BAB 3

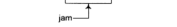
PENDARAB TITIK APUNGAN 32BIT BERTALIAN PAIP

3.1 : PENGENALAN

Bab ini menerangkan struktur PTA yang menggunakan talian paip dan format IEEE Standard 754 (di mana terdapat sedikit pengubahsuai kedudukan bagi memudahkan permerhatian dalam tatatanda perenambelas iaitu 8bit pertama untuk eksponen, 23 bit seterusnya untuk mantisa dan 1 bit untuk bit tanda). Ia boleh mengendali pendaraban di antara nombor positif dan negatif menggunakan tatatanda pelengkap duaan. Nilai masukan bagi `nombor_a` dan `nombor_b` diberi oleh persamaan IEEE berikut

$$n = (-1)^t 1.m\ 2^{e-127}$$

di mana t ialah bit tanda, m adalah mantisa dan e merupakan $e_{sebenar} + 127$ atau $(1 \le e \le 254)$ [13]. Semua data yang berkaitan adalah dalam jenis piawai IEEE iaitu pakej `std_logic_1164`. Keseluruhan senibina PTA menggunakan VHDL dengan peralatan Synopsys dan Xilinx. Perbinaan bermula dengan pembinaan rekabentuk teras (*core design*) seperti Rajah 3.1 dan Program 3.1 .

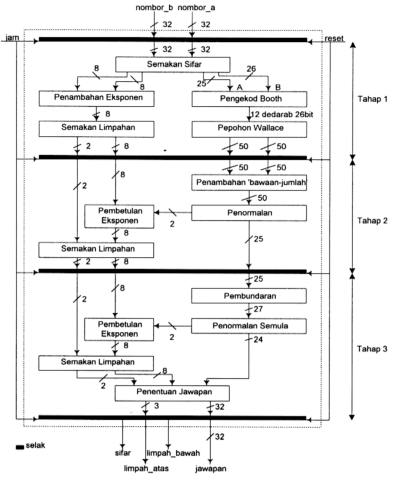


Rajah 3.1: Rekabentuk teras

```
--Program 3.1: Isyarat Masukan dan Keluaran
ENTITY pdrbta IS
  PORT(jam              : IN std_logic;
       reset            : IN std_logic;
       nombor_a         : IN std_logic_vector(32 downto 1);
       nombor_b         : IN std_logic_vector(32 downto 1);
       sifar            : OUT std_logic;
       limpah_atas      : OUT std_logic;
       limpah_bawah     : OUT std_logic;
       jawapan          : OUT std_logic_vector(32 DOWNTO 1));
END pdrbta;
```

3.2 Senibina

Senibina ini (rujuk Rajah 3.2) mengandungi tiga tahap talian paip yang menggunakan flip-flop tak-segerak [30][32]. Rekabentuk tiga kitaran dipilih untuk cuba meningkatkan kelajuan jam. Tahap pertama mengandungi penambahan eksponen dan semakan limpahan yang dilaksanakan serentak dengan pengekod Booth dan pepohon Wallace. Tetapi sebelum itu semakan sifar dilaksanakan terlebih dahulu. Manakala tahap kedua mengandungi operasi penambahan 'bawaan-jumlah' yang terhasil daripada pepohon Wallace pada penghujung kitaran pertama, pernormalan beserta pembetulan eksponen dan semakan limpahan eksponen. Tahap ketiga pula terdiri daripada operasi pembundaran, penormalan semula, pembetulan eksponen, semakan limpahan eksponen dan penentu keluaran.



Rajah 3.2: Organisasi pendarab titik apungan 32bit.

