A Built-In-Self-Test Arithmetic Logic Unit (A BIST ALU)

Disediakan Oleh
Suhaila Binti Ab Aziz

Laporan Latihan Ilmiah ini dihantar
Bagi memenuhi keperluan bergraduasi bagi
Sarjana Muda Sains Komputer
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Malaya

Februari 2003

ABSTRAK

Perkembangan pesat teknologi litar bersepadu telah merintis kepada perkembangan pembinaan litar sersepadu berskala besar (*Very Large Scale Intergrated Circuit* – VLSI). Dalam usaha meningkatkan prestasi, dan kebolehpercayaan litar-litar VLSI yang dibangunkan, pembangun perlulah memastikan ia bebas daripada sebarang bentuk kecacatan atau ralat. Ini merupakan pendorong kepada perkembangan kaedah *Design For Testability* (DFT) yang berfungsi untuk membantu para pembangun bagi kecacatan litar.

Built-In-Self-Test (BIST) merupakan salah satu teknik yang menggunakan konsep menggabungkan litar ujian bersama litar yang sedang diuji. Matlamat utama BIST adalah untuk mengurangkan pergantungan litar yang direkabentuk kepada penguji dan memperoleh kemudahan ujian pada kelajuan operasi kelajuan sistem dengan liputan kegagalan yang tinggi. Ia meliputi tiga fungsi utama iaitu; penjanaan vektor ujian, penggunaan vektor ujian dan analisis pengenalan (*signature analysis*). Sebagai permulaan kepada pembangunan BIST, 8 bit *Arihtmetic Logic Unit* (ALU) telah digunakan sebagai litar untuk diuji.

Pembinaan *Built-In-Self-Test Arithmetic Logic Unit* (BIST ALU) bermula dengan memahami konsep sebenar BIST dan ALU. Perlakuan kedua-dua komponen ini akan dimodulkan dalam bahasa pengaturcaraan *VHSIC Hardware Description Language* (VHDL) sebelum digabungkan. Seterusnya keseluruhan litar tersebut akan disimulasikan dan diuji bersama litar yang sedia ada bagi menguji prestasi dan kebolehpercayaan litar yang dibangunkan.

KANDUNGAN

Abstrak ii

Penghargaan iv

Kandungan vi

Senarai Jadual viii

Senarai Rajah ix

Bab 1 Pendahuluan **1**

1.1 Pengenalan

1.2 Skop Projek

1.3 Objektif Projek

1.4 Penjadualan Projek

1.5 Kekangan

1.6 Ringkasan Bab

1.7 Kesimpulan

Bab 2 Kajian Literasi **11**

2.1 Pengenalan

2.2 *Arithmetic Logic Unit*

2.3 Penambah (*Adder*)

2.4 *Designs For Testability*

2.5 Kesimpulan

Bab 3 Metodologi **35**

3.1 Pengenalan

3.2 Pendekatan Pembangunan BIST ALU

3.3 Kaedah Implementasi BIST ALU

3.5 Pembahagian Tugas Membangunkan BIST ALU

3.6 Kesimpulan

Bab 4 Rekabentuk Sistem **43**

4.1 Pengenalan

4.2 Pembangunan 8 Bit ALU

4.3 Rekabentuk BIST ALU

4.4 Kesimpulan

Bab 5 Implementasi Sistem **56**

5.1 Pengenalan

5.2 *PEAKFPGA Designer Suite FPGA Syntesis Edition 5.20c*

5.3 Strategi Pembangunan BIST ALU

5.4 Pengujian

5.5 Kesimpulan

Bab 6 Pengujian Sistem 73

6.1 Pengenalan

6.2 Pengujian Modul

6.3 Pengujian Integrasi

6.4 Pengujian Sistem

6.5 Kesimpulan

Bab 7 Penilaian Sistem 82

7.1 Pengenalan

7.2 Masalah Yang Dihadapi

7.3 Kelebihan BIST ALU

7.4 Kekurangan BIST ALU

7.5 Cadangan

7.6 Kesimpulan

Rujukan

Lampiran 1 – Manual Pengguna

Lampiran 2 – Kod Sumber

SENARAI JADUAL

JADUAL	TAJUK
Jadual 1.1	Operasi Yang Dilaksanakan Oleh ALU 8 Bit Yang Dibangunkan
Jadual 1.2	Penjadualan Pembangunan BIST ALU
Jadual 1.3	Aktiviti Fasa 3 dan Fasa 4
Jadual 2.1	Contoh Operasi Aritmetik Bagi Unit Aritmetik Dan Logik 4 Bit
Jadual 2.2	Jadual Kebenaran Bagi Penambah Penuh
Jadual 2.3	Kos bagi mengenalpasti dan mendiagnosis ralat pada pelbagai peringkat
Jadual 2.4	Suap Balik bagi Urutan Kepanjangan Maksimum LFSR
Jadual 2.5	Mod Operasi Bagi BILBO
Jadual 4.1	Jadual Kebenaran Unit Aritmetik 8 Bit ALU
Jadual 4.2	Jadual Kebenaran Bagi Unit Logik 8 Bit ALU

SENARAI RAJAH

RAJAH	TAJUK
Rajah 2.1	Rajah blok Unit Aritmetik Dan Logik
Rajah 2.2	Rajah Kotak Hitam Bagi Aritmetik Dan Logik 4 bit
Rajah 2.3	Gabungan Unit Aritmetik Dan Unit Logik Bagi Membentuk Unit Aritmetik Dan Logik
Rajah 2.4	Litar Penambah Penuh
Rajah 2.5	Penambah Selari 4 Bit
Rajah 2.6	Rajah Blok Bagi CLA 4 Bit
Rajah 2.7	Konsep Pengujian BIST
Rajah 2.8	Rajah Blok LFSR
Rajah 2.9	Rajah Blok MISR
Rajah 3.1	Pendekatan Bagi Membangunkan BIST ALU
Rajah 4.1	Rajah kotak hitam bagi 8 bit ALU
Rajah 4.2	Rajah Blok Litar Aritmetik
Rajah 4.3	Operasi Yang Diilaksanakan Oleh 8 Bit ALU
Rajah 4.4	Rajah Logik Bagi Litar Logik 8 Bit ALU
Rajah 4.5	Rekabentuk BIST ALU
Rajah 5.1	Carta alir langkah penggunaan PEAKFPGA
Rajah 5.2	Proses mencipta projek
Rajah 5.3	Proses mencipta modul.
Rajah 5.4	Proses mengkompil aturcara ralat.

Rajah 5.5	Isyarat digital yang diperolehi semasa mensimulasi modul.
Rajah 5.6	Carta Alir Pembangunan BIST ALU
Rajah 5.7	Hirarki pembinaan BIST ALU
Rajah 5.8	Contoh pengisytiharan entiti bagi *Carry Lookahead Adder*.
Rajah 5.9	Contoh Pengisytiharan *Behavior Bagi Carry Lookahead Adder*
Rajah 5.10	Pengisytiharan Struktur Bagi Litar Logik Bagi *Arithmetic Logic Unit* (ALU)
Rajah 6.1	Contoh *Test Bench* Bagi Menguji *Carry Lookahead Adder*
Rajah 6.2	Graf Isyarat Digital Hasil Simulasi *Carry Lookahead Adder* Berdasarkan Input Yang Dimasukkan Oleh *Test Bench*
Rajah 6.3	Gambaran Proses Pengujian Bagi Pembangunan BIST ALU

BAB 1

PENDAHULUAN

BAB 1

PENDAHULUAN

1.1. PENGENALAN

Dalam era perkembangan teknologi masakini, rata-rata syarikat perkomputeran sedang memperkatakan tentang teknologi litar bersepadu (*IC Technology*). Matlamat utama syarikat-syarikat ini adalah bagi menghasilkan cip yang bersaiz kecil tetapi mampu melaksanakan fungsi yang banyak. Cabaran utama bagi mencapai matlamat ini adalah bagi memastikan cip yang dibina itu dapat berfungsi dengan baik dan bebas ralat.

Bagi memastikan kewibawaan dan kebolehpercayaan bagi cip tersebut, pengujian demi pengujian perlulah dilakukan dari peringkat awal. Semakin kompleks litar bagi sesuatu cip, semakin sukar pengujian dilakukan ke atasnya dan kos pengujian juga akan turut meningkat.

Built-In-Self-Test (BIST) merupakan salah satu jalan penyelesaian bagi memastikan pengeluaran kos pengujian yang minima. BIST merupakan salah satu teknik *Design For Testability* (DFT) yang mana litar BIST tersebut digabungkan bersama litar yang ingin diuji. Dengan ini sebarang ralat atau

kecacatan yang timbul dapat dikesan diperingkat awal dan secara tidak langsung ia akan mengurangkan kos penyelenggaraan terhadap cip tersebut.

Sebagai titik permulaan bagi menghasilkan BIST, *Arithmetic Logic Unit* (ALU) 8 bit telah digunakan sebagai unit pengujian. Namun begitu, BIST ini juga boleh diaplikasikan ke atas mana-mana litar bersepadu yang lebih besar dan kompleks.

Built-In-Self-Test Arithmetic Logic Unit (BIST ALU) merupakan salah satu teknik rekabentuk yang menggunakan ALU sebagai unit untuk diuji Aplikasi ini membenarkan mengujian dilakukan ke atas ALU dengan menggabungkan BIST bersama litar ALU sebagai satu komponen. Bagi melaksanakan projek Latihan Ilmiah Tahap Akhir (WXES 3182), ALU 8 bit telah digunakan sebagai unit untuk diuji bagi komponen ini.

1.2. SKOP PROJEK

Pembinaan BIST ALU ini melibatkan pengujian ke atas ALU 8 bit yang terdiri daripada 3 pin pilihan. Ini bermakna unit ini hanya mampu melakukan 8 operasi aritmetik dan 4 operasi logik. Jadual 1.1 menunjukkan operasi aritmetik dan logik yang boleh dilaksanakan oleh 8 bit unit aritmetik dan logik yang dibangunkan.

OPERASI ARITMETIK	OPERASI LOGIK
UMPUKAN	DAN
PENOKOKAN	ATAU
PENAMBAHAN	EKSKLUSIF-ATAU
PENAMBAHAN BERSAMA NILAI SAMBUTAN 1	TAK (PELENGKAP SATUAN)
PENAMBAHAN BERSAMA PELENGKAP SATUAN BAGI OPERAN KEDUA	
PENOLAKKAN	
PENGURANGAN	

Jadual 1.1 : Operasi Yang Dilaksanakan Oleh ALU 8 Bit Yang Dibangunkan

Seterusnya pembinaan BIST terdiri darpada komponen utama iaitu *Linear Feedback Shift Register* (LFSR) dan *Multiple Input Shift Register* (MISR). LFSR berfungsi sebagai penjana corak uji manakala MISR pula digunakan semasa proses analisis pengenalan (*signature analysis*) bagi litar BIST ALU tersebut. Pembinaan kedua-dua komponen ini berasaskan kepada *Built-In Logic Block Observer* (BILBO) yang bertindak sebagai pengawal. BILBO mampu beroperasi sebagai LFSR atau MISR berdasarkan kepada mod yang dipilih

Projek Latihan Ilmiah Tahap Akhir hanya dilaksanakan sehingga ke peringkat prototaip BIST ALU dengan menggunakan bahasa pengaturcaraan *VHSIC Hardware Description Language* (VHDL). Perlakuan BIST ALU akan dimodelkan kepada kod VHDL dan seterusnya disimulasikan. Bagi memastikan output yang diperolehi tepat, bahasa pengaturcaraan C++ telah digunakan bagi tujuan pengujian.

Secara keseluruhannya pembangunan BIST ALU ini terdiri daripada 3 sub-proses, iaitu:

- Pembangunan Unit Aritmetik dan Logik 8 bit yang terdiri daripada 3 pin pilihan
- Pembangunan BIST ALU yang menggunakan litar BILBO sebagai asas pembangunan LFSR dan MISR.
- Pengekodan C++ bagi tujuan analisis pengenalan.

1.3 OBJEKTIF PROJEK

Objektif utama bagi pembangunan BIST ALU ini adalah bagi merekabentuk prototaip BIST yang diuji ke atas unit aritmetik dan logik 8 bit sebagai unit pengujian dengan menggunakan bahasa pengaturcaraan VHDL semasa memodelkannya.

Selain itu, bahasa pengaturcaraan C++ digunakan bagi tujuan analisis pengenalan (*signature analysis*) bagi memastikan output yang diperolehi daripada proses simulasi tersebut adalah tepat.

BIST ALU merupakan titik permulaan bagi membangunkan projek-projek BIST yang lebih besar.

1.4 PENJADUALAN PROJEK.

Secara keseluruhannya proses pembangunan BIST ALU ini melibatkan gerak kerja selama lebih kurang 8 bulan (Rujuk Jadual 1.2). Kitar hayat pembangunan BIST ALU melibatkan 5 fasa iaitu : fasa penyiasatan awal, fasa analisis, fasa rekabentuk , fasa pembangunan dan implementasi dan fasa sokongan dan operasi.

PERKARA	JUN	JULAI	OGOS	SEPTEMBER	OKTOBER	NOVEMBER	DISEMBER	JANUARI
FASA 1:PENYIASATAN AWAL								
FASA 2:ANALISIS								
FASA 3:REKABENTUK								
FASA 4:PEMBANGUNAN DAN IMPLEMENTASI								
FASA 5:SOKONGAN DAN OPERASI								

Jadual 1.2 : Penjadualan Pembangunan BIST ALU

Semasa Fasa 1, kajian dilakukan ke atas tajuk yang dipilih. Semasa fasa ini perbincangan demi perbincangan dilakukan bersama penyelia penasihat bagi memastikan skop bagi projek ini bersesuaian dengan Projek Latihan Ilmiah Tahap Akhir.

Fasa 2 melibatkan kerja-kerja menganalisis tajuk projek yang telah dipilih. Dalam fasa ini beberapa cadangan rekabentuk BIST ALU akan diketengahkan. Sumber maklumat diperolehi daripada buku-buku yang berkaitan dengan tajuk, internet dan kajian projek-projek terdahulu. Selain itu semasa fasa ini juga perbincangan diadakan bersama penyelia penasihat bagi memastikan rujukan yang digunakan bersesuaian dengan tajuk.

Seterusnya fasa 3 melibatkan perancangan aktiviti-aktiviti yang diperlukan bagi membangunkan BIST ALU. Semasa fasa ini juga rekabentuk BIST ALU yang akan dibangunkan akan dikenalpasti. Ini merupakan susulan bagi fasa 2 dimana analisis secara terperinci dilaksanakan ke atas rekabentuk sebenar BIST ALU. Semasa fasa ini juga sesi *viva* dilakukan bersama penyelia penasihat dan moderator projek. Rekabentuk BIST ALU akan dibentangkan semasa sesi ini dan perbincangan akan dilakukan bersama penyelia penasihat dan moderator bagi memilih rekabentuk yang sesuai bagi BIST ALU.

Pembangunan fasa 4 dan fasa 5 dilaksanakan serentak. Kedua-dua fasa ini saling berkaitan antara satu sama lain dimana proses sokongan terhadap sistem dilaksanakan dari semasa ke semasa bagi memastikan sistem sentiasa beroperasi dengan baik dan bebas ralat (Rujuk Jadual 1.3).

PERKARA	SEPTEMBER	OKTOBER	NOVEMBER	DISEMBER	JANUARI
PEMBANGUNAN 8 BIT ALU					
PEMBANGUNAN BIST					
PENGGABUNGAN 8 BIT ALU DAN BIST					
PENGEKODAN C++					
SOKONGAN DAN OPERASI					

Jadual 1.3 : Aktiviti Fasa 3 dan Fasa 4

1.5 KEKANGAN

Pembinaan rekabentuk BIST ALU menggabungkan rekabentuk BIST bersama rekabentuk litar yang ingin diuji. Ini mengakibatkan saiz litar BIST ALU menjadi lebih besar berbanding saiz ALU normal, akibat pertambahan komponen BIST. Seterusnya, akibat pertambahan bilangan pin di dalam litar BIST ALU mengakibatkan masa lengahan bagi proses aritmetik dan logik menjadi semakin bertambah dan secara tidak langsung meningkatkan masa pemprosesan litar tersebut.

Rekabentuk BIST ALU dalam WXES 3182 adalah dibuat berdasarkankan teori semata-mata. Pelbagai implikasi yang mungkin berlaku semasa proses pelaksanaan dilaksanakan.

1.6 RINGKASAN BAB

Sebanyak empat bab telah buat bagi menyediakan laporan Latihan Ilmiah Tahap Akhir (WXES 3182). Setiap bab membincangkan isu-isu yang berbeza yang terlibat dalam pembangunan BIST ALU.

Bab 1 merupakan pendahuluan bagi keseluruhan laporan. Bab ini menyatakan skop dan objektif projek BIST ALU, kitar hayat pembangunan BIST ALU serta kekangan-kekangan yang dihadapi sepanjang proses pembangunan BIST ALU.

Bab 2 pula melaporkan hasil analisis dan kajian yang bagi setiap topik-topik yang terlibat dalam pembangunan BIST ALU. Dalam bab ini empat topik telah dikenalpasti iaitu; ALU, Penambah, DFT dan BIST.

Seterusnya bab 3 membincangkan metod-metod yang terlibat sepanjang proses pembangunan BIST ALU. Bab ini juga menyatakan pembahagian tugas yang dilakukan oleh penulis sepanjang proses pembangunan BIST ALU dilaksanakan.

Rekabentuk keseluruhan BIST ALU dinyatakan dalam bab 4. Bab ini menerangkan dengan terperinci rekabentuk BIST ALU dari peringkat terendah (unit) hingga ke peringkat yang lebih tinggi (sistem).

Bab 5 akan menghuraikan tentang implementasi pembangunan BIST ALU. Secara keseluruhannya bab ini merupakan bab yang terpenting kerana ia membincangkan tentang kaedah-kaedah dan teknik-teknik yang digunakan bagi membangunkan BIST ALU dengan lebih mendalam.

Setelah selesai mengaturcara BIST ALU dalam aturcara VHDL, seterusnya proses pengujian dilakukan. Kejayaan pembangunan BIST ALU bergantung kepada kelancarannya semasa simulasi dan hasilnya yang tepat. Proses pengujian ini akan membincangkan dengan lebih mendalam dalam bab 6.

Seterusnya bab 7 akan membincangkan secara menyeluruh tentang hasil yang diperolehi daripada pembangunan BIST ALU. Ia akan membincangkan tentang masalah dan kekangan yang terdapat di dalam BIST ALU, kekangan yang terdapat dalam sistem serta cadangan untuk perkembangan projek ini untuk masa hadapan. Akhir sekali bab 8 akan menyimpulkan secara keseluruhan projek BIST ALU.

1.7 KESIMPULAN

Projek BIST ALU ini merupakan aplikasi salah satu teknik rekabentuk kebolehujian yang diuji ke atas unit aritmetik dan logik 8 bit sebagai unit pengujian. Projek ini dilakukan sebagai titik permulaan bagi melaksanakan projek-projek pembangunan BIST yang lebih besar.

BAB 2

KAJIAN LITERASI

BAB 2

KAJIAN LITERASI

2.1 PENGENALAN

Bagi membangunkan *Built-In-Self-Test Arithmetic Logic Unit* (BIST ALU), kajian telah dilakukan ke atas komponen-komponen yang terlibat dalam pembangunan BIST ALU, projek-projek terdahulu dan artikel-artikel berkaitan. Analisis dilakukan ke atas empat topik utama yang terkandung dalam BIST ALU, iaitu:

- *Arithmetic Logic Unit* (ALU)
- Penambah (*Adder*)
- *Design For Testability* (DFT)
- *Built-In-Self-Test* (BIST)

Analisis dan kajian telah dibuat ke atas keempat-empat topik ini bagi mengenalpasti kaedah dan rekabentuk yang sesuai digunakan bagi membangunkan BIST ALU.

2.2.*ARITHMETIC LOGIC UNIT*

ALU merupakan satu litar bersepadu yang berfungsi untuk melaksanakan set-set operasi aritmetik dan logik yang terkandung di dalam mikropemproses sesebuah komputer. Secara keseluruhannya, kesemua elemen yang terkandung dalam sesebuah sistem komputer – unit kawalan, daftar, ingatan – akan menghantar data ke ALU untuk diproses dan seterusnya hasil operasi yang dilaksanakan di ALU akan dihantar semula ke komponen-komponen tersebut [WILL 00].

