \quad (11)$$

$|Z|_m$, $|A|_m$, $|B|_m$, and $|C|_m$ are in the p-bit binary number representation, respectively. We convert the result of the product-sum calculation of Eq(8) into a residue number

$$Z' = |Z|_m = |A \times B + C|_m \quad (12)$$

When an error of the product-sum calculation occurs and results in $Z' \ne Z = A \times B + C$ if $z' = |Z|_m \ne z\,||A|_m \times |B|_m + |C|_m|_m$, then the error can be detected. Therefore, let the error signal be E, the

$$E = z - z' \quad (13)$$

From Eqs (9),(12) and (13), the residue checker consists of a residue product-sum circuit, four binary-to-residue converters and a subtraction

circuit. The binary-to-residue converts perform the operating A,B,C and Z from n-bit or 2n-bit binary number into $|A|_m$, $|B|_m$, $|C|_m$ and $|Z|_m$ all with p-bit binary number, where n>>p. When $E \neq 0$, the calculation error, which may occur in the product-sum circuit or in the error checker is detected. In conventional methods, the residue checker is implemented based on the ordinary binary number arithmetic, then the carry propagation and the complicated residue operation limit the operation speed of the residue checker.

An error checker constructed as shown in fig 1. There are a residue product-sum circuit consisting of a modulo m multiplier and adder and four binary-to-residue converters in the error checker. The binary-to-residue converter performs the operation converting A with an n-bit binary number representation, for example, in to $a=\langle A \rangle_m$ with a p-digit SD number representation. When $E \neq 0$ the calculation error, which may occur in the product-sum circuit or in the error checker, is detected. Since it is easy to detect E is 0 or not, the presented error using the SD number representation is much faster than that using the ordinary number representation.

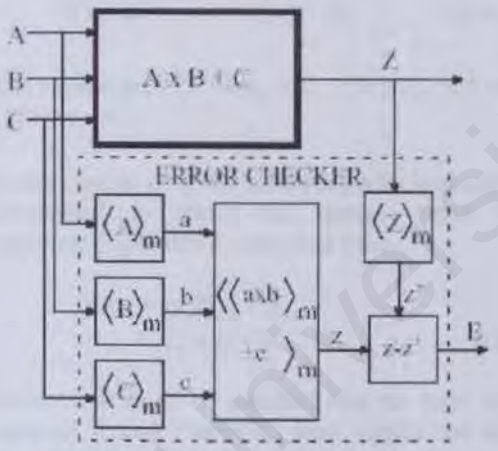


Fig.1 Arithmetic circuit with a residue checker

Modulo m SD Adder

Fig 2 shows a modulo m SD adder (MSDN) having p SD full adders (SDFAs). Each SDFA is constructed with blocks ADD1 and ADD2 the sum of two p-digit SD numbers, s = a+b, can be obtained by performing the following two steps (ADD1 and ADD2).

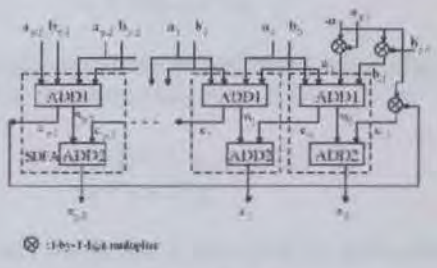


Fig.2 Modulo m SD adder (MSDA)

Algorithm 1 (a+b)

Let c_i, w_i and z_i be the intermediate carry, intermediate sum and sum at the ith SD digit (i = 0,1, ,,p-1),respectively. For each digit the following two steps are performed.