3.2.1 Tahap Pertama

`Semakan Sifar`

Program 3.2 melaksanakan pengesanan atau penyemakan samada data masukan yakni `p` dan `q` berisyarat `datasifar` yakni "`00000000 00000000 00000000 00000001`" di mana bit 1 ke 8 ialah eksponen dan bit 31 ke 9 adalah mantisa dan bit ke32 adalah bit-tanda. Isyarat `q` adalah isyarat yang sama dengan `nombor_b` dan ia bersifat sebagai pekali manakala `p` adalah isyarat yang sama dengan `nombor_a` dan ia bersifat sebagai nombor yang didarab.

```
--Program 3.2: Semakan sifar
--semakan untuk nombor_a
a_sifar1 <='1' when p = datasifar else '0';
--semakan untuk nombor_b
b_sifar1 <='1' when q = datasifar else '0';
```

Selain daripada semakan sifar ia juga melaksanakan perlanjutan bit-tanda untuk kedua-dua mantisa, di mana untuk `mantisa_p` mempunyai dua perlanjutan bit tanda dan `mantisa_q` tiga perlanjutan bit-tanda dengan satu bit 0 dikenakan pada bit kurang bererti seperti dalam Program 3.3

```
--Program 3.3 : Perlanjutan bit-tanda
--asem dan bsem ialah songsangan bit-tanda
asem <= '0' when a_sifar1 = '1' else not p(32);
bsem <= '0' when b_sifar1 = '1' else not q(32);
r <=  p(32) & asem & p(31 downto 9);              --mantisa_p
s <=  q(32) & q(32) & bsem & q(31 downto 9) & '0'; --mantisa_q
```

Penambahan Eksponen dan Semakan Limpahan

Program 3.4 menunjukan operasi penambahan eksponen dengan menggunakan fungsi yang sedia-ada dalam pengkompilasi VHDL Synopsys [28].

```
--Program 3.4: Program Penambahan eksponen
jeks1 := eks_a  + eks_b - "01111111";
```

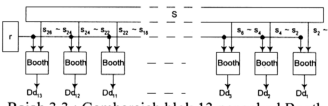
Persamaan dalam Program 3.4 juga melakukan penolakan nilai pincang 127 atau `"01111111"`. Di dalam *Synopsys DesignWare* (SDW) terdapat dua jenis penambah iaitu penambah bawaan riak dan penambah peninjau bawaan. Pengkompilasi dengan secara automatik memilih salah satu daripada penambah tersebut dengan berdasarkan kekangan yang dikenakan [28][33]. Hasiltambah eksponen tersebut seterusnya

disemak untuk memeriksa samada limpahan wujud atau tidak. Semakan limpahan ditentukan dengan menganalisa kesamaan bit kelapan `eks_a`, `eks_b` dan `jeks1` atau `jeks1 = "00000001"` seperti dalam Program 3.5.

```
--Program 3.5: Semakan Limpahan pada Tahap Pertama
if ((eks_a(7) = eks_b(7)) and (eks_a(7) /= jeks1(7))) or
     (jeks1 = "00000001") then
    latsl := not eks_a(7); --Limpahan Atas
    lbwhl := eks_a(7);     --Limpahan Bawah
end if;
```

Pengekod Booth

Terdapat duabelas pengekod yang menghasilkan tigabelas dedarab 26bit *Dd* secara serentak sila rujuk Rajah 3.3 di mana `r` dan `s` adalah daripada bahagian semakan sifar.



Rajah 3.3 : Gambarajah blok 13 pengekod Booth

Jadual 3.1 : Algoritma Booth Tertib Kedua

Masukan			Fungsi	Isyarat Keluaran	
				Dd	*T*
S_{j+1}	S_j	S_{j-1}			
0	0	0	0	Rentetan bit 0	0
0	0	1	$+r$	r dan dilanjutkan satu bit-tanda	0
0	1	0	$+r$	r dan dilanjutkan satu bit-tanda	0
0	1	1	$+2r$	Anjakan r kekiri satu bit dan ruang tersebut diganti dengan bit 0	0
1	0	0	$-2r$	Anjakan songsangan r kekiri satu bit dan ruang tersebut diganti dengan bit 1	1
1	0	1	$-r$	Songsangkan r dan lanjutan satu bit tanda	1
1	1	0	$-r$	Songsangkan r dan lanjutan satu bit tanda	1
1	1	1	*0*	Rentetan bit 0	0