Secara umumnya, ALU merupakan satu komponen elektronik yang terdapat di dalam komputer yang bertugas untuk melaksanakan set-set arahan logik dalam bentuk digital. Ini bermakna, arahan-arahan yang digunakan di dalam ALU adalah dalam bentuk Boolean (0 dan 1).

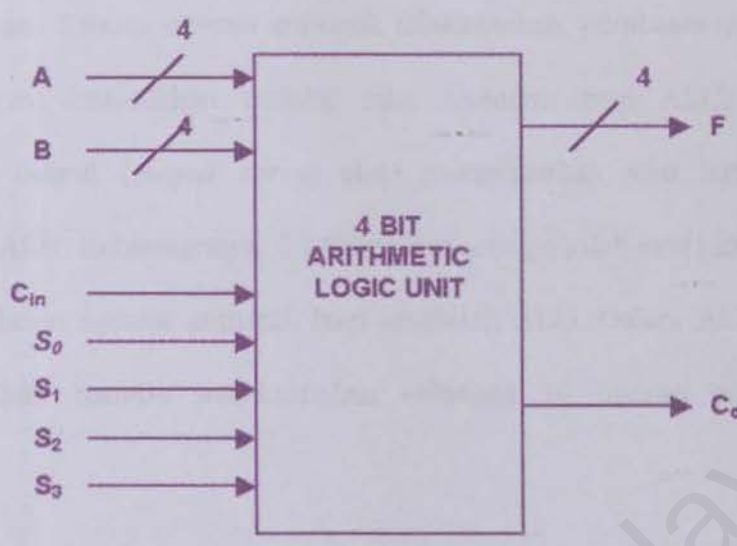
Rajah 2.1 menunjukkan rajah blok bagi ALU dan cara ALU berhubung dengan pemproses. Data-data yang diterima oleh ALU dihantar melalui daftar, manakala hasil pengiraan yang diperolehi oleh ALU juga akan disimpan di dalam daftar. Dalam konteks ini, daftar bertindak sebagai lokasi storan sementara yang terletak di antara pemproses dan ALU yang berfungsi untuk menghubungkan kedua-duanya dengan memberi isyarat laluan bagi data tersebut. ALU juga mengandungi set bendera yang berfungsi untuk mengeluarkan hasil bagi operasi yang dilakukan. Unit kawalan pula berfungsi untuk menyediakan isyarat bagi mengawal operasi dan pergerakkan data keluar-masuk melalui ALU.

Rajah 2.1 : Rajah blok Unit Aritmetik Dan Logik

2.2.1 REKABENTUK UNIT ARITMETIK DAN LOGIK

ALU melaksanakan operasi aritmetik dan logik dalam sesebuah komputer. Bilangan operasi aritmetik dan logik yang dapat dilaksanakan oleh sesebuah ALU bergantung kepada bilangan pin pilihan yang terdapat pada sesebuah ALU. Bilangan operasi yang mampu dilaksanakan ialah 2^n dimana n mewakili bilangan pin pilihan bagi ALU tersebut. Katakan, terdapat 5 pin pilihan dalam ALU *X*, maka bilangan operasi yang boleh dilakukan adalah 32 operasi; 16 operasi aritmetik dan 16 operasi logik [MORR 97].

Rajah 2.2 menunjukkan rajah blok bagi ALU 4 bit. Terdapat dua input data yang dimasukkan; 4 bit input dimasukkan melalui pin A manakala 4 bit berikutnya dimasukkan melalui pin B. Hasil bagi operasi aritmetik atau logik yang dilaksanakan akan dikeluarkan melalui pin F.



Rajah 2.2 : Rajah Kotak Hitam Bagi Aritmetik Dan Logik 4 bit

Pin S_3 beroperasi bagi menentukan mod operasi yang bakal dilaksanakan oleh ALU samada operasi aritmetik atau operasi logik. Sekiranya $S_3 = 0$ ini bermakna mod logik dipilih, maka ALU akan melaksanakan operasi logik. Manakala sekiranya $S_3 = 1$, bermaksud mod aritmetik dipilih dan ALU akan melaksanakan operasi aritmetik.

Pin S_2, S_1 dan S_0 pula bertindak sebagai penentu operasi aritmetik atau logik yang bakal dilaksanakan. Ini menunjukkan ALU 4 bit ini mampu melaksanakan 16 operasi iaitu 8 operasi aritmetik dan 8 operasi logik (Rujuk Jadual 2.1). Gabungan Pin S_0, S_1, S_2 dan S_3 membentuk kod operasi (opkod). Opkod yang diperolehi akan menentukan jenis operasi yang akan dilaksanakan oleh ALU.

Pin C_{in} dan C_{out} pula hanya digunakan semasa operasi aritmetik dilaksanakan. Semasa operasi aritmetik dilaksanakan, pembawa input (*input carry*) akan dimasukkan melalui nilai terendah bagi ALU, manakala pembawa output (*output carry*) akan mengeluarkan nilai tertinggi bagi sesebuah ALU. Kebiasaannya, C_{in} digunakan sebagai salah satu pin yang akan menggandakan operasi aritmetik bagi sesebuah ALU. Dalam ALU 4 bit ini menunjukkan mampu melaksanakan sehingga 16 operasi aritmetik ke atasnya.

Secara umumnya rekabentuk ALU terdiri daripada tiga peringkat. Peringkat pertama adalah rekabentuk unit aritmetik. Seterusnya rekabentuk unit logik pula akan dilaksanakan. Bagi melaksanakan ALU yang lengkap, kedua-dua rekabentuk aritmetik dan logik ini akan digabungkan dalam satu litar (Rujuk Rajah 2.3) [MORR 79].

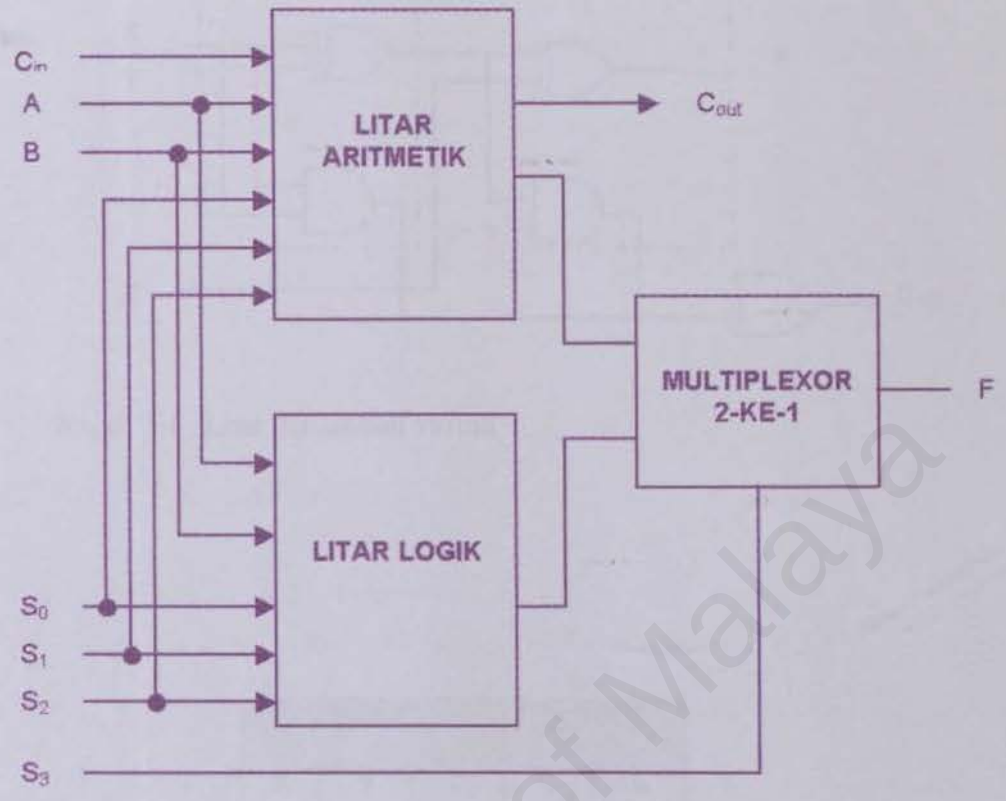
2.3 PENAMBAH (*ADDER*)

Litar aritmetik merupakan litar bersepadu yang melaksanakan operasi penambahan, penolakan, pendaraban dan pembahagian ke atas nombor binari atau nombor desimal dalam bentuk kod binari. Pembinaan rekabentuk litar aritmetik dibuat secara berhirarki. Asas bagi pembinaan sesebuah litar aritmetik adalah penambah penuh yang hanya mampu melaksanakan operasi penambahan (Rujuk Rajah 2.4). Hasil yang diperolehi dari penambah penuh

merupakan hasil manipulasi terhadap dua penambah separuh dan get ATAU[MORR 97].

OPKOD	OPERASI	PENERANGAN
0000	NOP	Tiada Operasi
0001	ADD	Penambahan Ke Atas Dua 4 Bit Input
0010	SUB	Penolakkan Ke Atas Dua 4 Bit Input
0011	MULT	Pendaraban Ke Atas Dua 4 Bit Input
0100	NOT	Sonsangan Ke Atas Nilai 4 Bit Input
0101	AND	Logik DAN Ke Atas 4 Bit Input
0110	OR	Logik ATAU Ke Atas 4 Bit Input
0111	XOR	Logik XOR Ke Atas 4 Bit Input
1000	ASL	Anjakan Aritmetik Ke Kiri Bagi 4 Bit Input
1001	LSL	Anjakan Logik Ke Kiri Bagi 4 Bit Input
1010	ASR	Anjakan Aritmetik Ke Kanan Bagi 4 Bit Input
1011	LSR	Anjakan Logik Ke Kanan Bagi 4 Bit Input
1100	ROTL	Bergerak (Rotate) Ke Kiri Bagi 4 Bit Input
1101	ROTR	Bergerak (Rotate) Ke Kanan Bagi 4 Bit Input
1110	NOP	Tiada Operasi
1111	NOP	Tiada Operasi

Jadual 2.1 : Contoh Operasi Aritmetik Bagi Unit Aritmetik Dan Logik 4 Bit

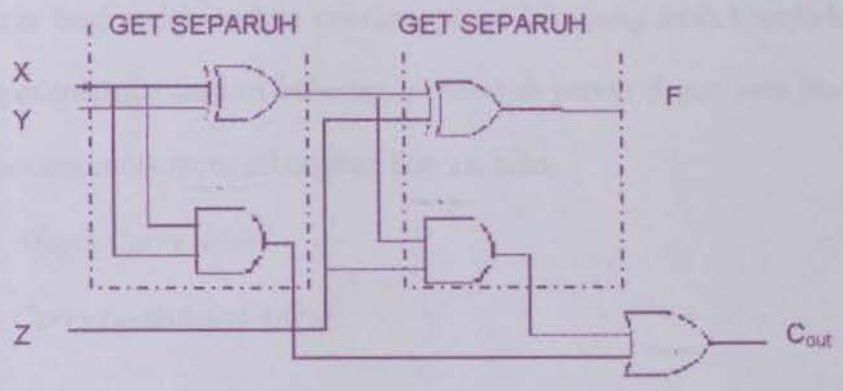


Rajah 2.3 : Gabungan Unit Aritmetik Dan Unit Logik Bagi Membentuk Unit Aritmetik Dan Logik

Penambah penuh melakukan operasi ke atas tiga bit input. Dua dari input tersebut mewakili nilai yang ingin ditambah. Manakala input ketiga mewakili nilai pembawa daripada operasi terdahulu. Terdapat dua output bagi penambah penuh yang membawa nilai antara 0 hingga 3 (2^2) (Rujuk Jadual 2.2). Output S mewakili hasil yang diperolehi daripada penambahan dua operan manakala C pula mewakili nilai pembawa bagi penambahan tersebut. Secara umumnya, nilai output bagi penambah penuh boleh diperolehi dengan melakukan operasi eksklusif-ATAU terhadap input sebanyak dua kali, iaitu:

$$S = (X \oplus Y) \oplus Z$$

$$C = XY + Z(X \oplus Y)$$



Rajah 2.4 : Litar Penambah Penuh

INPUT			OUTPUT	
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Jadual 2.2 : Jadual Kebenaran Bagi Penambah Penuh

Seterusnya bagi melaksanakan operasi-operasi lain yang lebih kompleks adalah dengan menggabungkan beberapa penambah penuh dalam satu litar. Terdapat dua cara untuk menggabungkan litar ini, iaitu:

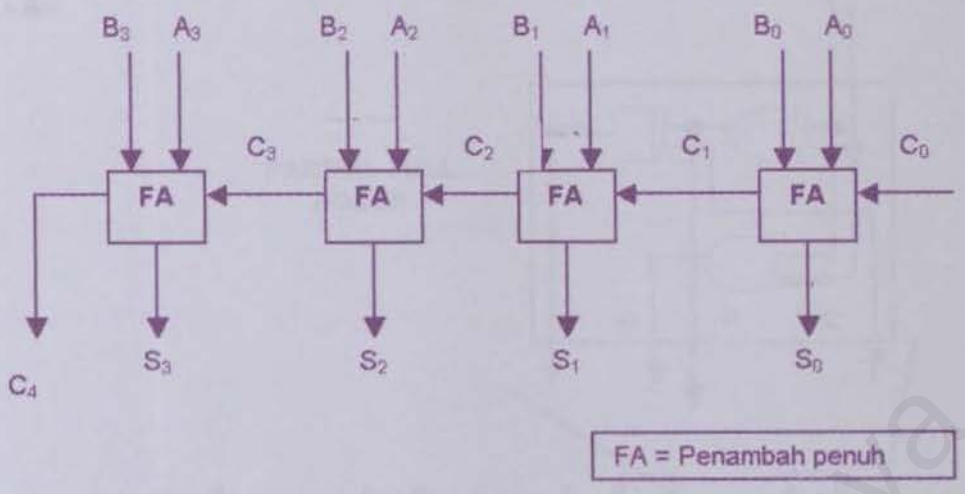
- *Ripple Carry Adder*
- *Carry Lookahead Adder*.

2.3.1 RIPPLE CARRY ADDER

Ripple Carry Adder juga dikenali sebagai penambah selari merupakan litar digital yang menghasilkan hasil bagi operasi aritmetik bagi dua nombor binari dengan menggunakan litar kombinasi. Penambah selari menggunakan n penambah penuh yang disusun secara selari. Seterusnya input akan dimasukkan secara berjujukan untuk mendapatkan hasil.

Dalam penambah selari penambah penuh ditambah secara berterusan (*cascade*)[MORR 97], dimana nilai pembawa daripada penambah penuh terdahulu dimasukkan ke dalam penuh yang berikutnya (Rujuk Rajah 2.5).

Bilangan penambah penuh yang digunakan merujuk kepada bilangan bit penambah selari yang ini digunakan. Ini bermakna semakin banyak bit, semakin banyak bilangan penambah penuh yang diperlukan.

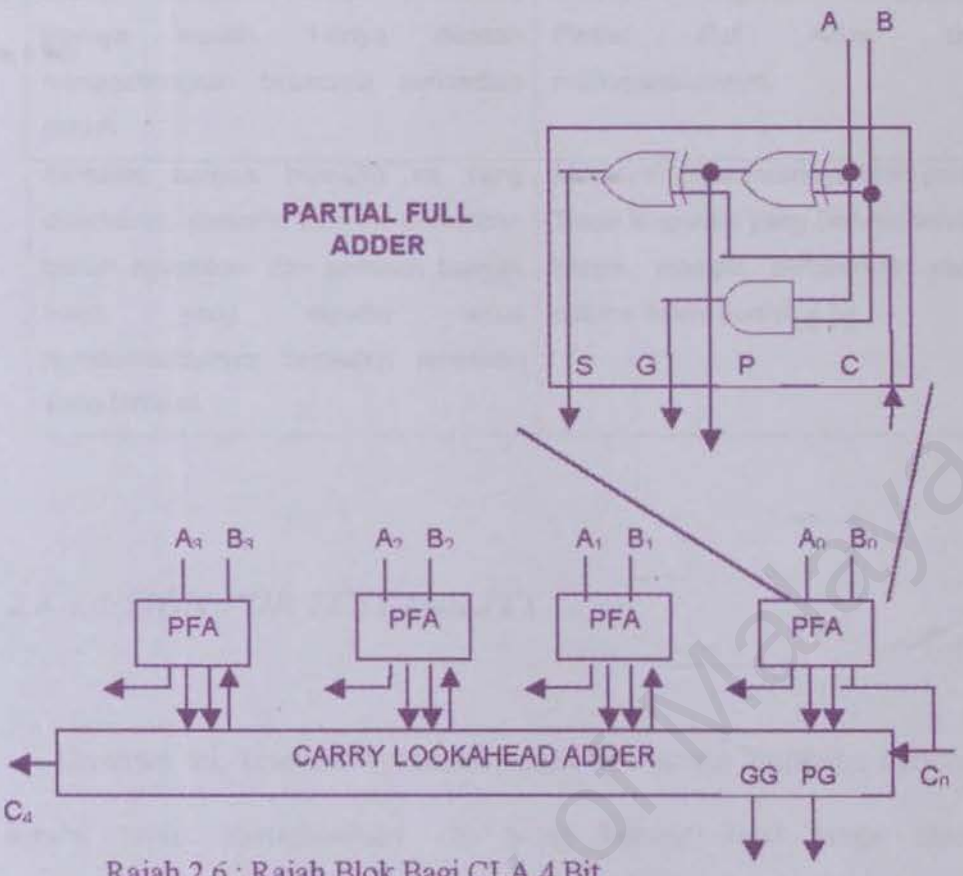


Rajah 2.5 : Penambah Selari 4 Bit

2.3.2 CARRY LOOKAHEAD ADDER

Carry Lookehead Adder (CLA) merupakan salah satu teknik yang digunakan bagi mengurangkan masa lengahan dalam operasi aritmetik. Matlamat CLA adalah bagi melakukan operasi penambahan dengan lebih cepat CLA menggunakan *Partial Full Adder* (PFA) bagi melaksanakan operasi penambahan (Rujuk Rajah 2.6). Dengan ini masa pemprosesan menjadi lebih cepat dengan mengurangkan tempoh lengahan tetapi litar yang digunakan adalah lebih kompleks.

CLA dapat beroperasi secara optima apabila penambahan 4 bit dilakukan. Bagi penambahan yang melibatkan lebih daripada 4 bit, beberapa kumpulan 4 bit CLA perlulah digabungkan bagi memastikan ia dapat beroperasi dengan baik.



Rajah 2.6 : Rajah Blok Bagi CLA 4 Bit

2.3.3 RIPPLE CARRY ADDER VS CARRY LOOKAHEAD ADDER

Secara umumnya, *Ripple Carry Adder* dan *Carry Lookahead Adder* berfungsi untuk melakukan operasi penambahan. Namun begitu terdapat beberapa perbezaan antara kedua-dua teknik ini, iaitu:

RIPPLE CARRY ADDER	*CARRY LOOKAHEAD ADDER*
Operasi penambahan yang dilakukan bergantung kepada hasil operasi penambahan terdahulu	Operasi penambahan tidak bergantung kepada hasil operasi terdahulu

Mudah untuk diimplementasikan, litarnya mudah. Hanya dengan menggabungkan beberapa penambah penuh.	Litarnya kompleks. Menggunakan *Partial Full Adder* untuk melaksanakannya.
Semakin banyak bilangan bit yang diperlukan, semakin banyak penambah penuh diperlukan dan semakin banyak masa yang diambil untuk melaksanakannya berikutan lengahan yang terhasil.	Masa pengoperasiannya lebih pantas. Tiada lengahan yang berlaku tetapi ia hanya mampu beroperasi secara optima dalam bentuk 4 bit.

2.4 DESIGN FOR TESTABILITY

Dizaman ini, kesemua syarikat-syarikat komponen komputer berlumba-lumba untuk menghasilkan cip yang bersaiz kecil tetapi mampu melaksanakan pelbagai fungsi. Keperluan utama yang menjadi ukuran jaminan kualiti sesuatu keluaran litar bersepadu ialah prestasi, kebolehpercayaan dan kos. Bagi menghasilkan cip yang mempu melaksanakan pelbagai fungsi, beratus-ratus get logik diperlukan bagi memastikan ianya berkualiti tinggi. Semakin banyak get logik yang digunakan, semakin kompleks sesuatu litar tersebut dan semakin sukar untuk memastikan ia beroperasi dengan cekap.

Oleh sebab itu, tidak mustahil terdapat kekurangan atau kecacatan pada sesuatu litar bersepadu. Kecacatan atau kekurangan yang timbul mungkin berpunca semasa proses pembuatan atau semasa proses pengurusan dilaksanakan. Terdapat dua pendekatan yang boleh digunakan bagi menguji

litar bersepadu iaitu melalui ujian sambutan logik atau ujian berparameter [ZAHA 94].