1. (ADD1) : when $abs(a_i) = abs(b_i)$

$$w_i = 0 \qquad ...(14)$$

and

$$c_i = (a_i + b_i) \text{ div } 2 \qquad ...(15)$$

when $abs(a_i) \neq abs(b_i)$

$$w_i = \begin{cases} -(a_i + b_i) & \text{if}(a_i + b_i) \text{ and } (a_{i-1} + b_{i-1}) \\ & \text{have the same sign} \\ (a_i + b_i) & \text{otherwise} \end{cases}$$

and

$$c_i = \begin{cases} -(a_i + b_i) & \text{if}(a_i + b_i) \text{ and } (a_{i-1} + b_{i-1}) \\ & \text{have the same sign} \\ 0 & \text{otherwise} \end{cases}$$

since $a_i, b_i \in \{-1,0,1\}$, $w_i, c_i \in \{-1,0,1\}$

2. (ADD2):$z_i = w_i + c_{i-1}$, where $c_{-1} = 0$.Thus

$$z = c_{p-1}2^p + z_{p-1}2^{p-1} + z_{p-2}2^{p-2} + \ldots + z_0$$
$$= z_p 2^p + Z_0 \qquad ...(18)$$

where $z_p = c_{p-1}$ and $Z_0 = z_{p-1}2^{p-1} + z_{p-2}2^{p-2} + \ldots + z_0$

In the above algorithm, it is always true that $2c_i + w_i = a_i + b_i$ and w_i and c_{i-1} do not have the same sign so that $z_i \in \{-1,0,1\}$. Thus the carry propagation is limited to one digit and parallel arithmetic can be achieved without the carry propagation which occurs during addition in an ordinary binary system.

Let $m = 2^p + \mu$ and $\mu \in \{-1,1\}$, then

$$\langle 2^p \rangle_m = -\mu \qquad ...(19)$$

In the case of $\mu = 0, m = 2^p$, $\langle z \rangle_m = Z_0$. For $\mu = \pm 1$ that is $m = 2^p \pm 1$,

$$\langle c_{p-1}\, 2^p \rangle_m = -c_{p-1} \times \mu = c_{-1}$$

where $c_i \in \{-1,0,1\}$, c_0 and w_0 are decided by using $a_{p-1}\mu$ and $b_{p-1}\mu$ as a_{-1} and a_{-1}, respectively. Therefore, the modulo m addition, $m = 2^p$ or $m = 2^p \pm 1$, can be performed by the following Algorithm.

Algorithm 2A ($\langle a+b \rangle_m$)

For $\mu = 0$ or $\mu = \pm 1$, let $c_{-1} = -\mu c_{p-1}$, $a_{-1} = -\mu a_{p-1}$ and $b_{-1} = -\mu b_{p-2}$ then perform the same two steps as Algorithm 1.

Example 2 :

Let p=5, $a=(1,0,-1,1,1)_{SD}$ and $b=(1,1,-1,0,0)_{SD}$. Thus, $m_1=2^p-1=31$, $m_2=2^p=32$ and $m_3=2^p+1=33$, a=15 and b=20.

	a) m= 31	b) m= 32	c) m= 33
i:	4 3 2 1 0	4 3 2 1 0	4 3 2 1 0
a:	1 0 -1 1 1	1 0 -1 1 1	1 0 -1 1 1
1)b:	1 1 -1 0 0	1 1 -1 0 0	1 1 -1 0 0
z:	0 1 0 -1 -1	0 1 0 -1 -1	0 1 0 -1 -1
2)c:	1 0 -1 1 1	1 0 -1 1 1	1 0 -1 1 1
s:	0 0 1 0 -1	0 0 1 0 -1	0 0 1 0 -1

The results are $\langle 15+20 \rangle_{31} = 4$, $\langle 15+20 \rangle_{32} = 3$ and $\langle 15+20 \rangle_{33} = 2$.

In the case of $\mu > 1$, the following addition can be considered to obtain the residue result by applying Algorithm 1 one more time

$$\begin{aligned} \langle Z \rangle_m &= \langle z_p 2^p \rangle_m + Z_0 \\ &= -z_p \times \mu + Z_0 \end{aligned} \qquad ..20)$$

However it must be satisfied that no such new carry as -1 and 1 from the most significant digit is generated again.