Dedarab 26bit didapati kerana perlanjutan bit-tanda dalam situasi "001", "010", "101" dan "110", dan disebabkan anjakan satu bit dalam situasi "011" dan "100" seperti dalam Jadual 3.1. Isyarat *t* ialah satu isyarat untuk menghasilkan *Dd* dalam pelengkap duaan iaitu $Dd + t$. Penambahan ini dilakukan dalam pepohon Wallace untuk mengurangkan laluan kritikal, yakni sekiranya penambahan tersebut dilakukan dalam pengekod Booth, ia akan menghasilkan satu unit penambah bagi setiap pengekod Booth. Program pengekod Booth ditulis dalam bentuk pakej untuk mengurangkan penggunaan ingatan berbanding dalam bentuk komponen.

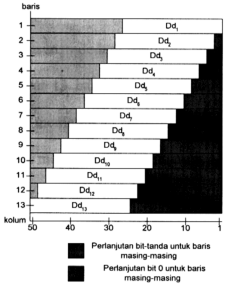
```
--Program 3.6: Pengekod Booth
case d_m is --d_m ialah data masukan tiga bit

   when "001" | "010" =>                          --fungsi +r
    x_k <= x_m(24)&x_m;
    n_1 <= '0';
   when "011" =>                                  --fungsi +2r
    x_k <= x_m&'0';
    n_1 <= '0';
   when "100" =>                                  --fungsi -2r
    for indek in 1 to 25 loop
    x_k(indek) <= not x_m(indek - 1);
    end loop;
    x_k(0) <= '1';
    n_1 <= '1';
   when "101" | "110" =>                          --fungsi -r
    for indek in 0 to 24 loop
    x_k(indek) <= not x_m(indek);
    end loop;
    x_k(25) <= not x_m(24);
    n_1 <= '1';
   when others =>                                 --sifar
    x_k <= (x_k'range => '0');
    n_1 <= '0';
 end case;
```

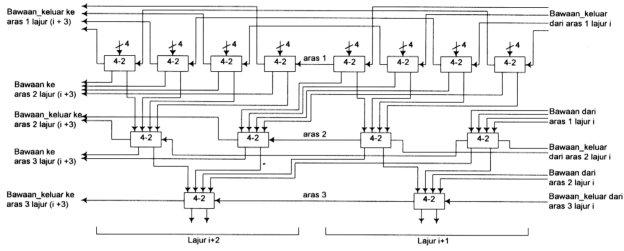
Program 3.6 merupakan aturcara untuk pengekod Booth di mana `d_m` ialah *s*, `x_m` ialah *r*, `x_k` ialah *Dd* dan `n_1` ialah *t*. Manakala Program 3.7 pula merupakan aturcara untuk Rajah 3.3, di mana keluaran `Dd` merupakan data dalam bentuk matrik 1×26.

```
--Program 3.7: Aturcara untuk penjanaan dedarab
```

```
booth (r, s(2 downto 0), t(1), Dd(1)(25 downto 0));
booth (r, s(4 downto 2), t(2), Dd(2)(27 downto 2));
booth (r, s(6 downto 4), t(3), Dd(3)(29 downto 4));
booth (r, s(8 downto 6), t(4), Dd(4)(31 downto 6));
booth (r, s(10 downto 8), t(5), Dd(5)(33 downto 8));
booth (r, s(12 downto 10), t(6), Dd(6)(35 downto 10));
booth (r, s(14 downto 12), t(7), Dd(7)(37 downto 12));
booth (r, s(16 downto 14), t(8), Dd(8)(39 downto 14));
booth (r, s(18 downto 16), t(9), Dd(9)(41 downto 16));
booth (r, s(20 downto 18), t(10), Dd(10)(43 downto 18));
booth (r, s(22 downto 20), t(11), Dd(11)(45 downto 20));
booth (r, s(24 downto 22), t(12), Dd(12)(47 downto 22));
booth (r, s(26 downto 24), t(13), Dd(13)(49 downto 24));
```