Ujian sambutan logik digunakan bagi mengesahkan litar bersepadu tersebut beroperasi seperti yang terkandung dalam jadual kebenaran bagi litar tersebut. Biasanya ralat seperti ini berlaku akibat kegagalan rekabentuk fizikal atau terdapat sebarang kecacatan fizikal semasa proses pembuatan cip tersebut. Ujian sambutan logik juga digunakan bagi mengenalpasti sesuatu kegagalan dan punca kegagalan tersebut. Dalam konteks ini, kegagalan merujuk kepada sebarang kecacatan yang mencegah litar bersepadu itu berfungsi seperti yang dikehendaki.

Ujian berparameter pula digunakan bagi mengesahkan parameter tertentu dalam sebuah litar bersepadu mengikut julat yang telah ditetapkan. Antara parameter yang penting dalam sesebuah litar bersepadu adalah masa pensuisan, arus dan voltan kendalian get logik.

Terdapat dua peringkat bagi pengujian litar yang biasa dilakukan, peringkat pertama ialah penjanaan vektor ujian. Pada peringkat ini satu jujukan corak masukan dikenakan pada masukkan utama. Untuk suatu litar digit, suatu set corak masukan yang mengandungi banyak vektor ujian diperlukan bagi mengesan semua kegagalan yang terdapat dalam litar berkenaan. Peringkat seterusnya dikenali sebagai penilaian sambutan ujian. Ia merupakan proses bagi mengesahkan sambungan litar gabungan yang diuji

dalam satu set ujian tertentu terhadap sambungan sesuatu litar adalah bebas ralat.

2.4.1 FAKTOR YANG MENDORONG DFT

Pengujian litar bersepadu menjadi bertambah rumit apabila kekompleksan sesuatu rekabentuk itu bertambah dan bilangan get setiap pin lebih tinggi, iaitu apabila rangkaian dalaman sukar dimasuki oleh rangkaian luaran. Rekabentuk kebolehujian merupakan salah satu alternatif yang digunakan bagi menguji litar yang dibangunkan pada tahap-tahap tertantu.

Kos pengujian bagi sesuatu litar bersepadu semakin bertambah dari satu peringkat ke peringkat seterusnya sebanyak 10 kali ganda (Rujuk Jadual 2.3). Dengan menggunakan teknik rekabentuk kebolehujian, sebarang ralat atau kecacatan yang terkandung dalam sesebuah litar dapat dikesan pada peringkat awal dan ini dapat menjimatkan kos pengujian dan pembangunan sesebuah litar bersepadu.

Bagi menghasilkan litar bersepadu yang mengandungi bilangan get yang banyak dan kompleks dalam satu cip yang kecil bukanlah satu proses yang mudah. Banyak isu-isu yang perlu diambilkira bagi memastikan litar tersebut dapat beroperasi dengan baik. Antara isu yang menjadi topik utama

RACIK (BIT SLICE)	PAPAN LITAR (BOARD)	SISTEM	PELANGGAN
t	10t	100t	1000t

T = Ringgit Malaysia

Jadual 2.3 : Kos bagi mengenalpasti dan mendiagnosis ralat pada pelbagai peringkat [IBRA 95]

dalam rekabentuk kebolehujian ialah pemilihan kaedah rekabentuk kebolehujian bagi sesebuah litar. Biasanya pemilihan sesuatu kaedah rekabentuk kebolehujian adalah bergantung kepada masa pelaksanaan yang diambil bagi sesuatu kaedah itu melengkapkan proses pengujian tersebut. Seterusnya, perkara-perkara yang mempengaruhi masa pemproses sesebuah litar bersepadu akan dibincangkan

Pengujian sesuatu litar yang mengandungi bilangan pin masukan yang banyak akan mengambil masa yang lama. Sesuatu litar logik bersepadu yang mengandungi n bilangan masukan memerlukan 2^n corak masukan yang membolehkan ia diuji dengan lengkap. Bagi litar berjujukan pula, m bit unsur berjujukan memerlukan 2^m gabungan keadaan untuk membolehkan ia diuji dengan lengkap. Ini bermakna, bagi sesuatu litar bersepadu yang mengandungi n bilangan masukan dan m bilangan unsur berjujukan, ia memerlukan 2^{n+m} vektor ujian untuk memastikan litar tersebut lengkap diuji [IBRA 95]. Dalam kes ini, katakan litar bersepadu *X* mempunyai 20 bilangan masukan dan 16 unsur berjujukan, litar tersebut memerlukan $2^{20+16} = 2^{36} = 6.87 \times 10^{10}$ vektor ujian. Sekiranya litar berkenaan diuji pada kelajuan jam 10

x 10^6 Hz (50 ns setiap bit), masa yang diperlukan bagi proses pengujian ialah 1 jam 54.53 minit. Oleh itu, masa yang diambil bagi melakukan prose pengujian sesebuah litar adalah agak lama walaupun litar tersebut agak mudah.

Kaedah pengujan yang baik perlu memastikan liputan kegagalan yang tingi terhadap setiap pengujian. Liputan kegagalan ditakrifkan sebagai nisbah bilangan kegagalan yang dikesan kepada jumlah bilangan kegagalan yang mungkin, iaitu:

$$\text{Liputan kegagalan} = \frac{\text{Bilangan kegagalan yang dikesan}}{\text{Bilangan kegagalan yang mungkin}} \times 100\%$$

Biasanya, liputan kawalan melebihi 95% diperlukan untuk kaedah pengujian yang baik [ZAHA 94].

2.4.2.PRINSIP DAN MATLAMAT REKABENTUK DFT

Matlamat utama rekabentuk kebolehujian adalah untuk meninggikan mutu dan kebolehpercayaan bagi setiap keluaran disamping mengurangkan kos [ZAHA94]. Suatu rekabentuk kebolehujian yang baik perlu memenuhi ciri-ciri berikut:

- Tokokan liputan kegagalan (pengesanan dan penempatan)
- Kemudahan mengenakan ujian
- Pengurangan masa ujian
- Keboleh ujian pada kelajuan

- Penjanaan corak uji yang mudah
- Penilai sambuatan yang boleh dipercayai, mudah dan mempunyai overhed perkakasan yang mudah.

Dua parameter penting untuk Rekabentuk Kebolehujian ialah kebolehkawalan dan kebolehcerapan [IBRA 95]. Kebolehkawalan merujuk kepada kebolehan meletakkan suatu litar dalaman kepada suatu keadaan yang diketahui, iaitu kebolehan mengenakan ujian. Kebolehcerapan pula merupakan kebolehan melihat atau memacu sambutan litar-litar dalaman pada keluaran utama.

2.4.3 KAEDAH DFT

Kaedah rekabentuk kebolehujian dikategorikan kepada empat kelas seperti berikut:

- Kaedah *Ad Hoc*
- Kaedah rekabentuk berstruktur
- Kaedah *Built-In-Self-Test* (BIST)
- Kaedah repabentuk imbas sempadan (*Boundary Scan*)

Kaedah *Ad Hoc* teknik rekabentuk tidak piawai. Ia digunakan untuk menyelesaikan masalah khusus untuk sesuatu rekabentuk sahaja dan tidak dapat digunakan oleh rekabentuk yang lain. Kaedah ini biasanya menyelesaikan masalah pada luaran yang tidak memerlukan perubahan pada

rekabentuk logik bagi menyelesaikannya. Kaedah *Ad Hoc* menambah kebolehcerapan dan kebolehkawalan bagi sesuatu litar dan cuba mengatasi masalah tersebut dengan menggunakan tiga teknik iaitu, penambahan titik ujian, pemetakan logik dan senibina bas. Biasanya jurutera rekabentuk yang menggunakan kaedah ini akan mengaplikasikan pengalaman mereka untuk menyusun semula sesuatu rekabentuk litar supaya mudah diuji dan tidak bergantung kepada mana-mana peraturan rekabentuk.

Rekabentuk Berstruktur biasanya digunakan untuk mengurangkan kekompleksan penjanaan ujian dan penyelakuan kegagalan kepada suatu peringkat keadaan yang munasabah. Kaedah ini cuba menyelesaikan masalah melalui metodologi rekabentuk. Ia melibatkan set peraturan bagi melaksanakan rekabentuk. Kaedah dalam kategori ini adalah seperti berikut:

- Laluan imbas (*Scan Path*)
- Rekabentuk *Level Sensitive Scan Design*
- Logic set imbas (*Scan/Set Logic*)
- Imbas Capaian Rawak (*Random Access Scan*)
- *Boundary Scan*

Kaedah *Built-In-Self Test* menggunakan penjana vektor uji dan penilai sambutan di atas sekeping racik (*bit slice*). Pada amnya penjana corak pseudorawak digunakan bagi tujuan ini. Penjana corak yang digunakan secara meluas ialah *Linear Feedback Shift Register* (LFSR). Litar yang sama juga boleh digunakan sebagai sebuah penilai sambutan dan penilai sambutan LFSR dikenali sebagai penganalisis pengenalan.

Rekabentuk imbas sempadan pula bertujuan untuk menambahkan kebolehujian pada peringkat papan litar. Rekabentuk ini memberikan suatu kemudahan untuk menguji litar bersepadu di atas papan litar tanpa memerlukan peralatan ujian dalaman.

2.5 BUILT-IN-SELF-TEST

Built-In-Self-Test (BIST) merupakan satu set teknik pengujian yang digunakan bagi menguji litar bersepadu dengan menyatukan litar ujian dan litar penguji di atas sekeping racik. Ini dilakukan bagi mengurangkan pergantungan litar yang direkabentuk kepada penguji dan memperoleh kemudahan ujian pada kelajuan operasi sistem dengan liputan kegagalan yang tinggi. BIST menyatukan penjana corak ujian dan pemampatan data di atas racik yang sama. Kaedah ini meliputi tiga fungsi yang nerupakan sebahagian daripada sistem, iaitu;

- Penjanaan vektor ujian (*Psuedo-random Pattern Generator*)
- Penggunaan vektor ujian
- Analisis pengenalan (*Signature Analysis*)

Rajah 2.7 menunjukan gambaran umum kaedah pengujian menggunakan konsep BIST. Apabila mod ujian dipilih oleh isyarat pemilih ujian (*test-select*), penjana uji akan mengaplikasikan corak ujian yang dihasilkan ke atas litar yang sedang diuji. Hasil yang diperolehi akan dicerap dan dianalisis bagi mengenalpasti sebarang kesilapan atau ralat yang terdapat di dalam sistem.

Rajah 2.7 : Konsep Pengujian BIST [IBRA 95]

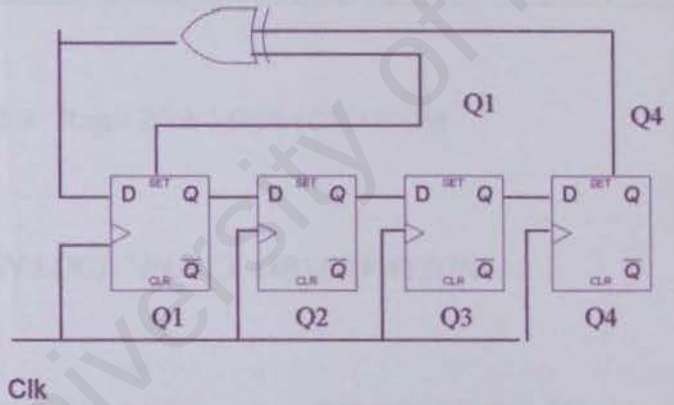
2.5.1 LINEAR FEEDBACK SHIFT REGISTER

Linear Feedback Shift Register (LFSR) biasanya digunakan untuk menjana corak ujian . Output bagi flip-flop pertama dan keempat akan diXOR dan hasilnya akan disuap balik bagi menjadi input kepada flip-flop D yang pertama bagi litar tersebut (Rujuk Rajah 2.8). Konsep asas LFSR ialah daftar anjakan ke atas hasil XOR bagi dua atau lebih flip-flop dan kemudian disuap balik kepada flip-flop pertama.

Bilangan corak uji yang mampu dijana oleh setiap LFSR adalah bergantung kepada bilangan bit daftar anjakan yang digunakan. Bagi setiap n bit daftar anjakan 2^n-1 bit corak akan dijana. Pelbagai corak akan dihasilkan hasil kombinasi keempat-empat output bagi daftar anjakan kecuali nilai semua '0'. Jadual 2.4 menunjukkan kombinasi suap balik yang akan dijana bagi kesemua 2^n-1 bit corak bagi LFSR sekitar purata n = 4 hingga 32.

n	Suap Balik
4, 6, 7	$Q_1 \oplus Q_n$
5	$Q_2 \oplus Q_5$
8	$Q_2 \oplus Q_3 \oplus Q_4 \oplus Q_8$
12	$Q_1 \oplus Q_4 \oplus Q_6 \oplus Q_{12}$
14, 16	$Q_3 \oplus Q_4 \oplus Q_5 \oplus Q_n$
24	$Q_1 \oplus Q_2 \oplus Q_7 \oplus Q_{24}$
32	$Q_1 \oplus Q_2 \oplus Q_{22} \oplus Q_{32}$

Jadual 2.4 : Suap Balik bagi Urutan Kepanjangan Maksimum LFSR

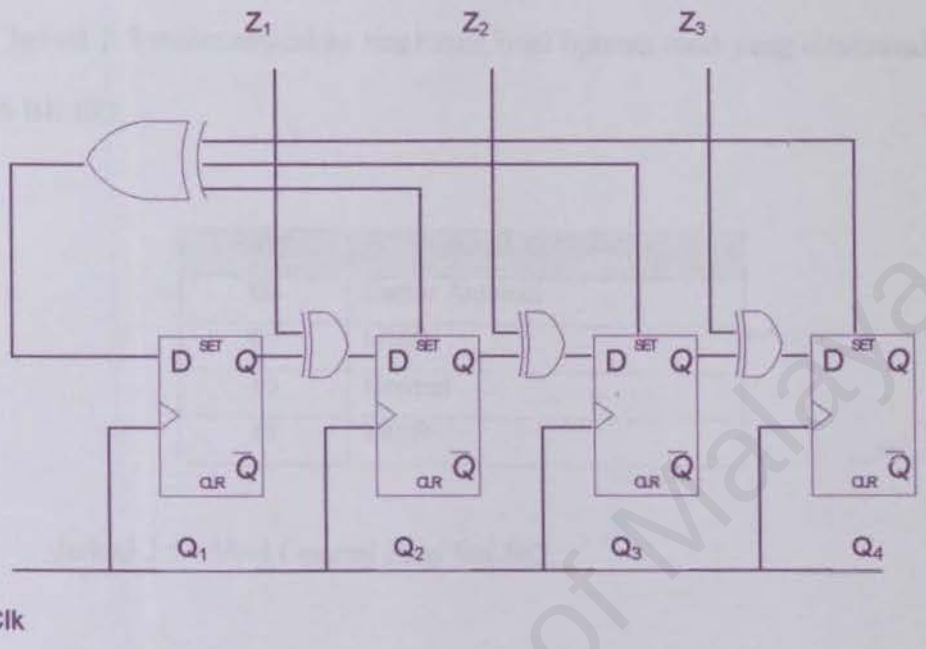


Rajah 2.8 : Rajah Blok LFSR

2.5.2 MILTIPLE INPUT SHIFT REGISTER

Multiple Input Shift Register (MISR) boleh dibina dengan mengubahsuai rekabentuk LFSR dengan menambah get XOR (Rujuk Rajah 2.9). Data yang diuji (Z1, Z2, Z3, Z4) akan diXORkan bersama ke dalam daftar bagi setiap

jam dan keputusan terakhir bagi proses ini akan mewakili satu pengenalan yang akan dibandingkan dengan pengenalan bagi komponen yang bebas ralat.



Rajah 2.9 : Rajah Blok MISR [CHAR 96]

2.5.3BUILT-IN LOGIC BLOCK OBSERVATION

Built-In Logic Block Observation (BILBO) [CHAR 96] merupakan daftar yang berfungsi sebagai unit kawalan yang yang mengawal mod operasi yang berlaku dalam BIST. Dengan menggunakan BILBO, fungsi setiap unit boleh diubahsuai mengikut keperluan sama ada untuk membentuk mod LFSR, MISR, normal atau daftar anjakan. Input B_1 dan B_2 bertugas sebagai input kawalan yang mengawal mod operasi bagi BILBO berkenaan. S_i dan S_o berfungsi sebagai masukan dan keluaran sesiri bagi mod daftar anjakan. Z_s pula merupakan input dari logik kombinasi. Persamaan bagi daftar BILBO adalah:

$D_1 = Z_1B_1 \oplus (S_iB_2' + FB\ B_2)(B_1' + B_2)$

$D_i = Z_iB_1 \oplus Q_{i-1}(B_1' + B_2)\ (I > 1)$

Jadual 2.5 menununjukkan ringkasan bagi operasi mod yang dilaksanakan oleh BILBO.

B1B2	MOD OPERASI
00	Daftar Anjakan
01	LFSR
10	Normal
11	MISR

Jadual 2.5 : Mod Operasi Bagi BILBO.

3.5 KESIMPULAN

Empat topik utama yang perlu diambil kira sepanjang proses menganalisis BIST ALU ialah; Unit Aritmetik dan Logik, Penambah, Rekabentuk Kebolehujian dan *Built-In-Self-Test*. Hasil daripada kajian yang dilakukan ke atas keempat-empat topik ini menentukan kaedah dan rekabentuk yang sesuai bagi BIST ALU yang akan dibincangkan dalam bab 4.

BAB 3

METODOLOGI

BAB 3

METODOLOGI

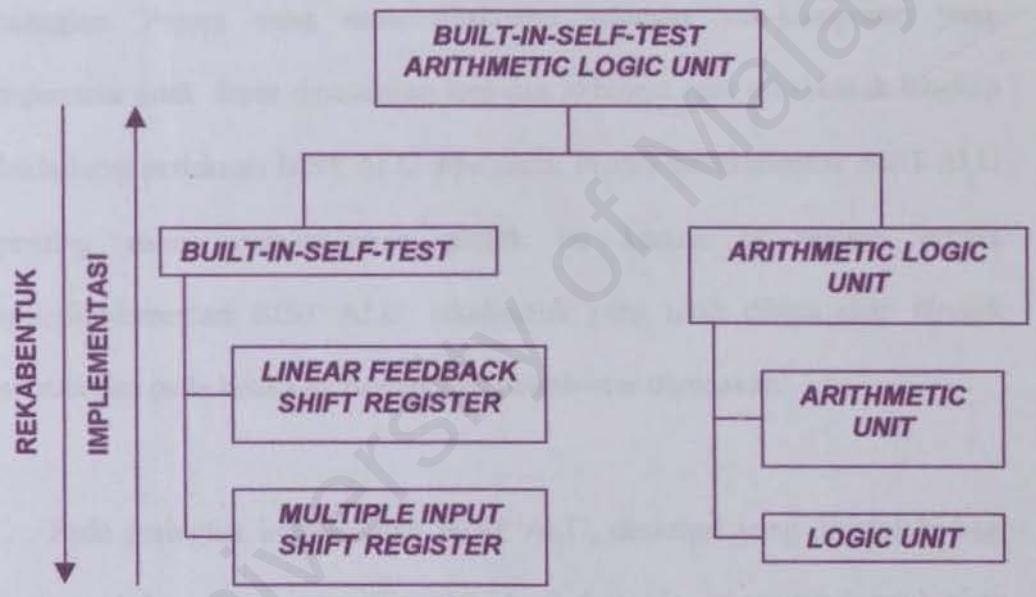
3.1 PENGENALAN

Masa merupakan salah satu faktor yang perlu diambil kira dalam pembangunan sesebuah projek. Selain itu, pembangun projek juga perlu memastikan projek yang dibangunkan dilaksanakan dengan cara yang paling mudah dan mendapat hasil yang optima. Kawalan prestasi juga perlu dilakukan dari semasa ke semasa bagi memastikan projek yang dibangunkan sentiasa berlandaskan spesifikasi yang telah ditetapkan pada awal projek.

Bagi pembangunan *Built-In-Self-Test Arithmetic Logic Unit* (BIST ALU), dua subjek utama telah dikenalpasti bagi memastikan proses pembangunan BIST ALU sentiasa berjalan lancar, siap dalam tempoh yang ditetapkan dan mengikut spesifikasi yang telah ditetapkan. Kedua-dua subjek ini; Pendekatan Pembangunan BIST ALU dan Kaedah Implementasi BIST ALU sentiasa bersandaran antara satu sama lain bagi memastikan proses pembangunan BIST ALU dapat dilaksanakan dengan lebih sistematik.

