Prosesor FFT/IFFT 64-point untuk Aplikasi Wireless LAN 802.11a

Soal

Buatlah rancangan suatu prosesor FFT 64-point sampai pada level diskripsi dalam VHDL yang synthesizable. Prosesor FFT ini ditujukan untuk aplikasi pada WLAN.

Jawaban

Pada proyek akhir ini, akan diimplementasikan prosesor FFT/IFFT 64-point menggunakan kode VHDL yang synthesizable untuk digunakan pada komunikasi Wireless LAN 802.11a. Arsitektur prosesor FFT/IFFT 64-point yang akan diimplementasikan mengikuti arsitektur yang dijelaskan pada paper berikut.

> K. Maharatna, E. Grass, and U. Jagdhold "A 64-point Fourier Transform Chip for High-Speed Wireless LAN Application Using OFDM", IEEE Journal of Solid-State Circuits, Vol. 39, No. 3, March 2004, pp. 484-493.

Dalam paper tersebut akan diimplementasikan FFT/IFFT 64-titik menggunakan dua buah FFT 8-titik. FFT 8-titik telah diimplementasikan sebagai proyek ujian akhir semester sehingga pada proyek ini cukup dilakukan penggunaan kembali FFT 8-titik yang telah diimplementasikan tersebut. Berikut ini adalah spesifikasi FFT/IFFT 64-titik yang akan diimplementasikan.

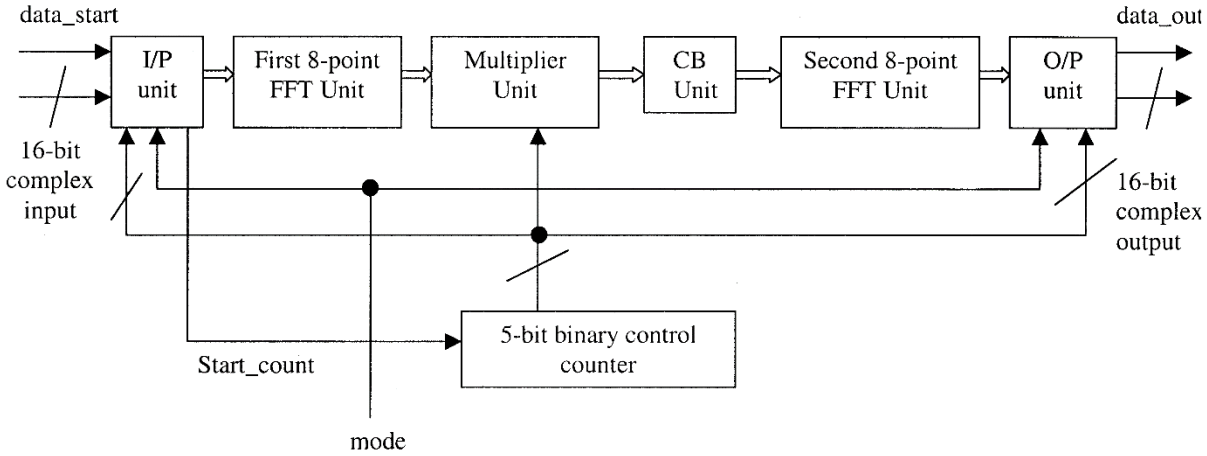
- Fast Fourier Transform (FFT) 64-titik yang terdiri atas dua buah Fast Fourier Transform 8-titik dengan arsitektur dua dimensi. Prosesor FFT 8-titik yang telah diimplementasikan sebelumnya adalah menggunakan arsitektur Decimation-in-Time.
- Prosesor FFT 64-titik yang akan direalisasikan dapat melakukan perhitungan penuh terhadap semua data set dalam 23 siklus clock dan perhitungan penuh tersebut harus dapat dilakukan kurang dari 4µS sesuai dengan spesifikasi minimum agar dapat digunakan dalam komunikasi WLAN 802.11a.
- Dengan memberikan tambahan rangkaian *swapping* pada bagian awal dan bagian akhir rangkaian, prosesor ini dapat digunakan untuk melakukan FFT dan Invers FFT (IFFT).
- Data set berupa data kompleks yang terdiri atas 16-bit data real dan 16-bit data imaginary dalam bentuk two's complement.

Complex Data Set (32-bit)	
31...16 Real Data (16-bit)	15...0 Imaginary Data (16-bit)

- Data real dan data imaginary mengikuti format *fixed point* Q4.12.
  Data terbesar : 7,9997558593750 (`0b0111111111111111`)
  Data terkecil : -8,0000000000000 (`0b1000000000000000`)
  Ketelitian : 0,0002441406250 (`0b0000000000000001`)
- Dengan menggunakan format *fixed point* Q4.12 tersebut, ada keterbatasan nilai yang dapat dikalkulasi. Pada dasarnya, magnitudo dari sebuah data individu dalam sebuah FFT dapat berkembang sebesar n kali dengan n adalah panjang dari FFT. Dengan demikian, dengan melakukan skala terhadap data sebelum memasuki tahap pertama FFT dengan faktor $1/n$, *overflow* dapat dicegah untuk setiap input. Namun, hal ini akan menyebabkan kita memperoleh rasio derau terhadap sinyal (*noise-to-signal ratio*) sebesar n^2 [Oppenheim & Schafer 1989, equation 9.101], [Welch 1969].
- Bila kita melakukan skala terhadap input data sebelum memasuki setiap tahap FFT dengan skala $1/2$, kita tetap dapat memperoleh total skala sebesar $1/n$ namun dengan rasio derau terhadap sinyal