Example 3 :

Let p=3, m=13, $a=(1,-1,0)_{SD}=2$ and $b=(1,1,1)_{SD}=7$. Then $\mu = m-2^p = 13-8 = 5 = (1,0,1)_{SD}$. Thus by applying Algorithm 1 twice,

$$\begin{aligned} Z &= a+b \\ &= (1,-1,0)_{SD} + (1,1,1)_{SD} \\ &= (1,0,0,1)_{SD} = 9 \end{aligned}$$

then

$$\begin{aligned} \langle z \rangle_{13} &= -\mu + Z_0 \\ &= -(1,0,1)_{SD} + (0,0,1)_{SD} \\ &= (-1,0,-1)_{SD} + (0,0,1)_{SD} \\ &= (-1,0,0)_{SD} = -4 \end{aligned}$$

The residue operation is performed successfully. However, in the case of $z = (1,-1,1,1)_{SD} = 7$

$$\begin{aligned} \langle z \rangle_{13} &= (-1,0,-1)_{SD} + (-1,1,1)_{SD} \\ &= (-1,-,1,0)_{SD} \end{aligned}$$

This result shows that a new carry,-1, at the most significant digit was generated.

As shown in Algorithm 2A, only one p-digit SD adder is needed to implement the residue addition for $\mu=0$ and $\mu=\pm 1$. For $\mu>1$, the residue addition can be considered by directly using addition is calculated by performing Algorithm twice. To overcame the new carry generation in the second SD addition, we select m as the modulus which has a value in $2^p + \mu$ and $\mu \in \{-1,1\}$. Then $\mu = m-2^p$ is in the range of $[-1,2^{p-1}-1]$. For $\mu>1$, μ is expressed in a p-digit SD number representation as follows :

$$\mu = (0,1, \mu p-3, \ldots \mu 0)_{SD} \qquad (21)$$

where $\mu_i \in \{-1,0,1\}$ for $i = 0,1, \ldots p_{-3}$. The largest value of μ is $2_{p-1}-1$, which is $(0,1,1, \ldots 1)_{SD}$. Therefore the modulo m SD addition, where $m = 2^p+\mu$, $\mu>1$, can be performed by the following algorithm.

Algorithm 2B ($\langle a+b \rangle_m$)

Let a and b be two integers in the p-digit SD number representation, and $\mu = m-2^p$ ($\mu>1$) expressed in Eq (21)

1) Calculate the sum of a and b by Algorithm 1

$$\begin{aligned} z &= a + b \\ &= (z_p, z_{p-1}, \ldots z_0)_{SD} \end{aligned}$$

Where $z_i \in \{-1,0,1\}$ for $i = 0,1, \ldots p$.

2) Calculate the residue number $s=\langle z \rangle_m$ in the following three cases with respect to the value of z_p.

(Case 1) when $z_p = 0$

$$\begin{aligned} s &= (z_{p-1}, z_{p-2}, \ldots z_0)_{SD} \\ &= (s_{p-1}, s_{p-2}, \ldots s_0)_{SD} \end{aligned}$$

(Case 2) when $z_p \neq 0$ and $z_p = -z_{p-1}$, let $s_{p-1} = -z_{p-1}$, so that

$$s = (-z_{p-1}, z_{p-2}, \ldots, z_0)_{SD}$$
$$s = (s_{p-1}, s_{p-2}, \ldots, s_0)_{SD}$$

(Case 3) when $z_p \neq 0$ and $z_p \neq -z_{p-1}$

$$\begin{aligned} s &= (z_{p-1}, z_{p-2}, \ldots, z_0)_{SD} - z_p\mu \\ &= (z_{p-1}, z_{p-2}, \ldots, z_0)_{SD} + (0, -z_p, \\ &\quad -z_p\mu_{p-3}, \ldots, z_p\mu_0)_{SD} \\ &= (s_{p-1}, s_{p-2}, \ldots, s_0)_{SD} \end{aligned}$$

The calculation of $\langle a-b \rangle_m$ can be realized by replacing b with $\bar{b}$ in the above modulo m addition algorithms.