3.2 PENDEKATAN PEMBANGUNAN BIST ALU

BIST ALU dibangunkan dengan menggunakan dua pendekatan, iaitu; pendekatan pembangunan atas-bawah bagi membentuk rekabentuk bagi BIST ALU manakala bagi memodulkan BIST ALU untuk disimulasi, pendekatan pembangunan bawah-atas digunakan (Rujuk Rajah 3.1).



Rajah 3.1 : Pendekatan Bagi Membangunkan BIST ALU

Pendekatan atas-bawah digunakan bagi merekabentuk BIST ALU. Pendekatan ini digunakan kerana ia dapat membahagikan BIST ALU kepada sub-sub komponen yang lebih kecil. Proses ini akan memudahkan aktiviti merekabentuk BIST ALU dimana ia dilakukan daripada draf kasar sehingga akhirnya diperincikan.

Pada awal pembangunan BIST ALU, rajah kotak hitam BIST ALU telah dihasilkan. Rajah kotak hitam ini menggambarkan secara umum rekabentuk BIST ALU yang dibangunkan. Ia menyatakan bilangan input dan output yang diperlukan bagi BIST ALU, ia juga menyatakan semua pin-pin yang terlibat dalam pembangunan BIST ALU seperti C_{in} dan C_{out}.

Selepas mengenalpasti item-item yang terlibat dalam kotak hitam, rekabentuk BIST ALU diperhalusi dengan memecahkannya kepada beberapa bahagian. Proses yang sama dilakukan sehingga sub-komponen yang diperolehi tidak dapat dipecahkan lagi dan akhirnya satu rekabentuk lengkap berhubung perlakuan BIST ALU diperolehi. Proses merekabentuk BIST ALU penting dalam pembangunan projek ini kerana ia semasa proses mengimplementasi BIST ALU, rekabentuk yang telah dibina akan dirujuk semula dan pada ketika ini pendekatan bawah-atas digunakan.

Pada peringkat implementasi BIST ALU, deskripsi yang diperolehi bagi setiap sub-komponen yang diperolehi hasil daripada proses rekabentuk akan dikodkan dalam bahasa pengaturcaraan *VHSIC Hardware Description Language (VHDL)*. Seterusnya keseluruhan deskripsi bagi sub-komponen tersebut akan digabungkan membentuk satu kod lengkap yang memodulkan BIST ALU.

3.3 KAEDAH IMPLEMENTASI BIST ALU

Pembinaan BIST ALU melibatkan aplikasi dua jenis bahasa pengaturcaraan iaitu *VHSIC Hardware Description Language* (VHDL) dan C++. VHDL digunakan untuk mendiskripsikan perlakuan BIST ALU dan mendapatkan prototaip bagi BIST ALU. Seterusnya C++ digunakan semasa proses analisis pengenalan (*signature analysis*).

3.3.1 VHDL

VHDL merupakan singkatan daripada perkataan *Very High Speed Integrated Circuit (VHSIC) Hardware Description Language* merupakan bahasa pengaturcaraan yang digunakan untuk mendiskripsikan sistem elektronik digital. VHDL mula diperkenalkan oleh Kerajaan United States pada tahun 1980. Ia diperkenalkan berikutan perkembangan mendadak litar bersepadu dan kesedaran tentang keperluan satu bahasa piawai bagi mendiskripsikan struktur dan fungsi sesebuah litar bersepadu.

Terdapat beberapa kelebihan VHDL yang telah dikenalpasti [PETE 96], antaranya:

- VHDL mampu mendiskripsikan keseluruhan rekabentuk sistem elektronik digital yang dibangunkan. Bagi mendiskripsikan sesebuah rekabentuk elektronik digital, pembangun perlu memahami perlakuan terperinci bagi sistem elektronik digital yang

dibangunkan. Seterusnya mengekodkan deskripsi tersebut dalam bahasa pengaturcaraan VHDL.

- Bahasa pengaturcaraan VHDL mudah untuk diimplementasikan. Konsep asas bagi VHDL sama seperti bahasa pengaturcaraan yang biasa digunakan seperti C++ dan PASCAL, dengan ini masa yang diambil untuk mempelajari VHDL adalah lebih singkat.

- VHDL membenarkan penggunaan semula komponen-komponen yang telah didiskripsikan dalam rekabentuk yang berbeza. Ini mengurangkan kebersandaran antara komponen-komponen yang terkandung dalam sistem tersebut.

- VHDL membenarkan rekabentuk yang dibangunkan disimulasi terlebih dahulu sebelum dibangunkan. Dengan ini sistem elektronik digital yang dibangunkan dapat diuji terlebih dahulu dan segala kelemahan dan ralat yang terdapat di dalam rekabentuk sistem yang dibangunkan dapat diselesaikan terlebih dahulu. Ini dapat mengurangkan masa lengahan dan pembaziran kos pembangunan projek berbanding membangunkan prototaip bagi sistem yang dibangunkan.

3.3.2 C++

Bahasa pengaturcaraan C++ telah dibangunkan oleh Bjarne Stroustrop bagi memperbaiki kelemahan yang terkandung dalam bahasa pengaturcaraan C. C++ merupakan salah satu bahasa pengaturcaraan yang sangat berkuasa, serbaguna dan fleksible untuk membina sesebuah perisian [SELL 99]. Ia sangat sesuai untuk membina perisian sistem seperti sistem pengoperasian, pengkompil , editor , pangkalan data dan komunikasi data.

Bahasa pengaturcaraan C++ bersifat ramah pengguna kerana ia menggunakan konsep berorientasikan objek. Ia juga menggunakan pelbagai teknik yang mudah untuk diaplikasikan seperti perwarisan dan kelas.

Salah satu kelebihan C++ ialah ia tidak bergantung kepada sebarang jenis mesin untuk dilaksanakan. Ini bermakna bagi melaksanakan aplikasi yang dibangunkan menggunakan bahasa pengaturcaraan C++, tiada sebarang penambahan atau perubahan yang perlu dilakukan ke atas mesin atau peranti tersebut.

3.4 PEMBAHAGIAN TUGAS MEMBANGUNKAN BIST ALU

BIST ALU ia telah dibangunkan dengan kerjasama Tengku Dian Shahida (WEK 000302). Ini berikutan faktor skop yang luas dan masa yang terhad. Bagi memastikan pembangunan BIST ALU dilaksanakan dengan serentak, pembahagian tugas telah dilakukan. Kerja-kerja pembangunan BIST ALU telah dibahagikan kepada tiga, iaitu:

- Pembangunan ALU 8 bit.
- Pembangunan *Linear Feedback Shift Register* (LFSR) dan *Multiple Input Shift Register* (MISR).
- Pengekodan C++ bagi tujuan analisis pengenalan (*signature analysis*).

Kerja-kerja pembangunan ALU 8 Bit dan pengekodan C++ dilaksanakan oleh penulis. Manakala kerja-kerja membangunkan LFSR dan MISR dilaksanakan oleh Tengku Dian Shahida.

3.5 KESIMPULAN

Dua perkara utama bagi memastikan pembangunan BIST ALU berjalan lancar ialah pendekatan pembangunan BIST ALU dan kaedah pembangunan BIST ALU. BIST ALU dibangunkan dengan menggabungkan dua bahasa pengaturcaraan iaitu VHDL dan C++. BIST ALU dibangunkan dengan kerjasama Tengku Dian Shahida.

BAB 4

REKABENTUK BIST ALU

BAB 4

REKABENTUK BIST ALU

4.1 PENGENALAN

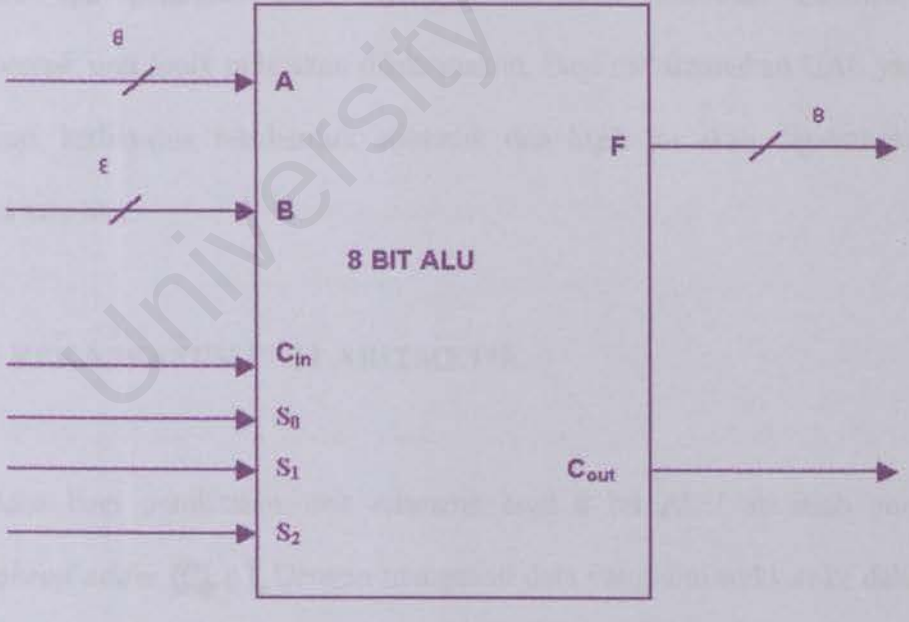
Selepas analisis dilakukan terhadap subjek-subjek yang berkaitan dengan *Built-In-Self-Test Arithmetic Logic Unit* (BIST ALU), langkah seterusnya adalah merekabentuk litar BIST ALU bagi tujuan memodelkannya. Proses merekabentuk BIST ALU melibatkan gabungan hasil kajian yang dilakukan bersama perbincangan yang dilakukan bersama penyelia penasihat dan moderator semasa proses *viva*.

Proses pembangunan BIST ALU melibatkan empat proses utama (Rujuk Jadual 1.4) iaitu pembangunan 8 bit ALU, pembangunan BIST, penggabungan 8 bit ALU dan BIST dan pengekodan C++. Bagi memastikan proses pembangunan berjalan lancar, rekabentuk BIST ALU yang dibangunkan perlulah dilakukan dengan teliti dan terperinci.

4.2 PEMBANGUNAN 8 BIT ALU

Arithmetic Logic Unit (ALU) merupakan jantung bagi setiap mikropemproses dimana ia berfungsi untuk pusat bagi melaksanakan semua operasi aritmetik dan logik bagi mikropemproses tersebut. ALU juga melaksanakan semua arahannya dalam bentuk digital (0,1).

Bagi pembangunan BIST ALU, 8 bit ALU yang terdiri daripada tiga pin pilihan digunakan sebagai litar untuk diuji (Rujuk Rajah 4.1). Pin A dan pin B berfungsi untuk menerima 8 bit data masukan manakala data keluaran pula disalurkan dari pin F.



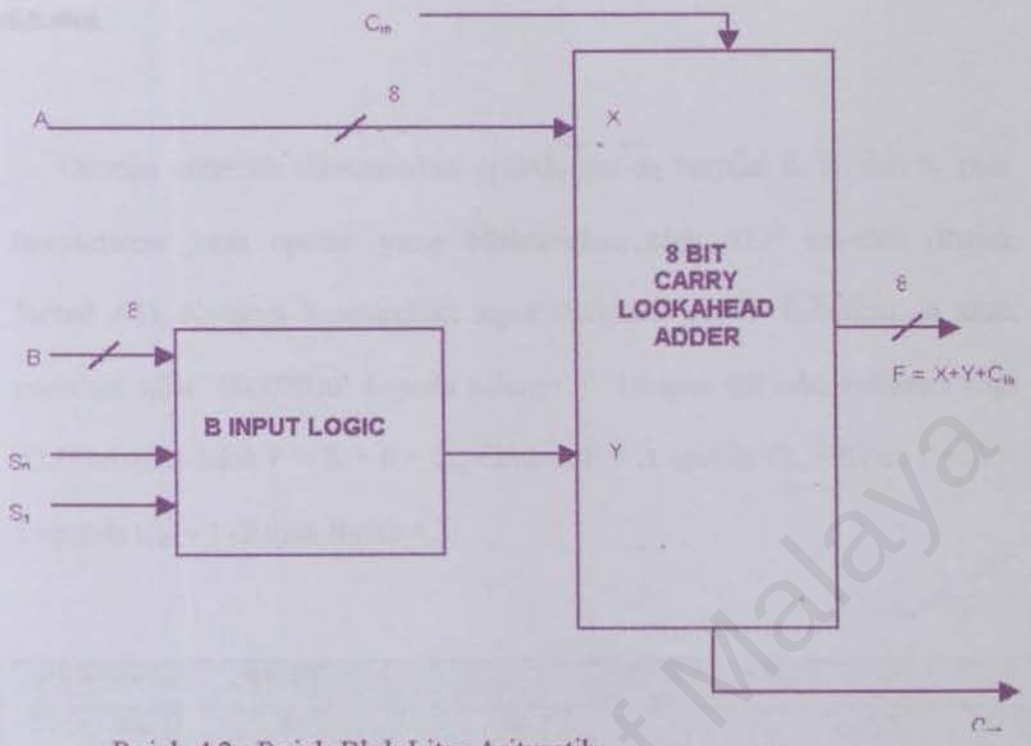
Rajah 4.1 : Rajah kotak hitam bagi 8 bit ALU

Terdapat tiga pin pilihan bagi 8 bit ALU yang dibangunkan. Pin S_0 dan S_1 gunakan bagi menentukan jenis operasi aritmetik atau logik yang perlu dilaksanakan manakala pin S_2 berfungsi untuk menentukan mod pengoperasian bagi ALU tersebut. Operasi aritmetik akan dilaksanakan sekiranya mod 0 dipilih seterusnya ALU akan melaksanakan operasi logik jika mod 1 dipilih. Hanya empat operasi logik dapat dilaksanakan oleh ALU ini tetapi lapan operasi aritmetik dapat dilaksanakan berikutan terdapat pin C_{in} yang mewakili pembawa input (*input carry*) yang menggandakan nilai operasi aritmetik yang dilaksanakan pin Cout pula akan mengeluarkan pembawa output (*output carry*) bagi ALU berkenaan.

Seterusnya, bagi membangunkan 8 bit ALU, rekabentuknya dibahagikan kepada tiga peringkat iaitu merekabentuk unit aritmetik. Seterusnya rekabentuk unit logik pula akan dilaksanakan. Bagi melaksanakan UAL yang lengkap, kedua-dua rekabentuk aritmetik dan logik ini akan digabungkan dalam satu litar.

4.2.1 REKABENTUK UNIT ARITMETIK

Asas bagi pembinaan unit aritmetik bagi 8 bit ALU ini ialah *carry lookahead adder* (CLA). Dengan mengawal data yang dimasukkan ke dalam CLA, ia mampu melaksanakan pelbagai operasi aritmetik yang berbeza. Rajah 4.2 menunjukkan konfigurasi bagi satu set input ke dalam CLA dengan dikawal oleh pin S_0 dan S_1.



Rajah 4.2 : Rajah Blok Litar Aritmetik

Terdapat dua pin masukan bagi litar ini iaitu pin A dan pin B dan pin F sebagai pin keluaran. Input dari pin B akan melalui *B input logic* seterusnya menjadi input bagi CLA melalui saluran Y. Pembawa sambutan (C_{in}) pula membawa pembawa input sambutan ke lokasi nilai bit paling rendah (*least-significant-bit position*) di dalam CLA. Output yang diperolehi dari C_{out} merupakan keluaran dari lokasi nilai bit paling tinggi (*most-significant-bit position*) dari CLA Nilai keluaran yang diperolehi dari CLA merupakan hasil operasi aritmetik, iaitu:

$$F = X + Y + C_{in}$$

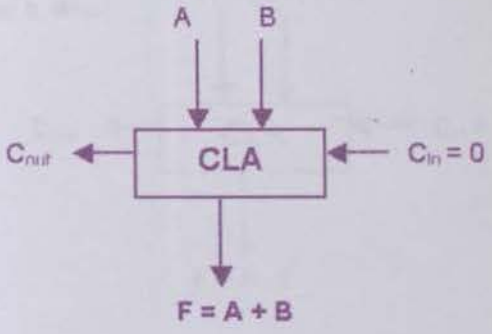
Di mana X merupakan 8 bit nombor binari dari pin A manakala Y pula merupakan 8 bit nombor binari yang diperolehi dari *B input logic*. Simbol '+'

yang digunakan mewakili sebarang operasi aritmetik yang digunakan bagi litar ini.

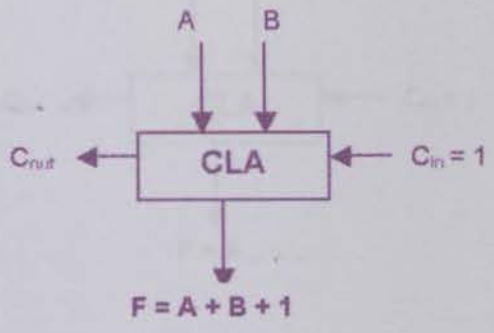
Operasi aritmetik dilaksanakan apabila pin S_2 bernilai 0. S_0 dan S_1 pula menentukan jenis operasi yang dilaksanakan oleh ALU tersebut (Rujuk Jadual 4.1). Katakan keseluruhan input daripada pin B diabaikan, ia akan memberi nilai '00000000' kepada saluran Y. Dengan ini nilai keluaran bagi ALU tersebut ialah $F = X + 0 + C_{in}$ dimana $F = A$ apabila $C_{in} = 0$ dan $F = A + 1$ apabila $C_{in} = 1$ (Rujuk Rajah 4.3).

PILIHAN		INPUT	$F = X + Y + C_{in}$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	Keseluruhan 0	G = A (Umpukan)	G = A + 1 (Penokokan)
0	1	B	G = A + B (Penambahan)	G = A + B + 1
1	0	B'	G = A + B'	G = A + B' + 1 (Penolakkan)
1	1	Keseluruhan 1	G = A – 1 (Pengurangan)	G = A (Umpukan)

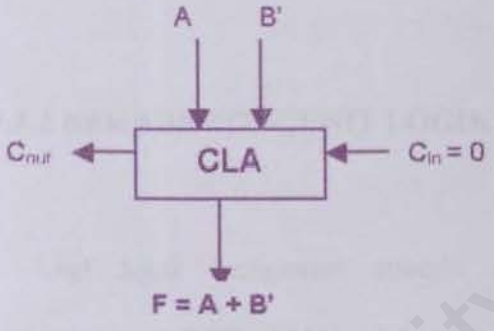
Jadual 4.1 : Jadual Kebenaran Unit Aritmetik.8 Bit ALU



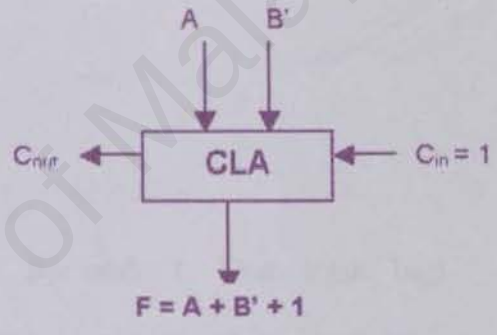
(a) Operasi Penambahan



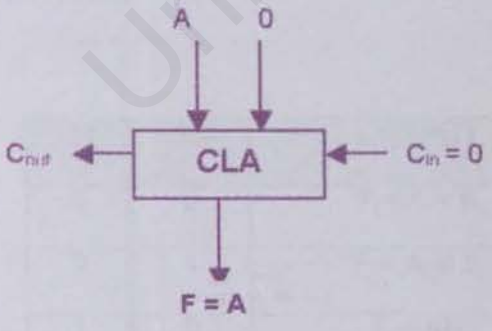
(a) Operasi Penambahan Dengan Sambutan



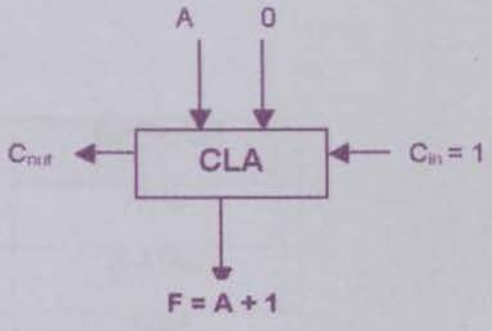
(c) Operasi Penambahan Dengan Pelengkap Duaan B



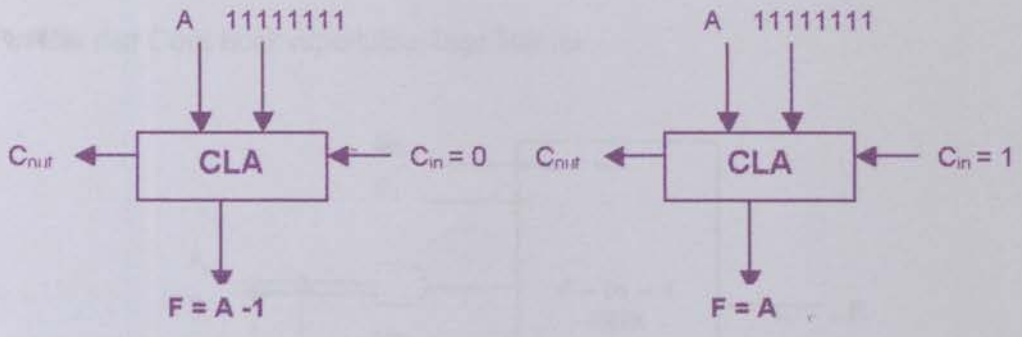
(d) Operasi Penolakkan



(e) Operasi Umpukan Nilai A



(f) Operasi Penokokan Nilai A



(g) Operasi Pengurangan nilai A (h) Umpukan nilai A

Rajah 4.3 : Operasi Yang Diilaksanakan Oleh 8 Bit ALU

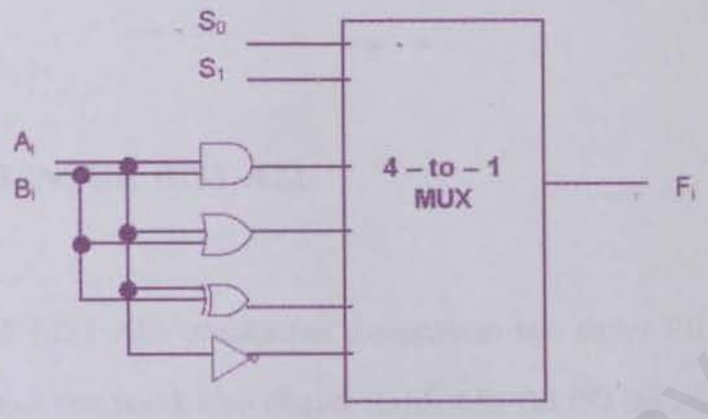
4.2.2 REKABENTUK UNIT LOGIK

Unit logik beroperasi apabila nilai S_2 ialah 1. Unit logik bagi pembangunan BIST ALU melaksanakan empat jenis operasi asas logik iaitu, DAN, ATAU, eksklusif-ATAU dan TAK. Dalam unit logik juga, S_0 dan S_1 yang menentukan jenis operasi logik yang perlu dilaksanakan oleh unit logik (Rujuk Jadual 4.2).