Rajah 3.4: Susun-atur bit-bit dedarab



Rajah 3.5: Sambungan antara penambah 16-2

Pepohon Wallace

Rajah 3.4 menunjukkan tatasusunan dedarab-dedarab berserta dengan perlanjutan bit-tanda dan sifar. Ia mempunyai 13 baris 50 lajur kedudukan bit. Untuk menambah tigabelas baris dedarab ini, ia memerlukan penambah yang bersifat 16-2 (rujuk Rajah 3.5), ini dilaksanakan dengan menggunakan pemampat 4-2 daripada Rajah 2.7. Satu bit isyarat pelengkap duaan t dan dua bit selebihnya diisyaratkan sebagai bit 0, di dalam proses sintesis, get-get logik yang tidak berkaitan atau tak memberi makna kepada fungsian akan dibuang atau dihapus secara automatik. Penambahan dedarab dengan t hanya berlaku pada aras pertama setiap lajur $j+1$ di mana $j = 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22$ .Oleh kerana terdapat 50 lajur maka 50 penambah 16-2 diperlukan untuk melaksanakan penambahan kesemua dedarab yang dihasilkan oleh pengekod Booth.

Rajah 3.5 juga menunjukkan sambungan penambah 16-2 di antara lajur. Jika diperhatikan terdapat tiga aras penambah 4-2 dalam penambah 16-2, aras-aras ini

penting kepada isyarat bawaan dan bawaan-keluar supaya sambungannya tidak menjadi riak (*ripple*) [12][34]. Setiap bawaan daripada aras k lajur i akan disambung ke aras $k+1$ lajur $i+1$ di mana $k=1,2\&3$ dan $i=1,2,3,\cdots,47,48,49$. Manakala bawaan-keluar disambung ke aras yang sama pada lajur yang berikutnya, sekali imbas ia nampak seperti riak tetapi jika diperhatikan dari segi gambarajah litar penambah 4-2 (Rajah 2.6 atau 2.7) ia adalah selari. Untuk lajur yang ke50 bawaan dan bawaan-keluar diabaikan.

```
--Program 3.8: Perlanjutan bit-tanda dan perlanjutan bit-sifar
--Bit-tanda
--m = 25 dan n = 26
lanjutant : for k in 1 to n/2 generate
                lanjtanda :for i in m+n-1 downto m+k*2-1 generate
                           Dd(k)(i) <= Dd(k)(m+k*2-2);
                           end generate;
            end generate;
--Bit-sifar
lanjutans : for k in 1 to n/2 - 1 generate
                lanjsifar :for i in 0 to k*2-1 generate
                           Dd(k+1)(i) <= '0';
                           end generate;
            end generate;

--Program 3.9: Susun-atur dedarab dalam bentuk matrik 13×1
--l = m+n
kolum: for kol in 1 to l-1 generate
          gen : for z in 1 to n/2 generate
                semz(kol)(z) <= Dd(z)(kol-1);
                end generate;
       end generate;
```

Program 3.8 menunjukkan aturcara perlanjutan bit-tanda, perlanjutan bit-sifar untuk setiap baris dan Program 3.9 melaksanakan susun-atur keseluruhan bit dalam Rajah 3.4 untuk membentuk data bermatrik 13×1 untuk digunakan dalam Program 3.10

yang merupakan sebahagian daripada aturcara sambungan antara penambah 16-2. Isyarat `semz` dan `t` adalah masukan dan `bw` dan `jum` adalah keluaran manakala `bkisy` ialah bawaan-keluar dan `cryisy` ialah bawaan.

```
--Program 3.10 : Pepohon Wallace.
p16_2 (semz(1), t(1), bkisy(0), cryisy(0), bkisy(1), cryisy(1), 1
                    bw(1), jum(1));
p16_2 (semz(2), '0', bkisy(1), cryisy(1), bkisy(2), cryisy(2),
                    bw(2), jum(2));
p16_2 (semz(3), t(2), bkisy(2), cryisy(2), bkisy(3), cryisy(3),
                    bw(3), jum(3));
.
.
.
p16_2 (semz(48), '0', bkisy(47), cryisy(47), bkisy(48), cryisy(48),
                    bw(48), jum(48));
p16_2 (semz(49), '0', bkisy(48), cryisy(48), bkisy(49), cryisy(49),
                    bw(49), jum(49));
p16_2 (semz(50), '0', bkisy(49), cryisy(49), bkisy(50), cryisy(50),
                    bw(50), jum(50));
```