Example :
Let p=3, m=11, $a=(1,-1,0)_{SD}=2$ and $b=(1,-1,1)_{SD}=7$. Then $\mu=m-2^p=11-8=3=(011)_{SD}$. Thus by Algorithm 2B, we have

$$\begin{aligned} Z &= a+b \\ &= (1,-1,0)_{SD} + (1,1,1)_{SD} \\ &= (1,0,0,1)_{SD} = 9 \end{aligned}$$

and

$$\begin{aligned} s &= \langle z \rangle_{11} \\ &= -(0,1,1)_{SD} + (0,0,1)_{SD} \\ &= (0,-1,-1)_{SD} + (0,0,1)_{SD} \\ &= (0,-1,0)_{SD} = -2 \end{aligned}$$

For $z = (1,-1,1,1)_{SD} = 7$, $s = (1,1,1)_{SD}$.
When $z = (1,1,1,1)_{SD} = 15$

$$\begin{aligned} s &= \langle z \rangle_{11} \\ &= (0,-1,-1)_{SD} + (1,1,1)_{SD} \\ &= (1,0,0)_{SD} = 4 \end{aligned}$$

Modulo *m* Multiplication

In this section, we present a parallel algorithms to perform the modulo *m* multiplication by use of modulo *m* addition algorithms. The parallel algorithm can be implemented by using a binary tree of modulo m SD adders (MSDAs) for the modulo *m* sum of the partial products. Therefore, when the partial products are obtained in a time no longer than O(log(*p*)) by fixing the value of *m*, the modulo *m* multiplication is performed in a time proportional to $\log_2 p$. A binary tree of the modulo *m* SD adders (MSDAs) can be constructed for the modulo *m* sum of the partial products as shown in fig 3.

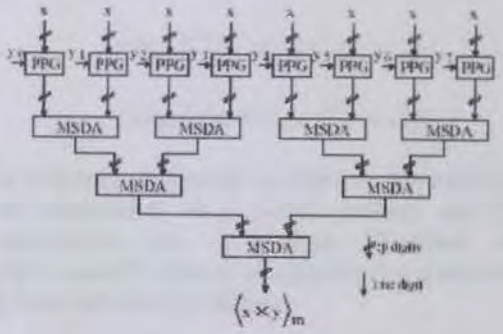


Fig.3 Modulo *m* SD multiplier (MSDM)

Parallel Modulo *m* Multiplication Algorithm
To calculate $\langle x \times y \rangle_m$, where x and y are integers in the p-digit radix-2 SD number representation, $x \times y$ is expanded as follows:-

$$\begin{aligned} x \times y &= (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0) \\ &\quad \times (y_{p-1}2^{p-1} + y_{p-2}2^{p-2} + \ldots + y_0) \\ &= \sum_{i=0}^{p-1} y_i 2^i \times (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0) \end{aligned}$$

Thus,

$$\begin{aligned} \langle x \times y \rangle_m &= \left\langle \sum_{i=0}^{p-1} \langle y_i 2^i \times (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots \right. \\ &\quad \left. + x_0) \rangle_m \right\rangle_m \\ &= \left\langle \sum_{i=0}^{p-1} pp_i \right\rangle_m \end{aligned}$$

where

$$pp_i = \langle y_i 2^i \times (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0) \rangle_m \quad \ldots(22)$$

denotes as a partial product. Since $y_i \in \{-1,0,1\}$,

$$pp_i = y_i \langle 2^i \times (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0) \rangle_m \quad \ldots(23)$$

Thus, a parallel algorithm for the modulo *m* multiplication can be implemented with the following three steps.

Algorithm 3 ($\langle x \times y \rangle_m$)
Let x and y be two p-digit radix-2 SD numbers.

(1) Calculate the residue of i-digit shifted numbers in parallel for $i = 0,1, \ldots, p-1$,

$$sx_i = \langle 2^i \times x \rangle_m \quad ..(24)$$

(2) Multiply sx_i by y_i to obtain the partial product, pp_i ($i = 0,1, \ldots, p-1$), shown in Eq. (23).

$$pp_i = y_i \times sx_i$$

(3) Calculate the modulo m sum of these partial products by performing Algorithm 2A or 2B repeatedly.

$$\langle x \times y \rangle_m = \langle pp_0 + pp_1 + \ldots + pp_{p-1} \rangle_m$$

A binary tree of modulo m SD adders can be constructed for the modulo m sum of the partial products in the step (3).