S1	S2	OUTPUT	OPERASI
0	0	$F_i = A_i + B_i$	ATAU
0	1	$F_i = A_i \oplus B_i$	X-ATAU
1	0	$F_i = A_iB_i$	DAN
1	1	$F_i = A_i'$	TAK

Jadual 4.2 : Jadual Kebenaran Bagi Unit Logik 8 Bit ALU

Rajah 4.4 menunjukkan litar unit logik bagi 8 bit ALU yang dibangunkan. Pin Cin dan Cout tidak diperlukan bagi litar ini.



Rajah 4.4 : Rajah Logik Bagi Litar Logik 8 Bit ALU

4.3.3 REKABENTUK *ARITHMETIC LOGIC UNIT*

Setelah selesai merekabentuk unit aritmetik dan unit logik, kedua-dua litar tersebut digabungkan bagi membentuk satu litar ALU yang lengkap (Rujuk Rajah 2.3). Output yang diperolehi bagi setiap unit akan disalurkan kepada pemultipleksan 2-ke-1 bersama-sama nilai pin pilihan (S_2). Seterusnya hasil keluaran bagi ALU tersebut akan dikeluarkan berdasarkan nilai mod S_2 yang dipilih.

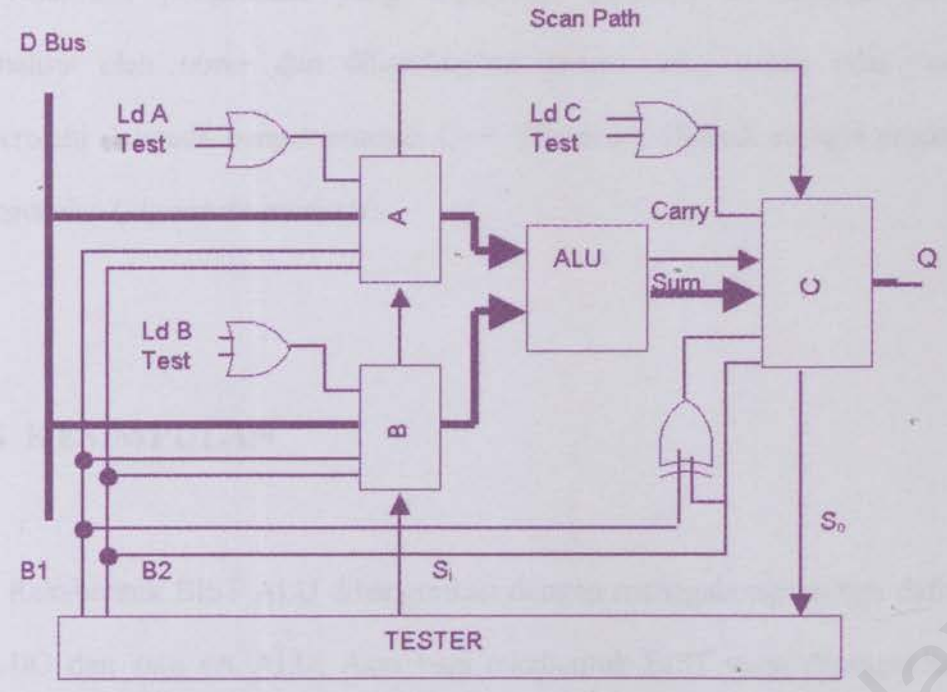
Rekabentuk 8 bit ALU ini mampu melaksanakan 12 operasi aritmetik dan logik; 8 operasi aritmetik dan 4 operasi logik (Rujuk Jadual 4.3). Setiap pin S_2, S_1, S_0 dan C_{in} berfungsi untuk menentukan jenis operasi yang dilakukan. Operasi aritmetik dilaksanakan apabila pin S_2 bernilai 0 manakala 4 operasi logik akan dilaksanakan apabila S_2 bernilai 1. pin C_{in} hanya berfungsi dalam

litar ini apabila operasi aritmetik dilaksanakan, ia tidak memberi apa-apa kesan terhadap hasil keluaran sekiranya operasi logik sedang dilaksanakan.

4.3 REKABENTUK BIST ALU

Rekabentuk BIST ALU melibatkan penggunaan tiga daftar BILBO dan litar ALU sebagai litar untuk diuji (Rujuk Rajah 4.5). BILBO digunakan bagi sebagai asas kepada rekabentuk LFSR dan MISR. Daftar A, B dan C merupakan daftar BILBO yang fungsinya boleh dikawal mengikut mod yang telah ditetapkan (Rujuk Jadual 4.4).

Rekabentuk BIST ALU ini mampu berfungsi dalam keadaan normal dan dalam keadaan pengujian. Dalam mod normal, litar ini beroperasi sebagai 8 bit ALU dimana data akan dihantar dari bas data (*D Bus*) ke ALU melalui daftar A dan B tanpa sebarang perubahan berlaku ke atasnya. Seterusnya data tersebut akan diproses di ALU dan hasil keluarannya akan dihantar melalui daftar C untuk dihantar semula kepada mikropemproses.



Rajah 4.5 : Rekabentuk BIST ALU

Proses pengujian berlaku apabila nilai B_1 dan B_2 bertukar kepada nilai . Apabila proses pengujian dilakukan, S_i akan mengeluarkan nilai awalan ada daftar A, B dan C. Nilai yang diberikan kepada B_1 dan B_2 telah nukarkan mod daftar BILBO A dan B kepada mod LFSR manakala daftar BO C kepada mod MISR. Dalam proses ini, LFSR akan menjana anyak 255 corak uji. (2^n-1). Seterusnya kesemua corak uji tersebut akan oses di dalam ALU dan hasilnya akan dimasukkan ke dalam MISR. Hasil rasi yang dilaksanakan tersebut akan dipadatkan (*compact*) di dalam SR membentuk pengenalan (*signature*) dan keluar sebagai S_o.

Nilai-nilai pengenalan yang diperolehi daripada S_o tersebut akan dianalisis oleh *tester* dan dibandingkan dengan nilai bebas ralat yang diperolehi daripada pengaturcaraan C++. Proses ini dikenali sebagai analisis pengenalan (*signature analysis*).

4.4 KESIMPULAN

Rekabentuk BIST ALU dibangunkan dengan menggabungkan tiga daftar BILBO dan satu set ALU. Asas bagi rekabentuk BIST yang dibangunkan ialah daftar BILBO dimana ia akan beroperasi mengikut mod yang telah ditetapkan.

BAB 5

IMPLEMENTASI SISTEM

BAB 5

IMPLEMENTASI SISTEM

5.1 PENGENALAN

Seperti yang telah dijelaskan di awal perbincangan, rekabentuk *Built-In-Self-Test Arithemetic Logic Unit* (BIST ALU) dibangunkan dengan membuat prototaip bagi litar ini dalam aturcara *VHSIC Hardware Description Language* (VHDL) disimulasi dengan menggunakan perisian *PeakFPGA Designer Suite FPGA Synthesis Edition 5.20c.* Aktiviti ini dibuat selepas menyediakan rekabentuk BIST ALU.

Seterusnya, bab ini akan membincangkan strategi-strategi yang diambil bagi mengimplementasi BIST ALU. Daripada keseluruhan langkah yang diambil, pengaturcaraan merupakan aktiviti terpenting kerana ia merupakan proses yang akan merealisasi segala perancangan rekabentuk yang telah dibuat.

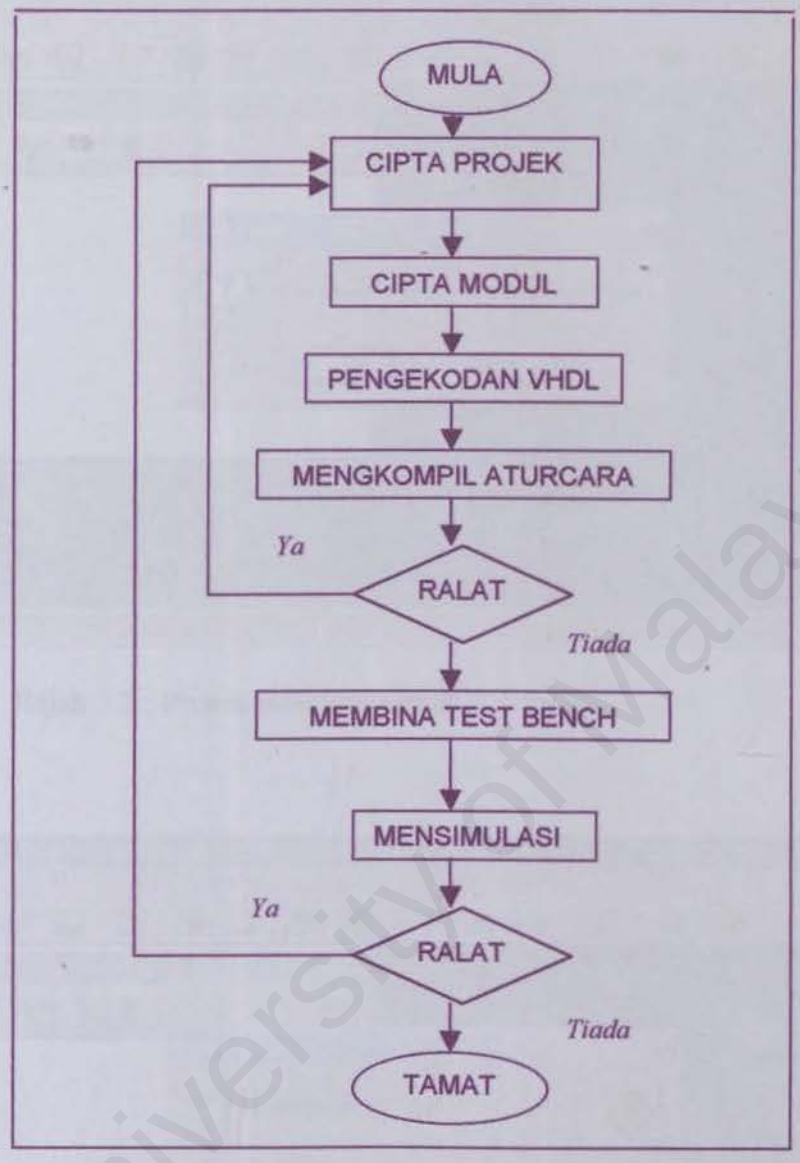
Fasa ini merupakan titik puncak yang menentukan kejayaan pembangunan BIST ALU di mana ketelitian dan kesabaran yang tinggi diperlukan. Kesilapan atau sebarang ralat yang timbul boleh mengakibatkan

sistem tidak dapat beroperasi dengan baik dan sempurna seterusnya akan melengahkan masa pembangunan sesebuah projek.

5.2 *PEAKFPGA Designer Suite FPGA Synthesis Edition 5.20c*

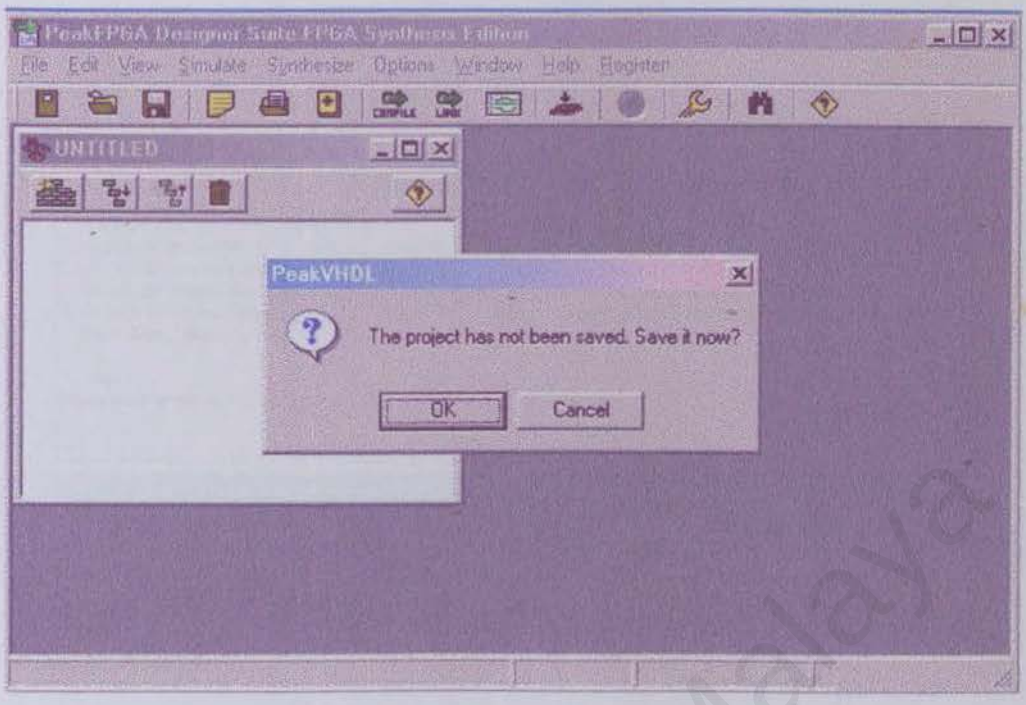
Bagi mensimulasi BIST ALU, perisian *PEAKFPGA Designer Suite FPGA Synthesis Edition 5.20c* telah digunakan. Perisian yang menyokong piawaian *IEEE Standard 1076-1993* ini mudah digunakan dan antaramukanya ramah pengguna. Rajah 5.1 menunjukkan carta alir proses yang diambil bagi menggunakan perisian ini.

Langkah pertama yang diambil untuk menggunakan perisian *PEAKFPGA Designer Suite FPGA Synthesis Edition 5.20c* ialah mencipta projek yang akan dibangunkan (Rujuk Rajah 5.2). Seterusnya modul yang terkandung dalam projek tersebut akan dicipta. Atrucara VHDL akan ditulis di dalam modul-modul tersebut (Rujuk Rajah 5.3). Modul-modul yang dicipta ini boleh diguna semula pada projek-projek lain dan ini akan memudahkan proses pembangunan projek terutamanya semasa proses membentuk hirarki.

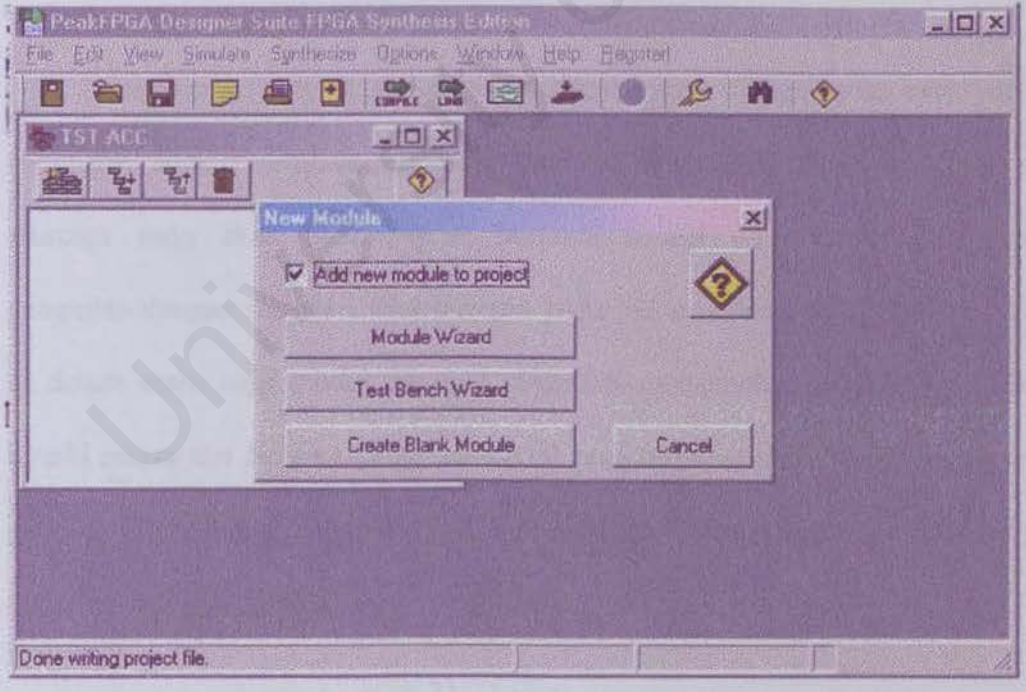


Rajah 5.1 : Carta alir langkah penggunaan PEAKFPGA.

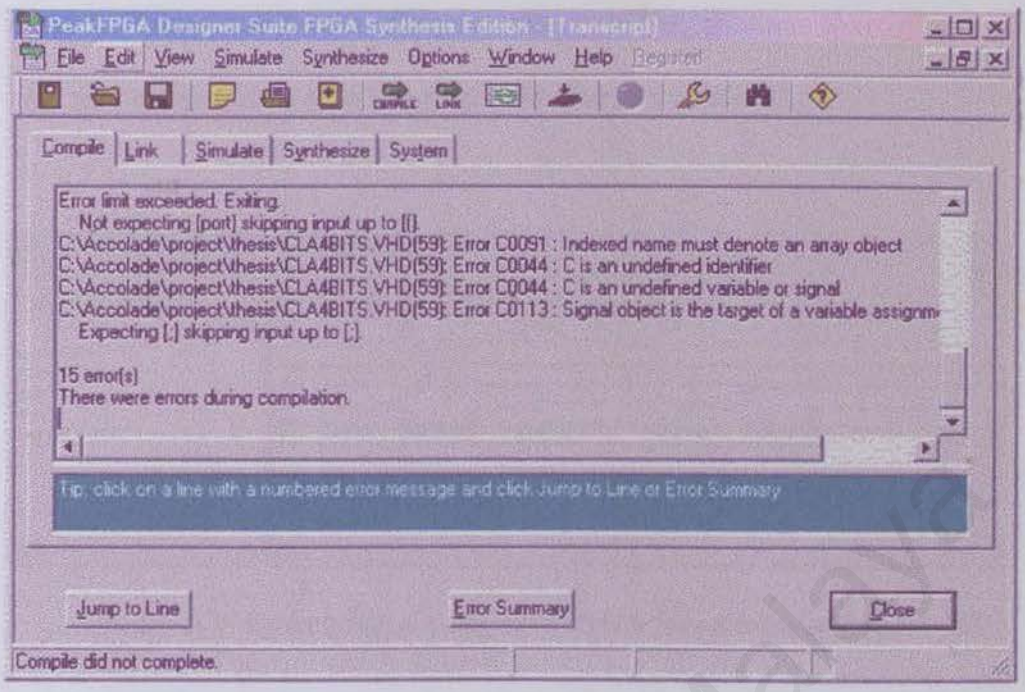
Setelah selesai membuat aturcara, langkah seterusnya ialah menkompil aturcara tersebut bagi mengenalpasti sebarang ralat sintaks yang terdapat padanya (Rujuk Rajah 5.4). Semasa proses ini semua ralat sintaks yang terdapat akan dipaparkan beserta baris dan sebab berlakunya ralat. Pada ketika ini, sekiranya terdapat ralat logik pada aturcara, ia perlu disemak semula sehingga aturcara yang dibuat bebas ralat.