Program 3.7 hingga 3.10 boleh digabungkan menjadi Program 3.11, sungguhpun begitu ia memerlukan penggunaan ingatan yang besar. Jadi untuk mengurang penggunaan ingatan program 3.8 hingga 3.10 digunakan ketika proses sintesis.

```
--Program 3.11: Gabungan program 3.7 hingga 3.10
dedarab : for k in 1 to n/2 generate
         booth (r, s(k*2 downto k*2-2), t(2*k-1), dd(k)(m+k*2-2 downto k*2-2));
               lanjtanda :for i in m+n-1 downto m+k*2-1 generate
                          dd(k)(i) <= dd(k)(m+k*2-2);
                          end generate;
         t(2*k) <= '0';
         end generate;
lanjutan : for k in 1 to n/2 - 1 generate
               lanjsifar :for i in 0 to k*2-1 generate
                          dd(k+1)(i) <= '0';
                          end generate;
```

```
          end generate;
lanjplkp : for i in 27 to 1 generate
          t(i) <= '0';
          end generate;

    kolum: for kol in 1 to 1 generate
      satunit : block
          signal semz : std_logic_vector(13 downto 1);
        begin
          gen : for z in 1 to n/2 generate
                semz(z) <= dd(z)(kol-1);
                end generate;
           p16_2 (semz, t(kol), bkisy(kol-1), cryisy(kol-1), bkisy(kol), cryisy(kol),
                  bw(kol), jum(kol));
               end block;
          end generate;
```

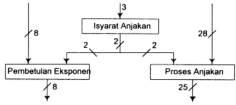
3.2.2 Tahap Kedua

Penambahan Bawaan dan Jumlah

Penambahan ini juga menggunakan fungsi penambahan yang sedia-ada dalam pengkompilasi VHDL Synopsys seperti dalam Program 3.12.

```
--Program 3.12: Penambahan Bawaan dan Jumlah
man_1(50 downto 2) <= jumlah(50 downto 2) + bawaan(49 downto 1);
man_1(1) <= jumlah(1);
```

Isyarat `jumlah(1)` tidak perlu melakukan penambahan kerana ia sudah merupakan bit hasildarab dan bawaan yang kelimapuluh `bawaan(50)` diabaikan. Isyarat `man_1` merupakan hasildarab (jawapan kepada pendaraban) yang mengandungi 50bit.



Rajah 3.6 : Penormalan dan pembetulan eksponen

Penormalan

Setelah mendapatkan hasildarab tersebut, bit48 hingga ke bit50 dalam isyarat `man_1` digunakan untuk mendapat isyarat `anjakan_1` (rujuk Program 3.13) bagi menentukan anjakan yang diperlukan dalam proses anjakan dan pembetulan eksponen yang dijalankan serentak (lihat Rajah 3.6). Isyarat `man_norm_1` merupakan isyarat keluaran setelah proses anjakan (rujuk Program 3.14) dilaksanakan dan bilangan bitnya ialah 25bit, di mana bit ke25 ialah bit-tanda.

```
--Program 3.13: Isyarat anjakan_1
anjakan_1 <= "10" when (man_1(50) xor man_1(49)) = '1' else
             "01" when (man_1(49) xor man_1(48)) = '1' else
             "00";
--Program 3.14: Proses anjakan
with anjakan_1 select
    man_norm_1 <= man_1(50) & man_1(48 downto 25) when "10",
                  man_1(49) & man_1(47 downto 24) when "01",
                  man_1(48) & man_1(46 downto 23) when others;
```