(*noise-to-signal ratio*) sebesar n [Oppenheim & Schafer 1989, equation 9.105], [Welch 1969]. Cara yang sangat efisien untuk melakukan skala dengan $^1/_2$ adalah dengan menggunakan *right shift*. Dalam proyek ini, faktor skala belum diaplikasikan sehingga input harus dijamin agar tidak meyebabkan kalkulasi *overflow*. Untuk melakukan aplikasi faktor skala, kita dapat dengan mudah melakukan *right shift* dengan menambahkan *rightshifter* di depan setiap tahap FFT.

- Gambar arsitektur FFT 64-point yang diambil dari paper tersebut diberikan sebagai berikut.

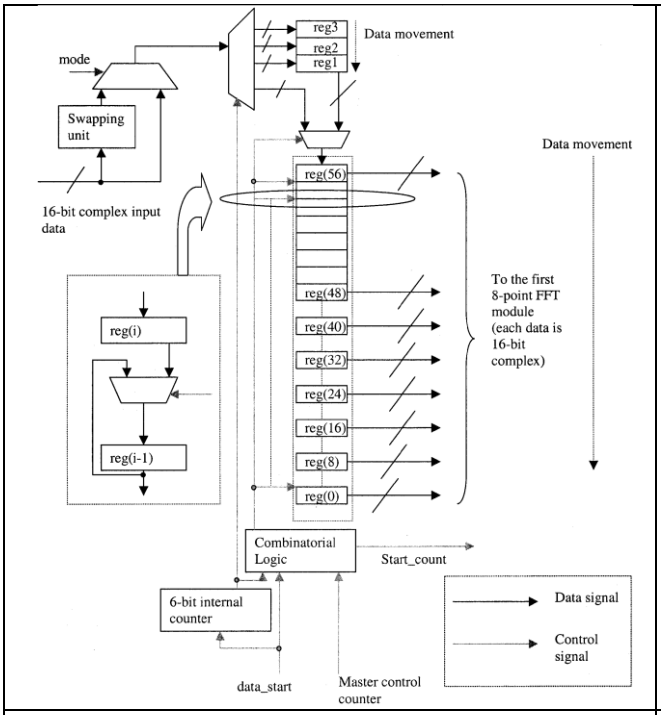


Terdapat enam blok utama dan satu blok kontrol untuk merealisasikan FFT 64-titik tersebut. Keenam blok utama dan satu blok kontrol tersebut adalah sebagai berikut.

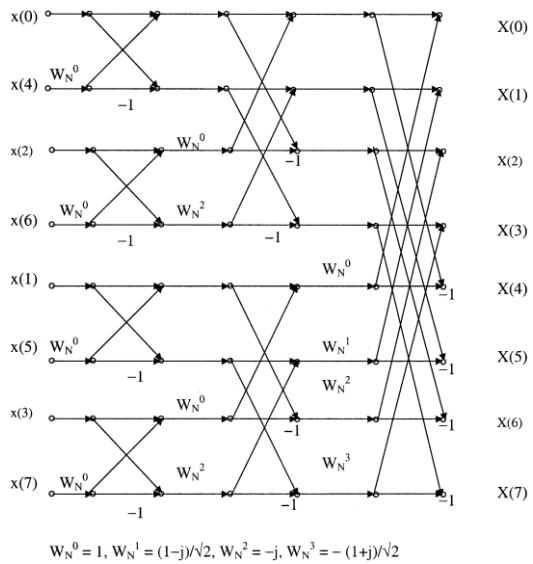
1. Blok Input Buffer (I/P unit) yang berfungsi untuk mengubah data serial menjadi data paralel, mengurutkan data masuk, dan buffer data masuk.
2. Blok FFT 8-point tahap pertama.
3. Blok Interdimensional Multiplier yang berfungsi untuk mengalikan data output dari FFT 8-point tahap pertama dengan konstanta twiddle factor interdimensional.
4. Blok Internal Register (CB unit) yang berfungsi untuk membuffer data output dari Blok Interdimensional Multiplier dan mengurutkan data.
5. Blok FFT 8-point tahap kedua.
6. Blok Output Buffer (O/P unit) yang berfungsi untuk mengubah data paralel menjadi data serial, mengurutkan data keluar, dan buffer data keluar.
7. Blok Master Control yang berfungsi untuk mengatur jalannya rangkaian. Blok Master Control ini dibantu oleh counter 6-bit pada tahap input dan counter 6-bit pada tahap output.

Ketujuh blok tersebut akan dijelaskan secara rinci pada bagian berikutnya beserta diagram rangkaian struktural yang diimplementasikan. Semua blok tersebut akan didefinisikan secara struktural dalam kode VHDL kecuali blok master control yang akan didefinisikan secara behavioral.