Rajah 5.2 : Proses mencipta projek.



Rajah 5.3: Proses mencipta modul.

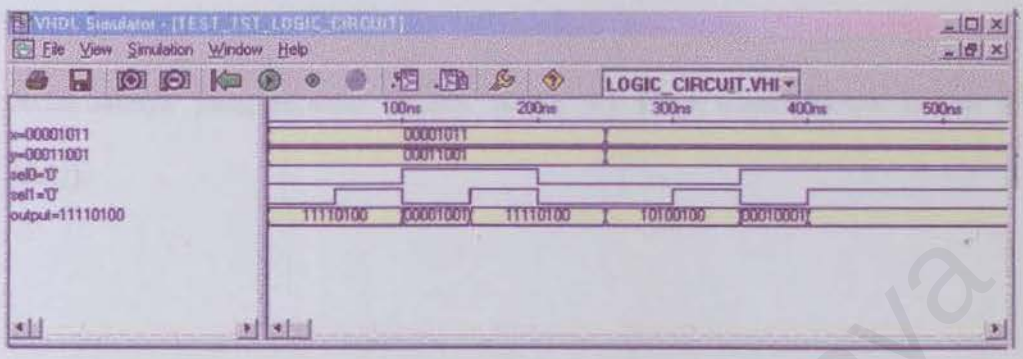


Rajah 5.4: Proses mengkompil aturcara. Rajah menunjukkan terdapat ralat pada aturcara yang sudah dibangunkan.

Seterusnya, setelah mendapati aturacara yang dibuat bebas ralat, aktiviti seterusnya adalah membina *test bench* bagi modul tersebut. *Test bench* ialah satu aturcara yang akan dihubungkan bersama modul yang dibina bagi tujuan pengujian dengan memberi nilai tertentu pada pin masukkan yang telah diisytihar di dalam entiti bagi modul tersebut. Sebelum mensimulasi modul yang dibina, hirarki antara *test bench* dan modul-modul yang terdapat di dalam projek tersebut perlulah dibina terlebih dahulu. *Test bench* sentiasa berada pada hirarki teratas di dalam projek yang dibina.

Proses terakhir yang diperlukan ialah mensimulasi modul. Hasil keluaran bagi modul yang dibuat akan dipaparkan dalam bentuk isyarat digital (Rujuk Rajah 5.5). Hasil yang diperolehi adalah bergantung kepada nilai masukkan yang

diberi di dalam *test bench*. Daripada nilai tersebut, aturcara akan dinilai semula bagi memastikan tiada sebarang ralat pada aturcara tersebut.



Rajah 5.5: Isyarat digital yang diperolehi semasa mensimulasi modul.

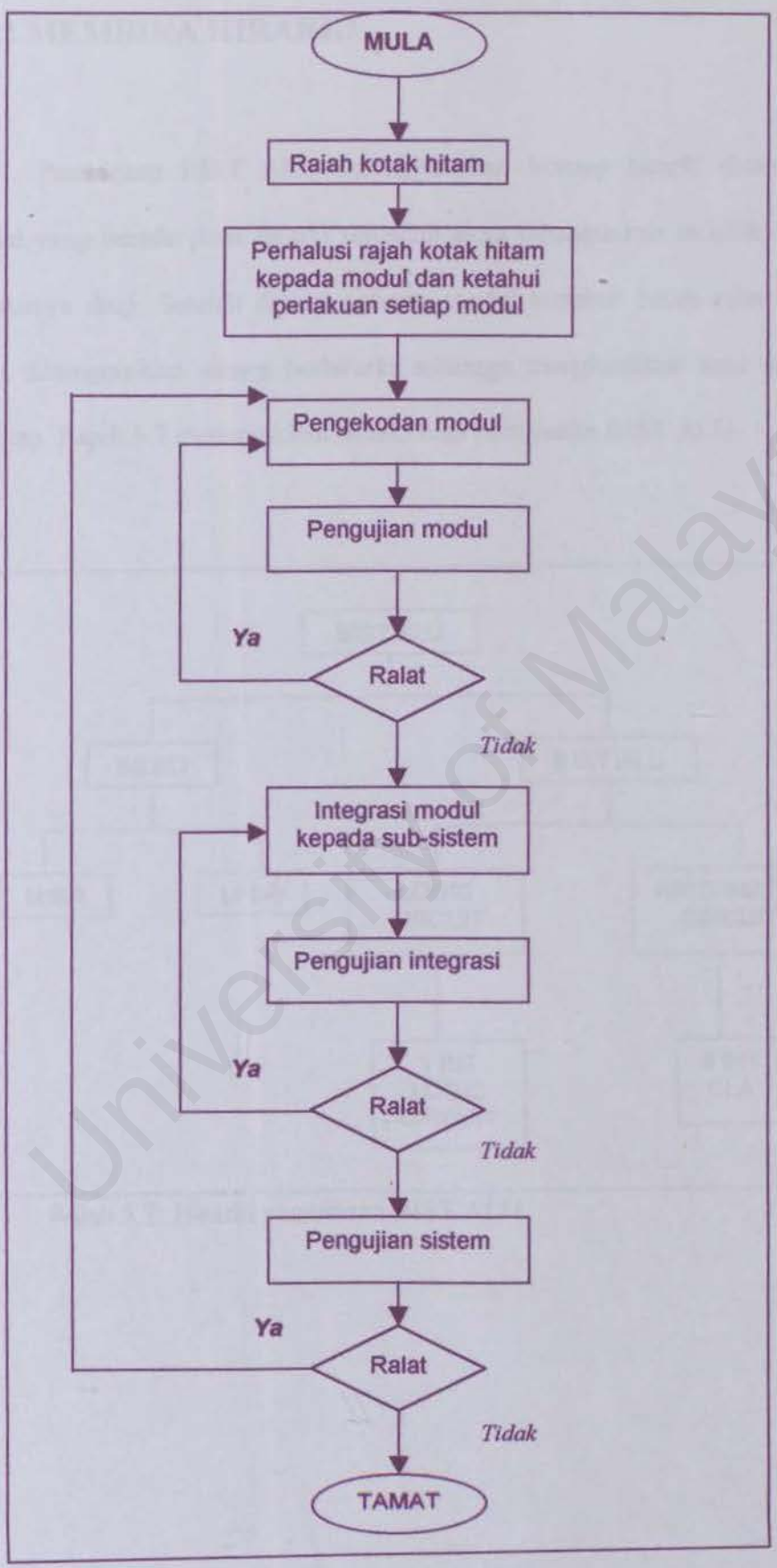
5.3 STRATEGI PEMBANGUNAN BIST ALU

Rajah 5.6 menunjukkan proses pelaksanaan yang diambil bagi membangunkan BIST ALU. Daripada rajah ini jelas menunjukkan bahawa proses pelaksanaannya dilaksanakan secara berperingkat-peringkat dan proses pengujian dilakukan pada setiap peringkat dari semasa ke semasa.

5.3.1 RAJAH KOTAK HITAM

Langkah pertama dalam membangunkan sesuatu rekabentuk elektronik adalah membuat rajah kotak hitam bagi rekabentuk tersebut. Rajah kotak hitam merupakan gambaran kasar tentang ciri-ciri masukkan dan keluaran bagi

rekabentuk yang sedang dibangunkan. Fungsi bagi setiap pin masukkan dan keluaran dikenalpasti. Kemudian rajah itu diperhalusi sehingga ke peringkat yang paling rendah. Disamping itu perlakuan (*behavior*) bagi rajah kotak hitam tersebut perlu dikenalpasti bagi memastikan litar yang dibangunkan mengikut perlakuannya yang sebenar (Rujuk Rajah 4.1 Bagi Rajah Kotak Hitam BIST ALU).



Rajah 5.6 : Carta Alir Pembangunan BIST ALU

5.3.2 PENGATURCARAAN

Setelah selesai merekabentuk rajah kotak hitam, langkah seterusnya ialah membangunkan rekabentuk tersebut dalam bentuk pengaturcaraan VHDL. Elemen yang paling penting bagi bahasa pengaturcaraan VHDL adalah menulis entiti bagi rekabentuk yang ingin dibangunkan. Entiti merupakan pengisyhtiharan pin-pin masukkan dan keluaran bagi sesebuah modul. Jenis pin masukkan dan keluaran yang digunakan juga dinyatakan semasa pengisytiharan entiti. Rajah berikut merupakan contoh pengisytiharan entiti bagi *Carry Lookahead Adder*.

```
entity CLA is
    port (
     a_in: in std_logic_vector(7 downto 0);
     b_in: in std_logic_vector(7 downto 0);
     c_in: in std_logic;
     c_out  : out std_logic_vector(7 downto 0));

end CLA;
```

Rajah 5.8 : Contoh pengisytiharan entiti bagi *Carry Lookahead Adder*.

Seperti mana yang telah dinyatakan, setiap modul atau sub-komponen yang dibangunkan perlu diperhalusi dan dikenalpasti fungsi dan cara ia beroperasi. Berdasarkan fungsi setiap modul tersebut, rekabentuk (*architecture*) bagi modul tersebut akan di tulis.

Secara umumnya, *architecture* akan mengenakan operasi ke atas nilai yang dimasukkan dalam pin masukkan dan hasil yang diperolehi daripada operasi tersebut akan dikeluarkan melalui pin keluaran. Operasi di sini boleh didefinasikan sebagai proses yang mengandungi pernyataan secara berjujukan yang beroperasi ke atas nilai yang diberi atau koleksi komponen yang mewakili sub-modul. Sekiranya proses tersebut memerlukan penjanaan nilai-nilai diantara modul-modul di dalamnya, isyarat (*signal*) akan digunakan. *Signal* merupakan analogi bagi pendawaian (*wiring*) yang terdapat di dalam modul.

Dalam proses pembangunan BIST ALU, *architecture* digunakan bagi mendiskripsikan *behavior* dan stuktur (*structure*) bagi modul-modul tersebut. Pengisytiharan *behavior* digunakan bagi menerangkan tentang rekabentuk dalaman dan cara ia beroperasi. Secara asasnya, *behavior* bagi modul dinyatakan dengan menggunakan pernyataan *signal* antara proses. Rajah 5.9 menunjukkan contoh pengisytiharan *behavior* bagi *Carry Lookahead Adder*.

```
architecture BEHAVIOR of CLA is

-- signal (internal wiring bagi CLA)
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 0);
signal  carry_in  :  std_logic;



begin
```

```
   sum(0) <= a_in(0) xor b_in(0) xor c_in;
      g <= a_in and b_in;
      p <= a_in xor b_in;

    P1: process(g,p,c_internal)

    begin
        carry_in <= c_in;

      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 P1;

   P2: process (c_out,p,c_internal)

   begin

   c_out (0) <= p(0) xor carry_in;

   for value in 1 to 7 loop
      c_out(value) <= p(value) xor c_internal(value);
   end loop;
   end process P2;

end BEHAVIOR;
```

Rajah 5.9: Contoh Pengisytiharan *Behavior Bagi Carry Lookahead Adder*

Kegunaan *architecture* yang seterusnya adalah bagi menerangkan struktur bagi sesebuah mudul. Pengisytiharan stuktur akan menyatakan sambungan yang

terdapat antara modul dan sistem. Sub-sub modul yang telah dibangunkan akan diintegrasikan di dalam stuktur. Tiga elemen penting dalam pengisytiharan stuktur ialah komponen (*component*), *signal* dan pemetaan (*port map*).

Component akan menyatakan modul-modul yang akan diintegrasikan dalam pengisytiharan struktur. *Signal* pula merupakan isyarat yang digunakan antara modul-modul untuk berkomunikasi. *Port map* pula merupakan sambungan bagi litar-litar yang dibangunkan dan cara setiap pin masukkan dan keluaran dipetakan. Biasanya pengisytiharan stuktur menggambarkan hirarki yang terdapat dalam sesebuah litar. Rajah 5.10 merupakan contoh bagi pengisytiharan struktur yang digunakan bagi membangunkan litar logik dalam *Arithmetic Logic Unit* (ALU):

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

entity toplevel is
    port (
        op1: in STD_LOGIC_VECTOR(7 downto 0);
        op2: in STD_LOGIC_VECTOR(7 downto 0);
        operator: in STD_LOGIC_VECTOR (2 downto 0);
        result: out STD_LOGIC_VECTOR(7 downto 0)
    );
end toplevel;

architecture toplevel_arch of toplevel is

signal s_result : std_logic;
```

```
component and1
    port (
        a: in STD_LOGIC_VECTOR(7 downto 0);
        b: in STD_LOGIC_VECTOR(7 downto 0);
        c_out: out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;

component or1
    port (
        a: in STD_LOGIC_VECTOR(7 downto 0);
        b: in STD_LOGIC_VECTOR(7 downto 0);
        d_out: out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;


component not1
port (a: in STD_LOGIC_VECTOR(7 downto 0);
      e_out: out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;

component xor1
    port (
        a: in STD_LOGIC_VECTOR(7 downto 0);
        b: in STD_LOGIC_VECTOR(7 downto 0);
        f_out: out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;


component host
    port (
        operator: in STD_LOGIC_VECTOR (2 downto 0);
        c_out : in STD_LOGIC_VECTOR(7 downto 0);
```

```
        d_out : in STD_LOGIC_VECTOR(7 downto 0);
        e_out : in STD_LOGIC_VECTOR(7 downto 0);
        f_out : in STD_LOGIC_VECTOR(7 downto 0);
        result: out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;

signal s_cout : STD_LOGIC_VECTOR(7 downto 0);
signal s_dout : STD_LOGIC_VECTOR(7 downto 0);
signal s_eout : STD_LOGIC_VECTOR(7 downto 0);
signal s_fout : STD_LOGIC_VECTOR(7 downto 0);

begin

U1:  and1 port map (a=>op1, b=>op2, c_out=> s_cout);
U2:  or1  port map (a=>op1, b=>op2, d_out=> s_dout);
U3:  not1 port map (a=>op1, e_out=> s_eout);
U4:  xor1  port map (a=>op1, b=>op2, f_out=> s_fout);
U5:  host port map (operator=>operator, c_out=>s_cout,
d_out=> s_dout, e_out=> s_eout,
f_out=>s_fout,result=>s_result);

end toplevel_arch;
```

Rajah 5.10 : Pengisytiharan Struktur Bagi Litar Logik Bagi *Arithmetic Logic Unit* (ALU)

5.4. PENGUJIAN

Setelah selesai mengekod rekabentuk BIST ALU, langkah seterusnya ialah menguji aturcara tersebut. Proses pengujian ini dilakukan secara berkala sejak awal pembangunan modul dijalankan. Kaedah yang digunakan begi menguji rekabentuk BIST ALU ialah dengan mensimulasikan aturcara tersebut dan membandingkan hasil yang diperolehi dengan nilai yang sebenar. Proses pengujian ini akan dihuraikan dengan lebih lanjut dalam bab seterusnya.

5.5 KESIMPULAN

Pembangunan BIST ALU dilakukan dengan menggunakan teknik bawah-atas di mana modul-modul yang berada pada hirarki rendah akan dibangunkan terlebih dahulu, seterusnya diuji. Selepas itu, barulah modul-modul ini akan digabung dan dikonfigurasikan dalam satu litar lengkap.

BAB 6

PENGUJIAN SISTEM

BAB 6

PENGUJIAN SISTEM

6.1 PENGENALAN

Pengujian merupakan langkah yang paling penting dalam kitar hayat pembangunan sistem. Ini adalah bagi memastikan sistem yang dibangunkan berjalan lancar dan memenuhi segala spesifikasi yang telah ditetapkan pada awal pelaksanaan projek. Proses pengujian dilakukan bagi mengenalpasti sebarang kesilapan atau ralat yang terdapat pada sistem yang sedang dibangunkan atau kegagalan yang terdapat pada perisian yang digunakan. Kegagalan perisian merujuk kepada sejauh mana sistem mampu berfungsi dengan baik dan tepat semasa penjanaan nilai masukkan dilaksanakan.

Tujuan pengujian dijalankan adalah bagi:

- Untuk memastikan sistem yang dibangunkan adalah bebas ralat sebelum ia diedarkan kepada pengguna.
- Menguji sistem bagi memastikan ia memenuhi segala spesifikasi yang telah ditetapkan di awal projek.

Secara umumnya, proses pengujian bagi pembangunan *Built-In-Self-Test Arithmetic Logic Unit* (BIST ALU) telah melibatkan tiga peringkat iaitu pengujian modul, pengujian integrasi dan pengujian sistem.

6.2 PENGUJIAN MODUL

Modul merupakan koleksi komponen yang berfungsi dengan sendiri dan beroperasi tanpa bergantung kepada komponen-komponen lain [NOOR 01]. Contoh modul yang dibangunkan dalam pembangunan BIST ALU ialah get ATAU, *Linear Feedback Shift Register* (LFSR) dan *Carry Lookahead Adder* (CLA).

Sepertimana yang telah dinyatakan dalam bab 5, proses pengujian bagi BIST ALU dilaksanakan secara berkala dimana setiap kali sesebuah modul dibangunkan, ujian akan dibuat ke atas modul tersebut. Proses pengujian modul dibuat dengan mensimulasi aturcara yang telah dibuat. Sebelum aturcara tersebut disimulasi, *test bench* perlulah dibuat terlebih dahulu.

Test bench merupakan satu aturcara yang memberi nilai masukkan ke atas litar tersebut. Hasil keluaran yang diperolehi daripada proses simulasi ini adalah dalam bentuk graf isyarat digital. Hasil ini akan dibandingkan dengan pengiraan secara manual bagi memastikan ketepatan keluaran.

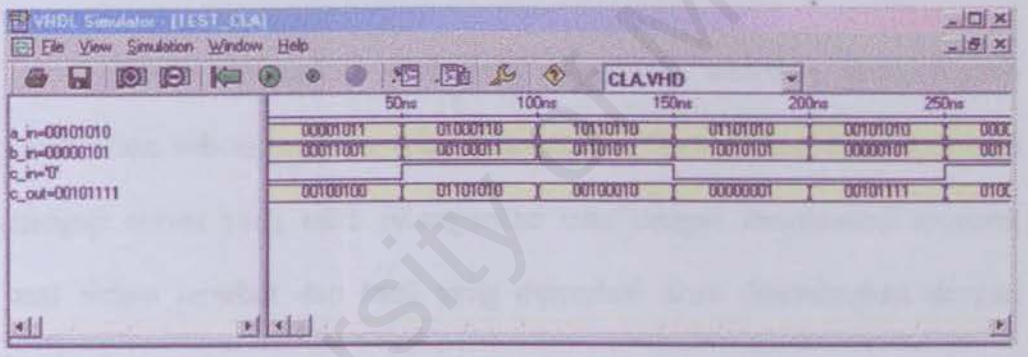
Tujuan utama pengujian modul dilaksanakan adalah bagi memastikan tiada ralat yang terdapat bagi modul-modul yang boleh mempengaruhi sistem setelah ia digabungkan kelak. Rajah 6.1 merupakan contoh aturcara *test bench* dan hasil keluaran yang diperolehi.

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

entity testrot is
end testrot;

architecture stimulus of testrot is
    component CLA
        port( a_in: in std_logic_vector(7 downto 0);
              b_in: in std_logic_vector(7 downto 0);
              c_in: in std_logic;
              c_out: out std_logic_vector(7 downto 0));
    end component;

    constant PERIOD: time := 50 ns;

    signal a_in     :  std_logic_vector(7 downto 0);
    signal b_in     :  std_logic_vector(7 downto 0);
   signal c_in      :  std_logic;
   signal c_out     :  std_logic_vector(7 downto 0);

begin
    DUT: CLA port map(a_in,b_in,c_in,c_out);

    INPUTS: process
    begin
        a_in <= "00001011";
        b_in <= "00011001";
        c_in <= '0';
        wait for PERIOD;
```

```
a_in <= "01000110";
b_in <= "00100011";
c_in <= '1';
wait for PERIOD;

a_in <= "10110110";
b_in <= "01101011";
c_in <= '1';
wait for PERIOD;


a_in <= "01101010";
b_in <= "10010101";
c_in <= '0';
wait for PERIOD;

a_in <= "00101010";
b_in <= "00000101";
c_in <= '0';
wait for PERIOD;


a_in <= "00001010";
b_in <= "00111001";
c_in <= '1';
wait for PERIOD;


a_in <= "00000010";
b_in <= "01010101";
c_in <= '0';
wait for PERIOD;


a_in <= "00001111";
b_in <= "00010101";
c_in <= '1';
wait for PERIOD;
```

```
        a_in <= "01101010";
        b_in <= "01010101";
        c_in <= '0';
        wait for PERIOD;


        wait;

    end process;
end stimulus;
```

Rajah 6.1 : Contoh *Test Bench* Bagi Menguji *Carry Lookahead Adder.*



Rajah 6.2 : Graf Isyarat Digital Hasil Simulasi *Carry Lookahead Adder* Berdasarkan Input Yang Dimasukkan Oleh *Test Bench.*

6.3 PENGUJIAN INTEGRASI

Pengujian integrasi merupakan proses untuk memastikan komponen-komponen bagi sistem tersebut boleh bekerjasama dan beroperasi dengan baik seperti yang telah dinyatakan dalam spesifikasi sistem dan rekabentuk program. Pengujian pada fasa ini akan melibatkan pengujian ke atas koleksi modul-modul yang telah diintegrasikan kepada sub-sistem. Pada ketika ini

masalah yang biasa timbul ialah salah penyesuaian pada antaramuka modul atau ralat pada pemetaan pin.