Pembetulan Eksponen dan Semakan Limpahan

Pembetulan eksponen (Program 3.15) merupakan penambahan antara hasiltambah eksponen yang didapati daripada tahap pertama iaitu `hsl_tamb_eksp` dengan nilai `anjakan_1` yang menghasilkan `jeks2`. Seterusnya nilai eksponen yang diperbetulkan

disemak semula untuk mengesan limpahan yang wujud. Untuk limpahan atas tahap kedua, ia meyemak samada terdapat isyarat limpahan atas dari tahap pertama atau mengesan kesamaan bit ke lapan bagi isyarat `jeks2` dan `hsl_tamb_eksp` supaya ia berisyarat 0. Manakala limpahan bawah menggunakan teknik songsangan limpahan bawah iaitu ia mengesan satuasi yang bukan menyebabkan limpahan bawah seperti dalam Program 3.16. Teknik ini juga digunakan dalam pembetulan eksponen dan semakan limpahan pada tahap ketiga.

```
--Program 3.15: Pembetulan Eksponen
jeks2 := (hsl_tamb_eksp) + ("000000" & anjakan_1);

--Program 3.16: Semakan Limpahan Tahap Kedua
--lbwheks2 & latseks2 merupakam isyarat limpahan_atas & limpahan
_bawah --daripada tahap pertama
--Limpahan Atas
if (lbwheks2 = '0') and ((latseks2 = '1') or
    ((jeks2(7) /= hsl_tamb_eksp(7)) and (hsl_tamb_eksp(7) = '0')))
then
     lats2 := '1';
--Limpahan Bawah
elsif (((jeks2(7)) /= hsl_tamb_eksp(7)) and
       (hsl_tamb_eksp(7) = '0') and (lbwheks2 = '1')) or
         ((hsl_tamb_eksp = "10000000") and (anjakan_1 /= "00")) then
     lbwh2 := '0';
end if;
```

3.2.3 Tahap Ketiga

Pembundaran

Isyarat pembundaran ditentukan oleh bit kurang bererti pada isyarat `man_norm_1` (yakni bit pertama) dan ia dinamakan `bit_bundar`. Pada tahap ini perlanjutan bit-tanda dikenakan pada `man_norm_1` (yakni `man_norm_1` dari bit kedua hingga ke25)

sebanyak tiga bit menjadikannya 27bit dan dinamakan `man_2`. Proses pembundaran menggunakan menggunakan skim pembundaran terdekat seperti dalam Program 3.17.

```
--Program 3.17: Proses Pembundaran
IF(bit_bundar = '0') THEN
   man_2(24 DOWNTO 1) <= man_norm_1(25 DOWNTO 2);
   man_2(27 downto 25)<= man_norm_1(25)&man_norm_1n(25)&man_norm_1(25);
ELSIF(bit_bundar = '1') THEN
   man_2(27 DOWNTO 1) <= man_norm_1(25)&man_norm_1(25)&man_norm_1(25)&
                man_norm_1(25 downto 2) + "00000000000000000000000001";
END IF;
```

Penormalan Semula

Setelah melakukan pembundaran atau tidak isyarat `man_2` dinormalkan semula. Untuk penormalan semula, bit25 hingga ke bit27 dalam isyarat `man_2` digunakan untuk mendapat isyarat `anjakan_2` bagi menentukan anjakan yang diperlukan dan pembetulan eksponen (rujuk Program 3.18 dan 3.19). Isyarat `man_norm_2` merupakan isyarat keluaran setelah proses penormalan dilaksanakan dan bilangan bitnya ialah 24bit.

```
--Program 3.18: Isyarat anjakan_2
anjakan_2 <= "10" when (man_2(27) xor man_2(26)) = '1' else
             "01" when (man_2(26) xor man_2(25)) = '1' else
             "00";
--Program 3.19: Proses anjakan
with anjakan_2 select
    man_norm_2 <= man_2(27) & man_2(25 downto 3) when "10",
                  man_2(26) & man_2(24 downto 2) when "01",
                  man_2(25) & man_2(23 downto 1) when others;
```