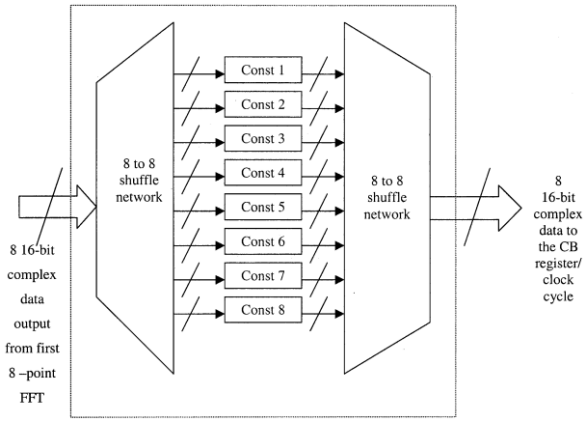
- Dengan mengintegrasikan ketujuh blok tersebut, prosesor FFT/IFFT 64-titik dapat direalisasikan dan dapat disintesis. Selain itu, hasil sintesis tersebut akan disimulasikan untuk memeriksa fungsionalitas dan timing dari proses FFT/IFFT 64-titik yang telah dibuat.
- Program yang digunakan dalam proyek ini adalah sebagai berikut.
    - Altera Quartus II 9.1sp2 untuk melakukan sintesis dan simulasi.
    - ModelSim Altera Edition 10.1e untuk melakukan simulasi.
      Simulasi pada ModelSim dilakukan untuk melihat lebih jauh perubahan-perubahan sinyal-sinyal internal di dalam sebuah entitas desain dengan lebih mudah. Pada Altera Quartus II 9.1sp2, hal ini tidak dapat dilakukan karena pada waveform simulasi, sinyal yang tergambar pada waveform berupa port input atau output entitas level tertinggi saja.



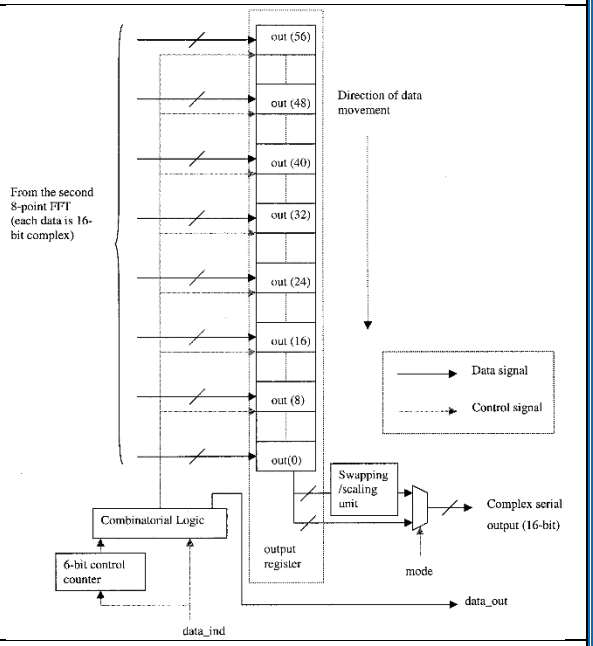
Input Buffer (I/P unit)



FFT 8-Point



Interdimensional Multiplier



Output Buffer (O/P Unit)

A. Tinjauan Matematika

Discrete Fourier Transform (DFT) $A(r)$ dari urutan data kompleks $B(k)$ sejumlah N dengan $r, k \in \{0, 1, \dots, N-1\}$ dapat ditulis dalam persamaan sebagai berikut.

$$A(r) = \sum_{k=0}^{N-1} B(k) W_N^{rk}$$

$$W_N = e^{-2\pi j/N}$$

Untuk $N = MT$, $r = s + Tt$, dan $k = l + Mm$ dengan $s, l \in \{0,1 \dots 7\}$ dan $m, t \in \{0,1, \dots, T-1\}$, persamaan tersebut dapat dituliskan kembali sebagai berikut.

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

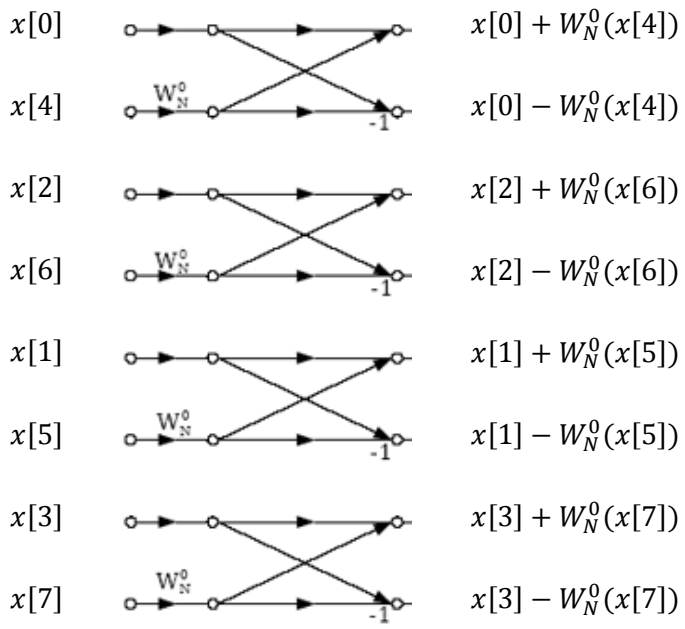
$$A(s+Tt) = \sum_{l=0}^{M-1} W_M^{lt} \left[W_{MT}^{sl} \sum_{m=0}^{T-1} B(l+Mm) W_T^{sm} \right]$$