Kaedah yang digunakan bagi pengujian integrasi adalah sama seperti kaedah pengujian modul iaitu dengan mensimulasi aturcara berdasarkan input yang diberi dari *test bench*.

6.4 PENGUJIAN SISTEM

Sistem merupakan hasil yang diperolehi daripada mengintegrasi keseluruhan sub-sistem yang telah dibangunkan. Metod yang digunakan bagi menguji sistem yang telah dibangunkan iaitu dengan mensimulasi aturcara bagi sistem tersebut dan hasil yang diperolehi akan dibandingkan dengan hasil sebenar.

Terdapat tiga metod yang boleh digunakan bagi menilai litar BIST ALU yang dibangunkan. Cara pertama yang boleh digunakan adalah dengan membuat pengiraan secara manual. Kaedah ini agak sukar dilaksanakan memandangkan hasil keluaran yang agak besar dan kebarangkalian berlakunya ralat agak tinggi.

Kaedah kedua yang boleh digunakan adalah dengan membuat aturcara yang sama dengan perlakuan BIST ALU. Kaedah ini telah digunakan bagi pengujian BIST ALU dan bahasa pengaturcaraan C telah digunakan bagi

mengimplementasikannya. Walaupun kaedah ini agak tepat dan menjimatkan masa hasil keluaran yang diperolehi daripadanya juga perlu dinilai secara manual bagi mengelakkan daripada berlaku sebarang ralat logik

Kaedah terakhir yang boleh digunakan adalah dengan menggunakan cip sebenar sebagai *Design Under Test* (DUT) bagi litar BIST yang dibangunkan. Kaedah ini akan menelan belanja yang agak besar kerana cip ALU yang digunakan tersebut perlulah dipastikan bebas ralat bagi memastikan tiada masalah semasa proses pengujian BIST dilaksanakan.

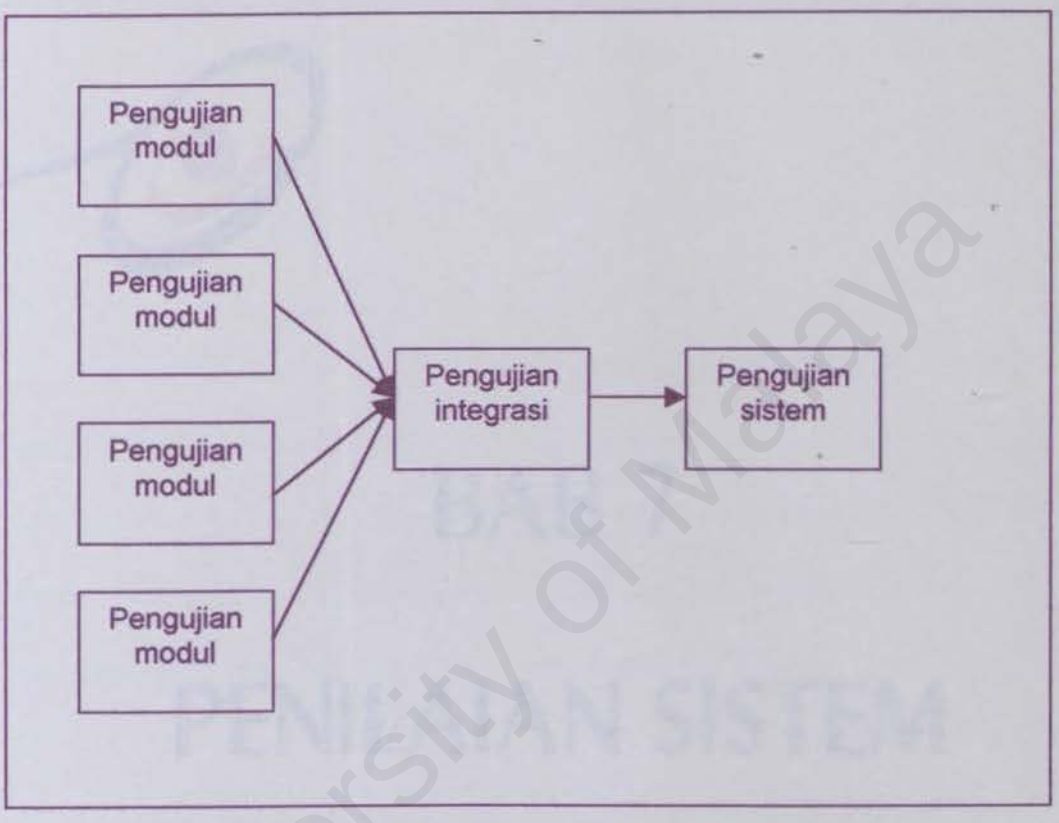
Proses pengujian ini perlulah dilakukan dengan teliti dan berhati-hati kerana kejayaan dalam aktiviti pengujian sistem ini boleh dikatakan sebagai mencerminkan kejayaan sistem yang dibangunkan itu dari segi kefungsian.

6.5 KESIMPULAN

Terdapat tiga peringkat pengujian yang digunakan bagi membangunkan BIST ALU iaitu:

- Pengujian Modul
- Pengujian Integrasi
- Pengujian Sistem.

Rajah 6.3 menunjukkan gambaran kasar proses pengujian yang dijalankan bagi pembangunan BIST ALU.



Rajah 6.3 : Gambaran Proses Pengujian Bagi Pembangunan BIST ALU

BAB 7

PENILAIAN SISTEM

BAB 7

PENILAIAN SISTEM

7.1 PENGENALAN

Aktiviti terakhir dalam proses pembangunan *Built-In-Self-Test Arithmetic Logic Unit* (BIST ALU) ialah penilaian sistem. Pada ketika ini, keseluruhan proses pembangunan BIST ALU akan dinilai semula dari semua aspek. Ia adalah bagi tujuan menilai kerelevenan pembangunan projek dan kesesuaiannya untuk diperkembangkan pada masa akan datang.

Antara isu-isu yang akan dibincangkan dalam bab ini ialah, masalah yang dihadapi sepanjang pembangunan BIST ALU, kelebihan dan kelemahannya, seterusnya cadangan-cadangan yang mungkin boleh diterima pakai bagi tujuan memperkembangkan projek ini pada masa hadapan. Seterusnya di akhir bab ini, penulis akan membuat kesimpulan bagi keseluruhan proses pembangunan BIST ALU.

7.2 MASALAH YANG DIHADAPI

Pada awal proses pembangunan BIST ALU, masalah pertama yang dihadapi adalah kesukaran memilih bahasa pengaturcaraan yang sesuai bagi tujuan membuat prototaip BIST ALU. Terdapat dua jenis bahasa pengaturcaraan yang dikenalpasti boleh digunakan untuk memprototaip BIST ALU iaitu *VHSIC Hardware Description Language* (VHDL) dan Verilog. Hasil penelitian yang dibuat, didapati bahasa pengaturcaraan VHDL lebih sesuai digunakan bagi tujuan pembangunan BIST ALU. Ini memandangkan tempoh pembangunan BIST ALU yang agak terhad dan ia lebih mudah untuk dikuasai.

Setelah menguasai aturcara VHDL, ia disimulasi menggunakan perisian *PeakFPGA Designer Suite FPGA Synthesis Edition 5.20c.* sungguhpun mengikut teori, bahasa pengaturcaraan VHDL agak mudah untuk dipelajari, namun proses membina dan mensimulasi tidaklah semudah seperti yang disangkakan. Walaupun secara teori tidak terdapat sebarang ralat sintak pada aturcara yang dibangunkan, tetapi setelah dikompil didapati terdapat sebahagian daripada sintak yang tidak disokong oleh perisian yang digunakan dan ini merupakan cabaran kepada penulis untuk mensimulasi BIST ALU.

Setelah mendapati aturcara adalah bebas ralat, masalah seterusnya yang dihadapi adalah ralat logik dimana hasil keluaran yang diperolehi gagal memenuhi spesifikasi yang telah ditetapkan. Akibatnya, aturcara yang dibina perlu dinilai semula bagi mengenalpasti punca kesilapan yang timbul.

7.3 KELEBIHAN BIST ALU

Sepertimana yang sudah sedia maklum, BIST ALU dibangunkan menggunakan bahasa pengaturcaraan VHDL. Aturcara BIST ALU dibangunkan secara berperingkat-peringkat mengikut modul. Setiap modul-modul yang dibangunkan boleh digunasemula. Selain itu, ia juga membenarkan sebarang pengubahsuaian dibuat samada untuk membuat pertambahan dan pengurangan terhadap sistem.

Selain itu, pembangunan prototaip BIST ALU juga dapat mengurangkan kos pembangunan. Ini kerana bahasa pengaturcaraan VHDL mampu memberi gambaran perlakuan (*behavior*) sebenar bagi litar bersepadu yang dibangunkan dan proses pengujian yang dilakukan dari semasa ke semasa dapat memastikan agar litar sebenar yang dihasilkan bebas ralat.

Penggunaan *Carry Lookahead Adder* dalam litar aritmetik telah mempercepatkan proses penambahan dibuat keatas nilai masukkan. Berbanding *Ripple Carry Adder, Carry Lookahead Adder* melaksanakan operasi penambahan secara selari dan ini mengurangkan nilai lengahan yang terdapat pada litar penambah penuh yang biasa digunakan dalam *Ripple Carry Adder.*

Dengan menggunakan perisian *PeakFPGA Designer Suite FPGA Synthesis Edition 5.20c,* hasil keluaran yang diperolehi boleh ditukar jenis nombornya kepada sebarang bentuk nombor samada binari, desimal, oktal

atau heksadesimal. Dengan ini proses penilaian hasil keluaran menjadi lebih mudah dilaksanakan. Pengguna tidak perlu mengambil masa yang lama untuk menukar jenis nombor bagi menilai output bagi hasil simlasi.

7.4 KEKURANGAN BIST ALU

Setelah dinilai, didapati penggunaan Unit Arithmetik dan Logik 8 bit (8 bit ALU) adalah tidak sesuai bagi *Design Under Test* (DUT) bagi *Built-In-Self-Test* (BIST). Ini kerana pergabungan litar 8 bit ALU dan BIST hanya akan menambah saiz bagi 8 bit ALU seterusnya menyebabkan lengahan pada masa pemproses DUT tersebut.

Walaubagaimanapun, bagi tujuan permulaan 8 bit ALU sesuai digunakan memandangkan saiznya yang kecil dan mudah untuk kita mengesan sebarang ralat yang timbul.

Selain itu, aturcara BIST yang dibangunkan tidak mampu menampung lebih daripada satu operasi aritmetik atau logik secara serentak. Hanya satu operasi mampu dilaksanakan dalam satu-satu masa dan ini mengakibatkan proses pengujian mengambil masa yang panjang.

7.5 CADANGAN

Hasil penilaian yang dibuat didapati bahawa 8 bit ALU bukanlah satu model yang sesuai untuk dijadikan DUT bagi litar BIST. Litar ini lebih sesuai digunakan dalam litar yang lebih besar seperti pemproses. Ini kerana rekabentuknya yang lebih rumit, besar dan kompleks membuatkan proses agak sukar dan ini akan melibatkan kos yang agak tinggi.

Sepertimana yang telah dinyatakan, litar BIST yang dibangunkan hanya mampu melaksanakan satu operasi aritmetik dan logik dalam satu-satu masa dan ini mengakibatkan masa pengujian bertambah. Bagi mengatasi masalah ini, dicadangkan agar konsep penimbal (*buffer*) digunakan. Dengan menggunakan penimbal, proses pengujian akan dilaksanakan secara berjujukan dan ini membolehkan beberapa operasi dilaksanakan serentak.

Seterusnya, seperti yang telah dinyatakan di awal pembangunan projek BIST ALU, projek ini hanya melibatkan proses pembinaan prototaip sehingga ke pringkat simulasi sahaja. Bagi tujuan penyelidikan, dicadangkan agar pihak Fakulti Sains Komputer dan Teknologi Maklumat (FSKTM) menyediakan peruntukan agar rekabentuk yang dibuat dapat dibangunkan sehingga ke peringkat pembentukkan cip. Dengan ini pelajar lebih mudah memahami dan membuat sendiri prototaip yang telah direkabentuk.

7.6 KESIMPULAN

Di era globalisasi masakini pembinaan litar bersepadu bukan lagi merupakan suatu isu yang baru. Malahan sekarang ini kebanyakkan barangan elektronik yang dihasilkan biasanya mempunyai litar bersepadu di dalamnya bagi memastikan ia mampu memenuhi kehendak pengguna. Penghasilan litar bersepadu menjadi bertambah rumit apabila saiz litar bertambah besar dan kompleks. Ini secara tidak langsung memberi kesan terhadap pengujian bagi litar tersebut dimana kos yang tinggi diperlukan bagi memastikannya mampu beroperasi dengan baik. Menyedari hal ini, rekabentuk BIST telah dibangunkan dengan menggunakan 8 bit ALU sebagai DUT.

Rekabentuk BIST ALU telah dibangunkan menggunakan pendekatan bawah-atas dengan menggunakan bahasa pengaturcaraan VHDL sebagai bahasa pengaturcaraan untuk memodelkannya. Didapati bahawa mengimplementasi BIST ALU tidaklah semudah apa yang dipelajari secara teori. Ini adalah akibat perisian yang digunakan iaitu *PeakFPGA Designer Suite FPGA Synthesis Edition 5.20c* tidak mampu menyokong aturcara yang digunakan.

Bagi memastikan litar BIST memenuhi objektif sebenar pembangunannya, terdapat tiga kaedah untuk membuat penilaian. Kaedah pertama adalah secara manual, kaedah ini agak sukar untuk diimplementasikan kerana ia mengambil masa yang agak panjang untuk membuat penilaian. Kaedah kedua adalah dengan menulis aturcara yang sama

dengan perlakuan BIST dengan menggunakan bahasa pengaturcaraan C. kaedah ini telah digunakan bagi menguji BIST ALU. Kaedah terakhir adalah dengan menggunakan cip sebenar sebagai DUT dalam litar BIST yang digunakan. Cip yang digunakan tersebut perlulah cip yang tepat hasil keluarannya dan bebas ralat.

Didapati juga bahasa pengaturcaraan VHDL adalah amat sesuai digunakan bagi tujuan memodelkan prototaip BIST ALU. Ini kerana ia mampu mewakili perlakuan sebenar BIST ALU dan ini akan mengurangkan kos pembangunan BIST ALU sekiranya berlaku ralat atau kegagalan.

Secara keseluruhannya, aturcara BIST ALU berjaya dibangunkan tetapi ia tidak mampu membuktikan bahawa pendekatan BIST mampu digunakan bagi mengurangkan kos pembangunan litar bersepadu dengan mengurangkan kos pengujian. Ini kerana model DUT yang digunakan bersaiz kecil dan akibatnya pergabungan 8 bit ALU dan BIST telah mengurangkan prestasi pemprosesan ALU itu sendiri.

RUJUKAN

RUJUKAN

SINGKATAN	RUJUKAN
[CHAR 96]	Charles H. Roth, Jr, "*Digital Systems Design Using VHDL*", PWS Publishing Company. 1996
[IBRA 95]	Ibrahim Abu Bakr. M, "A Built-In-Self-Testable Bit-Slice Processor", Universitiy Malaya. 1995
[MORR 79]	M. Morris Mano, " *Digital Logic And Computer Design*", Prentice Hall International Inc. 1979
[MORR 94]	M. Morris Mano, "*Logic & Computer Design Fundamental*", Prentice Hall International Inc. 1994
[PETE 96]	Peter J. Ashenden, "*The Designer's Guide To VHDL*", Morgan Kaufman Publisher Inc. 1996
[SELL 99]	P. Sellapan, "*C++ Through Example – Object Oriented Programming*", Istiramai Enterprise Sdn. Bhd. 1999
[WILL 00]	William Stallings, " *Computer Organization And Architecture*", Prentice Hall International Inc. 2000
[ZAHA 94]	Zahari Mohamad Darus, Iftekhar Ahmad, "*Rekabentuk VLSI Untuk Kebolehujian*", Dewan Bahasa Dan Pustaka. 1994
[NOOR 01]	Mohamad Noorhan Masrek, Safawi Abdul Rahman, Kamarul Arifin Abdul Jalis, "*Analisis & Rekabentuk Sistem Maklumat*", Mc Graw Hill. 2001

LAMPIRAN I

MANUAL PENGGUNA

MANUAL PENGGUNA

KANDUNGAN

1.0 PENGENALAN 1

2.0 BIST ALU 2

3.0 UNIT ARITMETIK DAN LOGIK 8 BIT 3

4.0 REKABENTUK BIST ALU 5

1.0 PENGENALAN

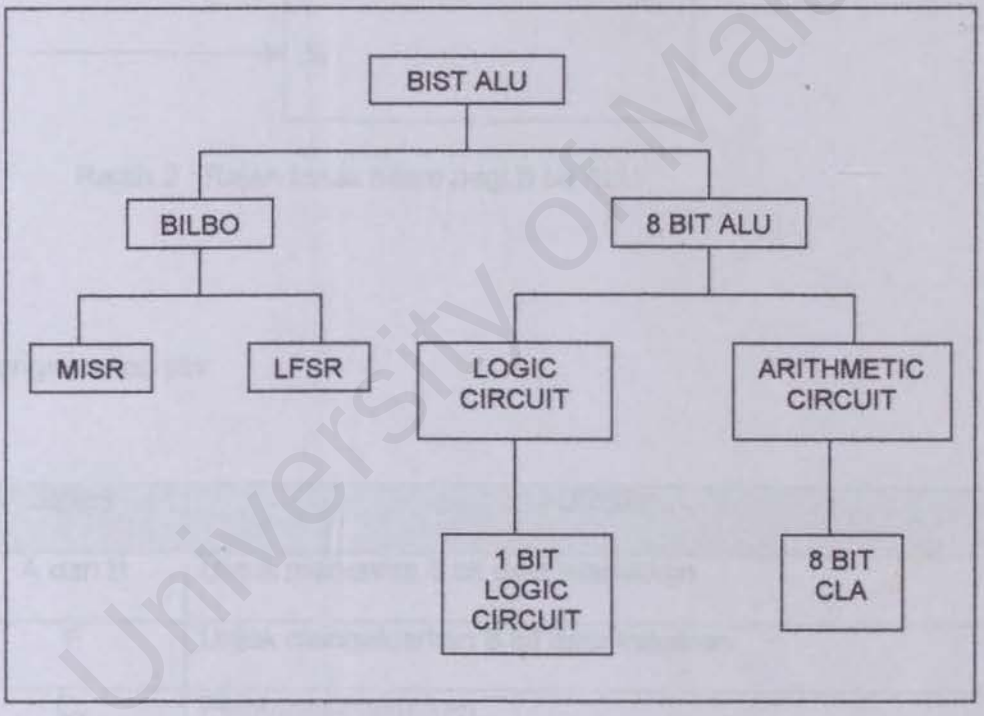
Built-In-Self-Test (BIST) merupakan salah satu jalan penyelesaian bagi memastikan pengeluaran kos pengujian yang minima. BIST merupakan salah satu teknik *Design For Testability* (DFT) yang mana litar BIST tersebut digabungkan bersama litar yang ingin diuji. Dengan ini sebarang ralat atau kecacatan yang timbul dapat dikesan diperingkat awal dan secara tidak langsung ia akan mengurangkan kos penyelenggaraan terhadap cip tersebut.