Penentu Isyarat Jawapan

Program 3.20 merupakan aturcara unutk perihalan penentu isyarat jawapan. Ia bermula dengan isyarat sifar, jika nombor_a atau nombor_b sifar maka jawapan yang diberikan ialah sifar iaitu "00000000 00000000 00000000 00000001". Sekiranya wujud limpahan atas, terdapat dua jawapan iaitu untuk nombor negatif dan positif, sekiranya positif, jawapan yang diberikan ialah nilai positif yang maksima iaitu "01111111 11111111 11111111 11111110" dan sekiranya negatif, jawapan ialah nilai negatif yang maksima iaitu "10000000 00000000 00000000 11111110".

```
--Program 3.20: Penentu isyarat jawapan
--isyarat sifar
if (a_sifar3 = '1') or (b_sifar3 = '1') then
                isy_sifar <= '1';
                jwp <= "00000000"&"00000000"&"00000000"&"00000001";
--isyarat limpahan atas
elsif (latseks5 = '1') then
                isy_latas <= '1';
                 --nombor positif
                 if (man_norm_2(23) = '0') then
                        jwp <= "01111111"&"11111111"&"11111111"&"11111110";
                 --nombor negatif
                 else
                        jwp <= "10000000"&"00000000"&"00000000"&"11111110";
                 end if;
--isyarat limpahan bawah
elsif (lbwheks5 = '1')  or ((eks_norm_2 = "00000001") and
       (man_norm_2 /= "00000000"&"00000000"&"00000000")) then
                 isy_sifar <= '1'; isy_lbawah <= '1';
                 jwp <= "00000000"&"00000000"&"00000000"&"00000001";
--selainnya
else
                 jwp <= man_norm_2&eks_norm_2;
end if;
```

3.3 ALIRAN PROSES MEREKABENTUK

Rajah 3.7 menunjukkan aliran proses merekabentuk litar. Proses ini bermula pada peringkat perihalan litar dengan VHDL dan diakhiri dengan penyelakuan peringkat penempatan (*placing*) dan penghalaan (*routing*). Alirannya adalah seperti berikut:

i. Menulis perihalan litar dalam VHDL

ii. Membuat analisa samada peringkat (i) boleh disintesiskan atau tidak. Sekiranya gagal maka perlu kembali ke peringkat (i) untuk pengubahsuaian. Jika berjaya lakukan penyelakuan terhadap perihalan tersebut.

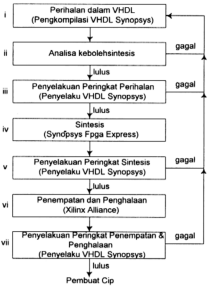
iii. Peringkat ini hanya menentukan kefungsian perihalan tersebut supaya ia memenuhi kefungsian yang dikehendaki. Jika gagal, kembali ke peringkat (i). Jika berjaya teruskan ke peringkat (iv) untuk proses sintesis dan pengoptimuman.

iv. Ia melakukan penganalisaan sintesis dan pengoptimuman. Jika terdapat ralat, ketika penganalisaan sintesis maka proses pengotimuman tidak dijalankan dan kembali semula ke peringkat (i). Sekira tiada ralat, proses pengotimuman akan dijalan secara automatik dan seterusnya ke peringkat (v).

v. Ia merupakan peringkat penyelakuan sintesis atau get, di mana jawapan hendaklah sama dengan kefungsian yang dikehendaki dalam peringkat (ii). Jika tidak, ulang semula dari peringkat (i).

vi. Peringkat ini melaksanakan proses penempatan dan penghalaan.

vii. Ini merupakan peringkat terakhir sebelum ia dibawa ke pembuat cip. Di sini pengesahbetulan fungsi dan pemasaan dilaksanakan di mana kefungsian hendaklah sama dengan peringkat (ii) dan (v).



Rajah 3.7: Aliran proses merekabentuk