Persamaan tersebut mengandung arti bahwa FFT dengan panjang N dapat direalisasikan dengan dua buah FFT dengan panjang M (FFT M-titik) dan T (FFT T-titik) sehingga $N = MT$. Dengan demikian, FFT 64-titik yang akan kita buat dapat didekomposisi menjadi dua buah FFT 8-titik sebagai berikut.

$$A(s+8t) = \sum_{l=0}^{7} \left[W_{64}^{sl} \sum_{m=0}^{7} B(l+8m) W_8^{sm} \right] W_8^{lt}$$

Dengan demikian, diperlukan realisasi terlebih dahulu FFT 8-titik sebelum menggabungkannya menjadi FFT 64-titik. Berikut ini adalah tinjauan matematika untuk FFT 8-titik yang direalisasikan pada tugas sebelumnya berdasarkan arsitektur FFT 8-titik yang dipilih yaitu Decimation in Time. Berdasarkan gambar struktur FFT 8-titik yang telah diberikan sebelumnya, kita dapat membagi FFT 8-titik ke dalam tiga tahap untuk $N = 8$ sebagai berikut.

Tahap pertama dari FFT 8-titik dijabarkan sebagai berikut.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Tahap kedua FFT 8-titik dijabarkan sebagai berikut.

$x[0] + W_N^0(x[4])$

$x[0] - W_N^0(x[4])$

$x[2] + W_N^0(x[6])$

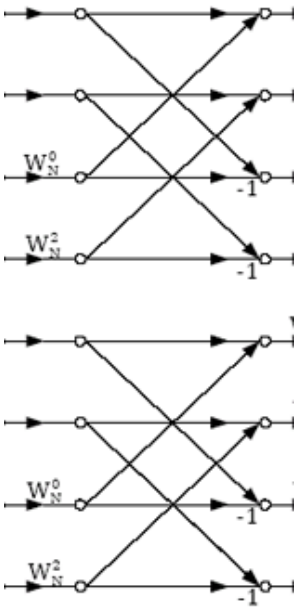
$x[2] - W_N^0(x[6])$

$x[1] + W_N^0(x[5])$

$x[1] - W_N^0(x[5])$

$x[3] + W_N^0(x[7])$

$x[3] - W_N^0(x[7])$



$x[0] + W_N^0(x[2]) + W_N^0(x[4]) + W_N^0 W_N^0(x[6])$

$x[0] + W_N^2(x[2]) - W_N^0(x[4]) - W_N^0 W_N^2(x[6])$

$x[0] - W_N^0(x[2]) + W_N^0(x[4]) - W_N^0 W_N^0(x[6])$

$x[0] - W_N^2(x[2]) - W_N^0(x[4]) + W_N^0 W_N^2(x[6])$

$x[1] + W_N^0(x[3]) + W_N^0(x[5]) + W_N^0 W_N^0(x[7])$

$x[1] + W_N^2(x[3]) - W_N^0(x[5]) - W_N^0 W_N^2(x[7])$

$x[1] - W_N^0(x[3]) + W_N^0(x[5]) - W_N^0 W_N^0(x[7])$

$x[1] - W_N^2(x[3]) - W_N^0(x[5]) + W_N^0 W_N^2(x[7])$

Tahap ketiga FFT 8-titik dijabarkan sebagai berikut.

$x[0] + W_N^0(x[2]) + W_N^0(x[4]) + W_N^0 W_N^0(x[6])$

$x[0] + W_N^2(x[2]) - W_N^0(x[4]) - W_N^0 W_N^2(x[6])$

$x[0] - W_N^0(x[2]) + W_N^0(x[4]) - W_N^0 W_N^0(x[6])$

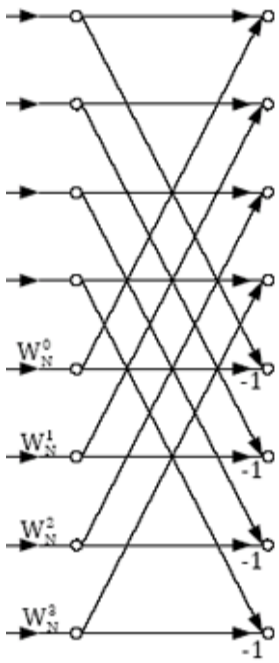
$x[0] - W_N^2(x[2]) - W_N^0(x[4]) + W_N^0 W_N^2(x[6])$