From $\langle 2^p \rangle_m = -\mu$ shown in Eq. (19), we have

$$sx_i = \langle 2^i \times (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0) \rangle_m$$

$$= \langle \langle 2^p (x_{p-1}2^{i-1} + x_{p-2}2^{i-2} + \ldots + x_{p-i}) \rangle_m$$

$$+ x_{p-i-1}2^{p-1} + x_{p-i-2}2^{p-2} + \ldots + x_0 2^i \rangle_m$$

$$= \langle x_{p-i-1}2^{p-1} + x_{p-i-2}2^{p-2} + \ldots + x_0 2^i$$

$$- \mu (x_{p-1}2^{i-1} + \ldots + x_{p-i+1}2 + x_{p-i}) \rangle_m$$

In this cases of $\mu=0$ and $\mu=\pm 1$,

$$sx_i = (x_{p-i-1}, x_{p-i-2}, \ldots, x_0$$

$$-\mu x_{p-1}, \ldots, -\mu x_{p-i+1}, -\mu x_{p-i})_{SD}$$

Thus, a partial product is simply obtained by an i-digit end-around-shift and a p-by-1 digit multiplication. The modulo m multiplications can be performed in a time proportional to $\log_2 p$.

For $\mu > 1$, however, it is not so easy to obtain the partial products.

When $\mu = 3 = 2^1 + 1$, for example,

$$sx_i = \langle x_{p-i-1}2^{p-1} + x_{p-i-2}2^{p-2} + \ldots + x_0 2^i$$

$$- \mu (x_{p-1}2^{i-1} + \ldots + x_{p-i+1}2 + x_{p-i}) \rangle_m$$
$$= \langle x_{p-i-1}2^{p-1} + x_{p-i-2}2^{p-2} + \ldots + x_0 2^i$$

$$- (x_{p-1}2^{i-1} + \ldots + x_{p-i+1}2 + x_{p-i}))$$

$$- (x_{p-1}2^i + \ldots + x_{p-i+1}2^2 + x_{p-i}2) \rangle_m .$$

In this case, a modulo m addition is needed for the calculation of a partial product and the computation time is constant. Therefore, we select a small value of to simplify the generation process of partial products.

Conclusion

The error detection method by using a residue checker has been presented. A new concept on the residue arithmetic circuits are using a p-digit signed-digit number representation. Based on the proposed algorithms, we select an integer m, $2^p - 1 \le m \le 2^p + 2^{p-1} - 1$ as modulus in an RNS and the modulo m addition is implemented by an SD adder or two SD adders. A modulo m multiplier can be constructed with a binary MSDA tree for the high speed calculations. Since they have a binary tree structure of radix-2 signed-digit number modulo m adders, the modulo m multiplication is performed in a time proportional to $\log_2 p$ and an n-bit binary number is converted into a p-digit SD residue number in a time proportional to $\log_2 (n/p)$. the application to the error detection of a product-sum computation is considered by using the SD modulo m arithmetic circuits and the presented error checker with the residue arithmetic circuits is faster than that with the residue binary arithmetic.

High-speed computations can be performed based on the assumption that input and output data of the residue arithmetic circuits are in the residue SD number form, because some computing system applications, such as digital filtering, require repeated calculations of sums of products before the final results are obtained. For integration with conventional binary systems, efficient circuits are required to convert into an out of the residue SD systems.

References

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

[2] D.Mandelbaum, "Error correction in residue arithmetic, "IEEE Trans. Computers, vol. C-21, pp.538-545, June 1972.

[3] F.Barsi and P.Maestrini, "Error correcting of redundant RNS, "IEEE Trans.Computers, vol C-23, pp.915-924, Sept 1974.