Sebagai titik permulaan bagi menghasilkan BIST, *Arithmetic Logic Unit* (ALU) 8 bit telah digunakan sebagai unit pengujian. Namun begitu, BIST ini juga boleh diaplikasikan ke atas mana-mana litar bersepadu yang lebih besar dan kompleks.

Built-In-Self-Test Arithmetic Logic Unit (BIST ALU) merupakan salah satu teknik rekabentuk yang menggunakan ALU sebagai unit untuk diuji Aplikasi ini membenarkan mengujian dilakukan ke atas ALU dengan menggabungkan BIST bersama litar ALU sebagai satu komponen.

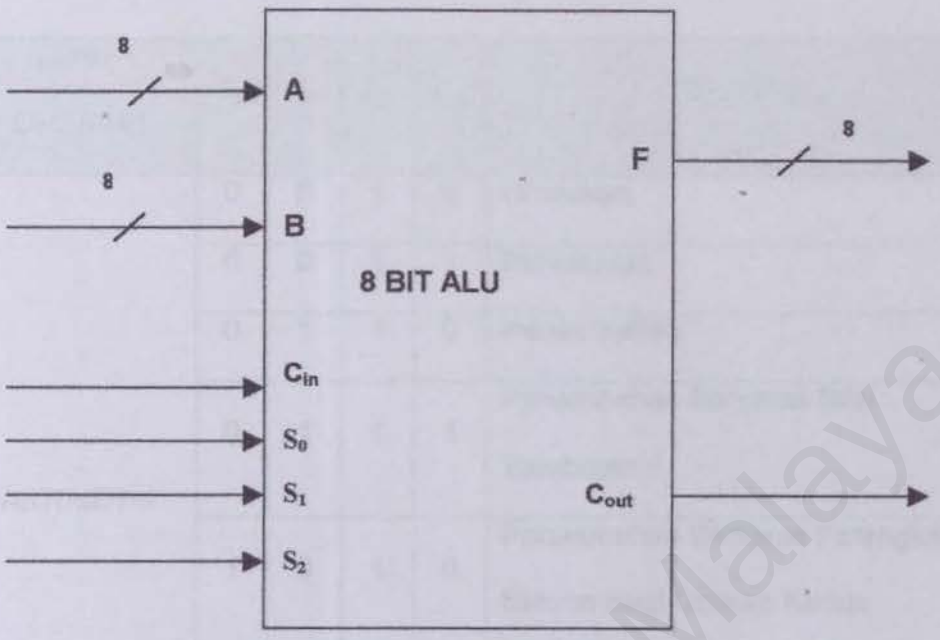
2.0 BIST ALU

BIST ALU dibangunkan menggunakan konsep hirarki dimana modul-modul yang berada pada hirarki terendah akan dibangunkan terlebih dahulu dan seterusnya diuji. Setelah dipasti bahawa modul tersebut bebas ralat, barulah ia akan diintegrasikan secara berhirarki sehingga menghasilkan satu sistem yang lengkap. Rajah 1 menunjukkan hirarki bagi pembinaan BIST ALU.



Rajah 1: Hirarki pembinaan BIST ALU.

3.0 UNIT ARITMETIK DAN LOGIK 8 BIT



Rajah 2 : Rajah kotak hitam bagi 8 bit ALU

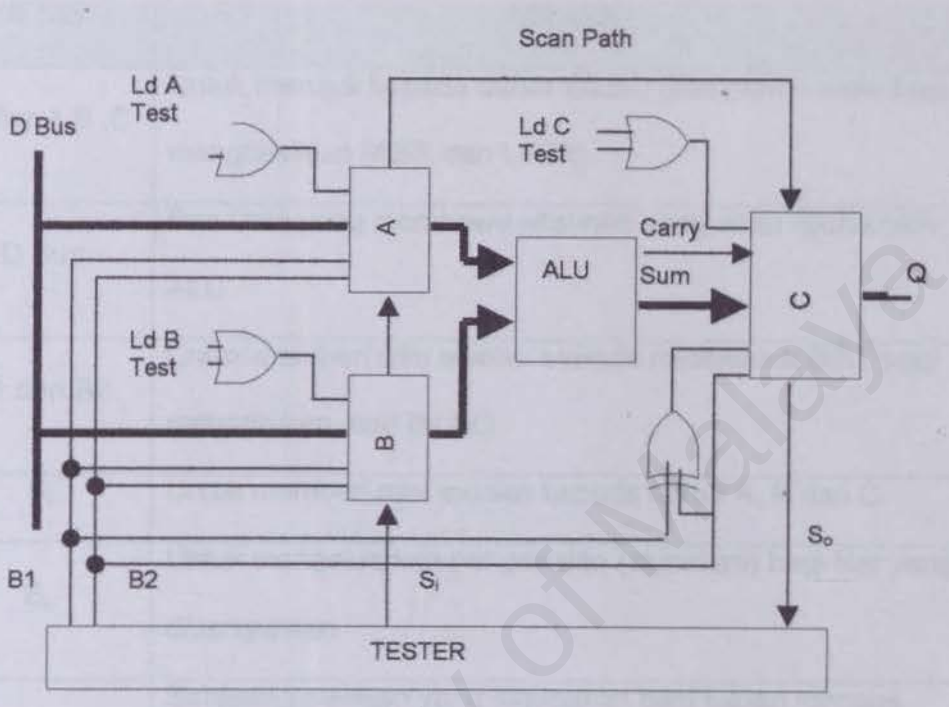
Fungsi setiap pin:

JENIS	FUNGSI
A dan B	Untuk menerima 8 bit data masukkan
F	Untuk mengeluarkan 8 bit data keluaran
C_{in}	Pembawa masukkan
C_{out}	Pembawa keluaran
S_0, S_1, S_2	Untuk menentukan mod pengoperasian yang ingin dilaksanakan.

Mod pengoperasian bagi Unit Aritmetik dan Logik 8 bit:

JENIS OPERASI	S_0	S_1	S_2	C_{in}	OPERASI
ARITMETIK	0	0	1	0	Umpukan
	0	0	1	1	Penokokan
	0	1	1	0	Penambahan
	0	1	1	1	Penambahan Bersama Nilai Sambutan 1
	1	0	1	0	Penambahan Bersama Pelengkap Satuan Bagi Operan Kedua
	1	0	1	1	Penolakkan
	1	1	1	0	Pengurangan
	1	1	1	1	Umpukan
LOGIK	0	0	0	X	Get ATAU
	0	1	0	X	Get X-ATAU
	1	0	0	X	Get DAN
	1	1	0	X	Get TAK

5.0 REKABENTUK BIST ALU



Rajah 3 : Rekabentuk BIST ALU

Fungsi setiap komponen:

JENIS	FUNGSI
Daftar A,B ,C	Untuk merujuk kepada daftar BILBO (komponen asas bagi menghasilkan MISR dan LFSR)
D Bus	Bas Data yang membawa nilai-nilai yang akan dijana oleh ALU
B1 dan B2	Untuk memberi nilai awalan kepada rekabentuk BIST bagi menentukan mod BILBO
S_i	Untuk memberi nilai awalan kepada daftar A, B dan C
S_o	Untuk mengeluarkan pengenalan (*signature*) bagi litar yang dibangunkan
Tester	Sebarang medium yang digunakan bagi tujuan menguji sgnature yang diperolehi daripada S_o.

Mod bagi daftar BILBO:

B1	B2	MOD OPERASI
0	0	Daftar Anjakan (*Shift Register*)
0	1	*Pseudo-Random Pattern Generator* (PRPG)
1	0	Normal
1	1	*Multiple-Input Signature Register* (MISR)

LAMPIRAN 2

KOD SUMBER

```
-- BIST system with an 8 bit ALU

library ieee;
use ieee.std_logic_1164.all;

entity bist1 is
    port (
        reseth : in std_logic;
        clk : in std_logic;
        LdA : in std_logic;
        LdB : in std_logic;
        LdC : in std_logic;
        Ci : in std_logic;
        Si : in std_logic;
        bilbo_mode : in std_logic_vector(1 downto 0);
        m_sel0 : in std_logic;
        m_sel1 : in std_logic;
        m_sel2 : in std_logic;
        Dbus : out std_logic_vector(7 downto 0);
        So : out std_logic;
        Output : inout std_logic_vector(7 downto 0)  );
end entity bist1;

architecture bist1_behav of bist1 is

    component ALU8BIT is
    port (
        ValA     : in std_logic_vector(7 downto 0);
        ValB     : in std_logic_vector(7 downto 0);
        Carry        : in std_logic;
        mod0     : in std_logic;
        mod1     : in std_logic;
            mod2     : in std_logic;
        Val_out      : out std_logic_vector(7 downto 0)  );
    end component;

    component bilbo is
        generic (nbits : natural range 8 to 16 := 8);
        port (
            reseth       : in std_logic;
            clk      : in std_logic;
            ce           : in std_logic;
            si           : in std_logic;
            bilbo_mode   : in std_logic_vector(1 downto 0);
            z            : in std_logic_vector(nbits downto 0);
            so           : out std_logic;
            q            : inout std_logic_vector(nbits downto 0)  );
    end component;

    signal Aout, Bout : std_logic_vector(7 downto 0);
    -- signal Cin : std_logic_vector(7 downto 0);
    signal Sum : std_logic_vector(7 downto 0);
    signal ACE, BCE, CCE, Test, S1, S2 : std_logic;
    signal CB1 : std_logic_vector(1 downto 0);

begin

    Test <= not bilbo_mode(1) or bilbo_mode(0);
    ACE <= Test or LdA;
    BCE <= Test or LdB;
    CCE <= Test or LdC;
    CB1 <= ( bilbo_mode(1) xor bilbo_mode(0) ) & bilbo_mode(0);

    RegA : bilbo
            port map (reseth, clk, ACE, S1, bilbo_mode, Dbus, S2, Aout);

    RegB : bilbo
            port map (reseth, clk, BCE, Si, bilbo_mode, Dbus, S1, Bout);

    RegC : bilbo
            port map (reseth, clk, CCE, S2, CB1, Sum, So, Output);

    UUT : ALU8BIT
            port map (Aout, Bout, Ci, m_sel0, m_sel1, m_sel2, Sum);
```

```
-- An 8 bit BILBO Register

library ieee;
use ieee.std_logic_1164.all;

entity bilbo is

  generic (nbits: natural range 8 to 16 := 8);
  port(
    reseth      : in std_logic;
    clk         : in std_logic;
    ce          : in std_logic;
    si          : in std_logic;
    bilbo_mode  : in std_logic_vector(1 downto 0);
    z           : in std_logic_vector(nbits downto 1);
    so          : out std_logic;
    q           : inout std_logic_vector(nbits downto 1) );

end entity bilbo;

architecture behavior of bilbo is

  signal s_fb      : std_logic;
  signal s_q       : std_logic_vector(nbits downto 1);

  -- bilbo modes
  constant c_shift : std_logic_vector(1 downto 0):="00";
  constant c_prpg  : std_logic_vector(1 downto 0):="01";
  constant c_normal: std_logic_vector(1 downto 0):="10";
  constant c_misr  : std_logic_vector(1 downto 0):="11";


begin

  s_fb <= q(2) xor q(3) xor q(4) xor q(nbits) when (nbits = 8);

  bilbo_reg : process(clk, reseth)
  begin
   if (reseth = '1') then
      s_q   <= (others =>'0');
      so    <='0';

   elsif (clk'event and clk = '1') then
     if (ce = '1') then

       if (bilbo_mode = c_shift) then
         s_q <= q(nbits-1 downto 1) & si;
       end if ;

       if (bilbo_mode = c_prpg) then
         s_q <= q(nbits-1 downto 1) & s_fb ;
       end if ;

       if (bilbo_mode = c_normal) then
         s_q <= z;
       end if ;

       if (bilbo_mode = c_misr) then
         s_q <= (q(nbits-1 downto 1) & s_fb) xor z(nbits downto 1);
       end if ;

       so <= s_q(nbits);

    end if ;

  end if ;

  end process bilbo_reg;

q  <= s_q;

end architecture behavior;
```

```
        P1: ARITH8BIT port map (x => ValA, y => ValB, Cin => Carry, s0 => mod0, s1 => mod1, sum
=> arith);
        P2: LOGIC_CIRCUIT port map (in1 => ValA, in2 => ValB, sel0 => mod0, sel1 => mod1, output
=> logic);
        P3: host port map (operator => mod2, arith_out => arith, logic_out =>logic, result =>
Val_out);

    end top_level;
```

A:\BIST_SYSTEM\ARITH8BIT.VHD

```
-- ********************************************
-- *                                          *
-- *Item          :   8 bit arithmetic circuit  *
-- *                                          *
-- *File name     :   ARITH8BIT.vhd             *
-- *                                          *
-- *Author        :   Sunaila Ab Aziz           *
-- *                                          *
-- ********************************************


library ieee;
use ieee.std_logic_1164.all;

entity ARITH8BIT is -- penglaytiharan entiti bagi 8 bit arithmetic circuit
    port (
        x       : in std_logic_vector(7 downto 0);
        y       : in std_logic_vector(7 downto 0);
        Cin     : in std_logic;
        s0      : in std_logic;
        s1      : in std_logic;
        sum     : out std_logic_vector(7 downto 0)

    );

end ARITH8BIT; -- tamat penglaytiharan entiti


architecture BEHAVIOR of ARITH8BIT is
-- penglaytiharan perlakuan (behavior) bagi 8 bit arithmetic circuit


-- penglaytiharan komponen-komponen yang terdapat dalam 8 bit arithmetic circuit
-- (modul CLA + modul ALU_LOGIC_CIRCUIT)

component ALU_LOGIC_CIRCUIT is -- penglaytiharan component1 (modul ALU_LOGIC_CIRCUIT)
    port (
        M: in std_logic;-- penglaytiharan nilai masukan & keluaran bagi alu_logic_circuit
        N: in std_logic;
        O: in std_logic;
        F: out std_logic
        );

end component;-- tamat penglaytiharan component1

component CLA is -- penglaytiharan component2 (modul CLA)
    port (
        a_in : in std_logic_vector(7 downto 0); -- penglaytiharan pin masukkan & keluaran
        b_in : in std_logic_vector(7 downto 0);
        c_in: in std_logic;
        c_out    : out std_logic_vector(7 downto 0)
        --carry : out std_logic
        );
end component;-- tamat penglaytiharan component2


signal input_b : std_logic_vector(7 downto 0);
-- signal yang diperlukan bagi menghubungkan component yang terdapat dalam 8 bit arithmetic
circuit


--proses pemetaan component & arith8bit
begin

    logic0 : ALU_LOGIC_CIRCUIT port map (M => y(0), N => s0, O =>s1, F=> input_b(0));
    logic1 : ALU_LOGIC_CIRCUIT port map (M => y(1), N => s0, O =>s1, F=> input_b(1));
    logic2 : ALU_LOGIC_CIRCUIT port map (M => y(2), N => s0, O =>s1, F=> input_b(2));
    logic3 : ALU_LOGIC_CIRCUIT port map (M => y(3), N => s0, O =>s1, F=> input_b(3));
    logic4 : ALU_LOGIC_CIRCUIT port map (M => y(4), N => s0, O =>s1, F=> input_b(4));
    logic5 : ALU_LOGIC_CIRCUIT port map (M => y(5), N => s0, O =>s1, F=> input_b(5));
    logic6 : ALU_LOGIC_CIRCUIT port map (M => y(6), N => s0, O =>s1, F=> input_b(6));
    logic7 : ALU_LOGIC_CIRCUIT port map (M => y(7), N => s0, O =>s1, F=> input_b(7));
    logic8 : CLA port map (a_in => x, b_in => input_b, c_in=>Cin, c_out => sum);
```

A:\BIST_SYSTEM\CLA.VHD

```
-- ********************************************
-- *                                          *
-- *Item         :   Carry Lookahead Adder     *
-- *                                          *
-- *File name    :   CLA.vhd                   *
-- *                                          *
-- *Author       :   Suhaila Ab Aziz           *
-- *                                          *
-- ********************************************


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


entity CLA is
    port (
        a_in : in std_logic_vector(7 downto 0); -- penglaytiharan pin masukkan & keluaran
        b_in : in std_logic_vector(7 downto 0);
        c_in: in std_logic;
        c_out    : out std_logic_vector(7 downto 0)

        );

end CLA;

architecture BEHAVIOR of CLA is

-- signal (internal wiring bagi CLA)
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 0);
signal  carry_in :   std_logic;


begin

   sum(0) <= a_in(0) xor b_in(0) xor c_in; --PFA

        g <= a_in and b_in; -- PFA
        p <= a_in xor b_in;

    P1: process(g,p,c_internal) -- proses 1 -> operasi dalam partial CLA
    begin
        carry_in <= c_in;     -- untuk initialize nilai awalan signal carry_in

        c_internal(1) <= g(0) or (p(0) and carry_in); -- nilai pertama logik untuk operasi
dalaman bagi CLA

            FOR i In 1 TO 6 LOOP
                c_internal (i+1) <= g(i) or (p(i) and c_internal(i)); -- logik untuk operasi
dalam CLA
            END LOOP;

    end process P1; -- tamat proses 1


    P2: process (c_out,p,c_internal)-- proses 2 -> operasi mengeluarkan output bagi CLA

    begin

    c_out (0) <= p(0) xor carry_in; -- nilai output pertama bagi CLA

    for value in 1 to 7 loop
        c_out(value) <= p(value) xor c_internal(value); -- nilai output ke 2 hingga ke 8
    end loop;

    end process P2; -- tamat proses 2


end BEHAVIOR; -- tamat penglaytiharan behavior
```

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

entity host is
    port (
        operator : in STD_LOGIC;
        arith_out    : in STD_LOGIC_VECTOR(7 downto 0);
        logic_out    : in STD_LOGIC_VECTOR(7 downto 0);
        result       : out STD_LOGIC_VECTOR(7 downto 0)
    );
end host;

architecture host_arch of host is

--begin


constant c_arith : std_logic := '1';
constant c_logic  : std_logic := '0';


    begin


with operator select
  result <= arith_out when c_arith,
     logic_out when c_logic,
        "00000000" when others;
--  end process;

end host_arch;
```

```
-- ------------------------------------------
-- *
-- *Item         :   Logic Circuit (8 bit)    *
-- *                                          *
-- *File name    :   logic_circuit.vhd        *
-- *                                          *
-- *Author       :   Suhaila Ab Aziz          *
-- *                                          *
-- ------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

entity LOGIC_CIRCUIT is --penglsytiharan entiti bagi logic circuit
    port (
        in1: in std_logic_vector(7 downto 0);
        in2: in std_logic_vector(7 downto 0);
        sel0: in std_logic;
        sel1: in std_logic;
        output: out std_logic_vector(7 downto 0));

end LOGIC_CIRCUIT; -- tamat penglsytiharan entiti


architecture behav of LOGIC_CIRCUIT is

begin


process(sel0,sel1)

begin


    -- pernyataan if-else -> memilih jenis operasi logik yang ingin dibuat
    -- mod logik bergantung kepada nilai S0 dan S1 yang diberi

        if (sel0 = '0' and sel1 = '0') then -- operasi gate OR
                output <= in1 or  in2;

        end if;


        if (sel0 = '0' and sel1 = '1') then  -- operasi gate XOR
                    output <= in1 xor  in2;


        end if;


        if (sel0 ='1' and sel1 = '0') then
                    output <= in1 and  in2;


        end if;

        if (sel0 = '1' and sel1 = '1') then
                output <= not in1; -- operasi gate NOT


        end if; -- tamat pernyataan if-else

        end process;


    end behav; -- tamat penglsytiharan behav of logic_circuit
```