$x[1] + W_N^0(x[3]) + W_N^0(x[5]) + W_N^0 W_N^0(x[7])$

$x[1] + W_N^2(x[3]) - W_N^0(x[5]) - W_N^0 W_N^2(x[7])$

$x[1] - W_N^0(x[3]) + W_N^0(x[5]) - W_N^0 W_N^0(x[7])$

$x[1] - W_N^2(x[3]) - W_N^0(x[5]) + W_N^0 W_N^2(x[7])$



$$x[0] + W_N^0(x[1]) + W_N^0(x[2]) + W_N^0 W_N^0(x[3]) + W_N^0(x[4]) + W_N^0 W_N^0(x[5]) + W_N^0 W_N^0(x[6]) + W_N^0 W_N^0 W_N^0(x[7]) = X[0]$$

$$x[0] + W_N^1(x[1]) + W_N^2(x[2]) + W_N^1 W_N^2(x[3]) - W_N^0(x[4]) - W_N^0 W_N^1(x[5]) - W_N^0 W_N^2(x[6]) - W_N^0 W_N^1 W_N^2(x[7]) = X[1]$$

$$x[0] + W_N^2(x[1]) - W_N^0(x[2]) - W_N^0 W_N^2(x[3]) + W_N^0(x[4]) + W_N^0 W_N^2(x[5]) - W_N^0 W_N^0(x[6]) - W_N^0 W_N^0 W_N^2(x[7]) = X[2]$$

$$x[0] + W_N^3 x(x[1]) - W_N^2(x[2]) - W_N^2 W_N^3(x[3]) - W_N^0(x[4]) - W_N^0 W_N^3(x[5]) + W_N^0 W_N^2(x[6]) + W_N^0 W_N^2 W_N^3(x[7]) = X[3]$$

$$x[0] - W_N^0(x[1]) + W_N^0(x[2]) - W_N^0 W_N^0(x[3]) + W_N^0(x[4]) - W_N^0 W_N^0(x[5]) + W_N^0 W_N^0(x[6]) - W_N^0 W_N^0 W_N^0(x[7]) = X[4]$$

$$x[0] - W_N^1(x[1]) + W_N^2(x[2]) - W_N^1 W_N^2(x[3]) - W_N^0(x[4]) + W_N^0 W_N^1(x[5]) - W_N^0 W_N^2(x[6]) + W_N^0 W_N^1 W_N^2(x[7]) = X[5]$$

$$x[0] - W_N^2(x[1]) - W_N^0(x[2]) + W_N^0 W_N^2(x[3]) + W_N^0(x[4]) - W_N^0 W_N^2(x[5]) - W_N^0 W_N^0(x[6]) + W_N^0 W_N^0 W_N^2(x[7]) = X[6]$$

$$x[0] - W_N^3(x[1]) - W_N^2(x[2]) + W_N^2 W_N^3(x[3]) - W_N^0(x[4]) + W_N^0 W_N^3(x[5]) + W_N^0 W_N^2(x[6]) - W_N^0 W_N^2 W_N^3(x[7]) = X[7]$$

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Dalam tabel tersebut, W_N^0, W_N^1, W_N^2, dan W_N^3 adalah *twiddle factor* untuk FFT 8-point ($N = 8$). Dalam FFT 8-point, terdapat delapan buah *twiddle factor* dengan 1 *twiddle factor* trivial yang bernilai 1. Kedelapan nilai *twiddle factor* diberikan pada tabel berikut. Nilai twiddle factor akan berulang di luar nilai yang tertera pada tabel berikut. Sebagai contoh, untuk $\mathrm{N} = 8$, Nilai $W_N^0 = W_N^8 = W_N^{16}$ dan seterusnya. Dengan demikian nilai $W_N^n = W_N^{n+mN}$ dengan $n \in \{0\,..\,N-1\}$ dan $m \in \mathbb{Z}$.

W_N^0	1	W_N^0
W_N^1	$\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$	W_N^1
W_N^2	$-j1$	W_N^2
W_N^3	$-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$	W_N^3
W_N^4	-1	$-W_N^0$
W_N^5	$-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}$	$-W_N^1$
W_N^6	$j1$	$-W_N^2$
W_N^7	$\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}$	$-W_N^3$

Dari tabel tersebut, kita dapat mengetahui bahwa hanya terdapat empat *twiddle factor* saja yang direalisasikan dalam rangkaian (termasuk satu trivial). *Twiddle factor* yang lain dapat diimplementasikan dengan perkalian menggunakan bilangan -1 (invers tanda). Perhatikan pula bahwa $W_N^1 W_N^2 = W_N^3$ dan $W_N^2 W_N^3 = -W_N^1$. Dengan mengganti *twiddle factor* trivial dan perkalian dua buah *twiddle factor* dengan yang dijelaskan sebelumnya, kita memperoleh hubungan input dan output sebagai berikut.