[4] F.Barsi and P.Maestrini, "Error detection and correction by product codes in RNS, "IEEE Trans.Computers, vol C-23, pp.915-924, Sept 1974.

[5] W.Kenneth Jenkins, "Self Checking Properties of Residue Number Error Checkers Based On Mixed Radix Conversion, "IEEE Trans.Computers, August 18,1988.

[6] A.Anizienis, "Arithmetic algorithms for error-coded operands, "IEEE Trans. Computers, vol. C-22, no.pp.567-572, June 1973.

[7] T.R.N.Rao, "Biresidue error-correcting codes for computer arithmetic, "IEEE Trans. Computers, vol. C-19, no.pp.398-402, May 1970.

[8] F.J.Taylor, "A VLSI residue arithmetic multiplier, "IEEE Trans. Computers, vol. C-31, no.pp.540-546, June 1982.

[9] A.Hiasat, "New memoryless, mod (2n1) residue multiplier, "Electron. Lett., vol. 28, no.3, pp.314-315, Jan 1992.

[10] A.Anizienis, "Signed-digit number representations for fast parallel arithmetic, "IRE Trans. Elect.Computers, EC-10, pp.389-400, Sept. 1961.

[11] S.Wei and K.Shimizu, "A Novel Residue Arithmetic Hardware Algorithm Using a Signed-Digit Number Representation, "IEEE Trans. Inf. & Syst. vol. E-83D, no.12, pp.2056-2064, Dec. 2000.

RUJUKAN

[1] N.S.Szabo and R.I.Tanaka, " Residue Arithmetic and Its Applications to Computer Technology", New York :McGraw-Hill, 1967.

[2] D.P. Agrawal and T.R.N.Rao,"Modulo (2 + 1) arith-metic logic,"IEE J. Electronic Circuits and Systems,Vol.2, pp. 186-188, Nov. 1978.

[3] F.J.Taylor,"A VLSI residue arithmetic multiplier,"IEEE Trans. Comput., Vol.C-31, pp.540-546,June 1982.

[4] A.Hiasat,"New memoryless, mod $(2n \pm 1)$ residue mul-tiplier," Electron. Lett., Vol.28, No.3, pp.314-315,Jan.1992.

[5] A.Avizienis,"Signed-digit number representations forfast parallel arithmetic,"IRE Trans. Elect. Comput.,EC-10,pp.389-400, Sept. 1961.

[6] S.Wei and K.Shimizu, "A Novel Residue Arithmetic Hardware Algorithm Using a Signed-Digit Number Representation", IEICE Trans.Inf. & Syst.,Vol.E83-D, pp.2056-2064,Dec.2000.

[7] S.Wei and K.Shimizu, "Residue Arithmetic MultiplierBased on the Radix-4 Signed-Digit Multiple-Valued Arithmetic Circuits", pp.1-6.

[8] S.Wei and K.Shimizu, "Residue Arithmetic with a Signed-Digit Number System", pp.349-354.

[9] S.Wei and K.Shimizu, "Residue Arithmetic Circuits Using a Signed-Digit Number Representation", Int. Symp. On Circuits and Systems, pp.24-27,May 2000.

[10] S.Wei and K.Shimizu, "Residue Arithmetic Circuits Based on the Signed-Digit Number Representation and the VHDL Implementation, pp.1-4.

[11] S.Wei and K.Shimizu, "Error Detection of Arithmetic Circuits Using a Residue Checker with Signed-Digit Number System", Proc. 28th Int. Symp. On Defect and FaultTolerance in VLSI Systems, pp.1-6,2001.

[12] N. Zainalabedin. (1998). VHDL Analysis and Modeling of Digital Systems. 2nd ed. McGraw –Hill.

[13] Brown, S. D.(2000). Fundamentals of digital logic with VHDL design . Singapore : McGraw-Hill.

[14] http://www.acc-eda.com/vhdlref/refguide/language_overview/using_standard_logic/ieee

[15] http://www.acc-eda.com/products/pkfpgads_facrs.html

[16] http://www.vcc.com/fpga.html