$$X[0] = x[0] + x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7]$$
$$X[1] = x[0] + W_N^1 x[1] + W_N^2 x[2] + W_N^3 x[3] - x[4] - W_N^1 x[5] - W_N^2 x[6] - W_N^3 x[7]$$
$$X[2] = x[0] + W_N^2 x[1] - x[2] - W_N^2 x[3] + x[4] + W_N^2 x[5] - x[6] - W_N^2 x[7]$$
$$X[3] = x[0] + W_N^3 x[1] - W_N^2 x[2] + W_N^1 x[3] - x[4] - W_N^3 x[5] + W_N^2 x[6] - W_N^1 x[7]$$
$$X[4] = x[0] - x[1] + x[2] - x[3] + x[4] - x[5] + x[6] - x[7]$$
$$X[5] = x[0] - W_N^1 x[1] + W_N^2 x[2] - W_N^3 x[3] - x[4] + W_N^1 x[5] - W_N^2 x[6] + W_N^3 x[7]$$
$$X[6] = x[0] - W_N^2 x[1] - x[2] + W_N^2 x[3] + x[4] - W_N^2 x[5] - x[6] + W_N^2 x[7]$$
$$X[7] = x[0] - W_N^3 x[1] - W_N^2 x[2] - W_N^1 x[3] - x[4] + W_N^3 x[5] + W_N^2 x[6] + W_N^1 x[7]$$

Selanjutnya, tinjauan matematika untuk FFT 64-titik dilakukan sebagai berikut. Tinjauan matematika untuk FFT 64-titik dilakukan dengan mengaitkan hubungan antara persamaan FFT 64-titik yang telah diuraikan menjadi dua FFT 8-titik dengan struktur rangkaian FFT 64-titik yang akan dibuat. Dengan demikian, terdapat tiga tahap komputasi yang dilakukan yaitu FFT 8-titik tahap satu, perkalian kompleks dengan konstanta interdimensional, dan FFT 8-titik tahap dua. Hal yang perlu diingat adalah $W_N^n = W_N^{n+mN}$ dengan $n \in \{0\,..\,N-1\}$ dan $m \in \mathbb{Z}$.

Tahap pertama FFT 64-titik dijelaskan sebagai berikut. Komputasi dilakukan untuk data input $B(0)$ hingga $B(63)$. Aturan komputasi menurut struktur blok input buffer dilakukan sebagai berikut.

Set	Input Data
0	$B(0), B(8), B(16), B(24), B(32), B(40), B(48), B(56)$
1	$B(1), B(9), B(17), B(25), B(33), B(41), B(49), B(57)$
2	$B(2), B(10), B(18), B(26), B(34), B(42), B(50), B(58)$
3	$B(3), B(11), B(19), B(27), B(35), B(43), B(51), B(59)$
4	$B(4), B(12), B(20), B(28), B(36), B(44), B(52), B(60)$
5	$B(5), B(13), B(21), B(29), B(37), B(45), B(53), B(61)$
6	$B(6), B(14), B(22), B(30), B(38), B(46), B(54), B(62)$
7	$B(7), B(15), B(23), B(31), B(39), B(47), B(55), B(63)$

Dengan mengingat bahwa $W_N^n = W_N^{n+mN}$ dengan $n \in \{0 \,..\, N-1\}$ dan $m \in \mathbb{Z}$, berikut ini adalah hasil operasi yang dilakukan oleh FFT 8-titik tahap pertama.

Set	Data	Hasil Komputasi
0	0	$B(0)W_8^0 + B(8)W_8^0 + B(16)W_8^0 + B(24)W_8^0 + B(32)W_8^0 + B(40)W_8^0 + B(48)W_8^0 + B(56)W_8^0$
0	1	$B(0)W_8^0 + B(8)W_8^1 + B(16)W_8^2 + B(24)W_8^3 + B(32)W_8^4 + B(40)W_8^5 + B(48)W_8^6 + B(56)W_8^7$
0	2	$B(0)W_8^0 + B(8)W_8^2 + B(16)W_8^4 + B(24)W_8^6 + B(32)W_8^8 + B(40)W_8^{10} + B(48)W_8^{12} + B(56)W_8^{14}$
0	3	$B(0)W_8^0 + B(8)W_8^3 + B(16)W_8^6 + B(24)W_8^9 + B(32)W_8^{12} + B(40)W_8^{15} + B(48)W_8^{18} + B(56)W_8^{21}$
0	4	$B(0)W_8^0 + B(8)W_8^4 + B(16)W_8^8 + B(24)W_8^{12} + B(32)W_8^{16} + B(40)W_8^{20} + B(48)W_8^{24} + B(56)W_8^{28}$
0	5	$B(0)W_8^0 + B(8)W_8^5 + B(16)W_8^{10} + B(24)W_8^{15} + B(32)W_8^{20} + B(40)W_8^{25} + B(48)W_8^{30} + B(56)W_8^{35}$
0	6	$B(0)W_8^0 + B(8)W_8^6 + B(16)W_8^{12} + B(24)W_8^{18} + B(32)W_8^{24} + B(40)W_8^{30} + B(48)W_8^{36} + B(56)W_8^{42}$
0	7	$B(0)W_8^0 + B(8)W_8^7 + B(16)W_8^{14} + B(24)W_8^{21} + B(32)W_8^{28} + B(40)W_8^{35} + B(48)W_8^{42} + B(56)W_8^{49}$
1	0	$B(1)W_8^0 + B(9)W_8^0 + B(17)W_8^0 + B(25)W_8^0 + B(33)W_8^0 + B(41)W_8^0 + B(49)W_8^0 + B(57)W_8^0$
1	1	$B(1)W_8^0 + B(9)W_8^1 + B(17)W_8^2 + B(25)W_8^3 + B(33)W_8^4 + B(41)W_8^5 + B(49)W_8^6 + B(57)W_8^7$
1	2	$B(1)W_8^0 + B(9)W_8^2 + B(17)W_8^4 + B(25)W_8^6 + B(33)W_8^8 + B(41)W_8^{10} + B(49)W_8^{12} + B(57)W_8^{14}$
1	3	$B(1)W_8^0 + B(9)W_8^3 + B(17)W_8^6 + B(25)W_8^9 + B(33)W_8^{12} + B(41)W_8^{15} + B(49)W_8^{18} + B(57)W_8^{21}$
1	4	$B(1)W_8^0 + B(9)W_8^4 + B(17)W_8^8 + B(25)W_8^{12} + B(33)W_8^{16} + B(41)W_8^{20} + B(49)W_8^{24} + B(57)W_8^{28}$
1	5	$B(1)W_8^0 + B(9)W_8^5 + B(17)W_8^{10} + B(25)W_8^{15} + B(33)W_8^{20} + B(41)W_8^{25} + B(49)W_8^{30} + B(57)W_8^{35}$
1	6	$B(1)W_8^0 + B(9)W_8^6 + B(17)W_8^{12} + B(25)W_8^{18} + B(33)W_8^{24} + B(41)W_8^{30} + B(49)W_8^{36} + B(57)W_8^{42}$
1	7	$B(1)W_8^0 + B(9)W_8^7 + B(17)W_8^{14} + B(25)W_8^{21} + B(33)W_8^{28} + B(41)W_8^{35} + B(49)W_8^{42} + B(57)W_8^{49}$
2	0	$B(2)W_8^0 + B(10)W_8^0 + B(18)W_8^0 + B(26)W_8^0 + B(34)W_8^0 + B(42)W_8^0 + B(50)W_8^0 + B(58)W_8^0$
2	1	$B(2)W_8^0 + B(10)W_8^1 + B(18)W_8^2 + B(26)W_8^3 + B(34)W_8^4 + B(42)W_8^5 + B(50)W_8^6 + B(58)W_8^7$
2	2	$B(2)W_8^0 + B(10)W_8^2 + B(18)W_8^4 + B(26)W_8^6 + B(34)W_8^8 + B(42)W_8^{10} + B(50)W_8^{12} + B(58)W_8^{14}$
2	3	$B(2)W_8^0 + B(10)W_8^3 + B(18)W_8^6 + B(26)W_8^9 + B(34)W_8^{12} + B(42)W_8^{15} + B(50)W_8^{18} + B(58)W_8^{21}$
2	4	$B(2)W_8^0 + B(10)W_8^4 + B(18)W_8^8 + B(26)W_8^{12} + B(34)W_8^{16} + B(42)W_8^{20} + B(50)W_8^{24} + B(58)W_8^{28}$
2	5	$B(2)W_8^0 + B(10)W_8^5 + B(18)W_8^{10} + B(26)W_8^{15} + B(34)W_8^{20} + B(42)W_8^{25} + B(50)W_8^{30} + B(58)W_8^{35}$
2	6	$B(2)W_8^0 + B(10)W_8^6 + B(18)W_8^{12} + B(26)W_8^{18} + B(34)W_8^{24} + B(42)W_8^{30} + B(50)W_8^{36} + B(58)W_8^{42}$
2	7	$B(2)W_8^0 + B(10)W_8^7 + B(18)W_8^{14} + B(26)W_8^{21} + B(34)W_8^{28} + B(42)W_8^{35} + B(50)W_8^{42} + B(58)W_8^{49}$
3	0	$B(3)W_8^0 + B(11)W_8^0 + B(19)W_8^0 + B(27)W_8^0 + B(35)W_8^0 + B(43)W_8^0 + B(51)W_8^0 + B(59)W_8^0$
3	1	$B(3)W_8^0 + B(11)W_8^1 + B(19)W_8^2 + B(27)W_8^3 + B(35)W_8^4 + B(43)W_8^5 + B(51)W_8^6 + B(59)W_8^7$
3	2	$B(3)W_8^0 + B(11)W_8^2 + B(19)W_8^4 + B(27)W_8^6 + B(35)W_8^8 + B(43)W_8^{10} + B(51)W_8^{12} + B(59)W_8^{14}$
3	3	$B(3)W_8^0 + B(11)W_8^3 + B(19)W_8^6 + B(27)W_8^9 + B(35)W_8^{12} + B(43)W_8^{15} + B(51)W_8^{18} + B(59)W_8^{21}$
3	4	$B(3)W_8^0 + B(11)W_8^4 + B(19)W_8^8 + B(27)W_8^{12} + B(35)W_8^{16} + B(43)W_8^{20} + B(51)W_8^{24} + B(59)W_8^{28}$
3	5	$B(3)W_8^0 + B(11)W_8^5 + B(19)W_8^{10} + B(27)W_8^{15} + B(35)W_8^{20} + B(43)W_8^{25} + B(51)W_8^{30} + B(59)W_8^{35}$
3	6	$B(3)W_8^0 + B(11)W_8^6 + B(19)W_8^{12} + B(27)W_8^{18} + B(35)W_8^{24} + B(43)W_8^{30} + B(51)W_8^{36} + B(59)W_8^{42}$
3	7	$B(3)W_8^0 + B(11)W_8^7 + B(19)W_8^{14} + B(27)W_8^{21} + B(35)W_8^{28} + B(43)W_8^{35} + B(51)W_8^{42} + B(59)W_8^{49}$
4	0	$B(4)W_8^0 + B(12)W_8^0 + B(20)W_8^0 + B(28)W_8^0 + B(36)W_8^0 + B(44)W_8^0 + B(52)W_8^0 + B(60)W_8^0$
4	1	$B(4)W_8^0 + B(12)W_8^1 + B(20)W_8^2 + B(28)W_8^3 + B(36)W_8^4 + B(44)W_8^5 + B(52)W_8^6 + B(60)W_8^7$

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

4	2	$B(4)W_8^0 + B(12)W_8^2 + B(20)W_8^4 + B(28)W_8^6 + B(36)W_8^8 + B(44)W_8^{10} + B(52)W_8^{12} + B(60)W_8^{14}$
4	3	$B(4)W_8^0 + B(12)W_8^3 + B(20)W_8^6 + B(28)W_8^9 + B(36)W_8^{12} + B(44)W_8^{15} + B(52)W_8^{18} + B(60)W_8^{21}$
4	4	$B(4)W_8^0 + B(12)W_8^4 + B(20)W_8^8 + B(28)W_8^{12} + B(36)W_8^{16} + B(44)W_8^{20} + B(52)W_8^{24} + B(60)W_8^{28}$
4	5	$B(4)W_8^0 + B(12)W_8^5 + B(20)W_8^{10} + B(28)W_8^{15} + B(36)W_8^{20} + B(44)W_8^{25} + B(52)W_8^{30} + B(60)W_8^{35}$
4	6	$B(4)W_8^0 + B(12)W_8^6 + B(20)W_8^{12} + B(28)W_8^{18} + B(36)W_8^{24} + B(44)W_8^{30} + B(52)W_8^{36} + B(60)W_8^{42}$
4	7	$B(4)W_8^0 + B(12)W_8^7 + B(20)W_8^{14} + B(28)W_8^{21} + B(36)W_8^{28} + B(44)W_8^{35} + B(52)W_8^{42} + B(60)W_8^{49}$
5	0	$B(5)W_8^0 + B(13)W_8^0 + B(21)W_8^0 + B(29)W_8^0 + B(37)W_8^0 + B(45)W_8^0 + B(53)W_8^0 + B(61)W_8^0$
5	1	$B(5)W_8^0 + B(13)W_8^1 + B(21)W_8^2 + B(29)W_8^3 + B(37)W_8^4 + B(45)W_8^5 + B(53)W_8^6 + B(61)W_8^7$
5	2	$B(5)W_8^0 + B(13)W_8^2 + B(21)W_8^4 + B(29)W_8^6 + B(37)W_8^8 + B(45)W_8^{10} + B(53)W_8^{12} + B(61)W_8^{14}$
5	3	$B(5)W_8^0 + B(13)W_8^3 + B(21)W_8^6 + B(29)W_8^9 + B(37)W_8^{12} + B(45)W_8^{15} + B(53)W_8^{18} + B(61)W_8^{21}$
5	4	$B(5)W_8^0 + B(13)W_8^4 + B(21)W_8^8 + B(29)W_8^{12} + B(37)W_8^{16} + B(45)W_8^{20} + B(53)W_8^{24} + B(61)W_8^{28}$
5	5	$B(5)W_8^0 + B(13)W_8^5 + B(21)W_8^{10} + B(29)W_8^{15} + B(37)W_8^{20} + B(45)W_8^{25} + B(53)W_8^{30} + B(61)W_8^{35}$
5	6	$B(5)W_8^0 + B(13)W_8^6 + B(21)W_8^{12} + B(29)W_8^{18} + B(37)W_8^{24} + B(45)W_8^{30} + B(53)W_8^{36} + B(61)W_8^{42}$
5	7	$B(5)W_8^0 + B(13)W_8^7 + B(21)W_8^{14} + B(29)W_8^{21} + B(37)W_8^{28} + B(45)W_8^{35} + B(53)W_8^{42} + B(61)W_8^{49}$
6	0	$B(6)W_8^0 + B(14)W_8^0 + B(22)W_8^0 + B(30)W_8^0 + B(38)W_8^0 + B(46)W_8^0 + B(54)W_8^0 + B(62)W_8^0$
6	1	$B(6)W_8^0 + B(14)W_8^1 + B(22)W_8^2 + B(30)W_8^3 + B(38)W_8^4 + B(46)W_8^5 + B(54)W_8^6 + B(62)W_8^7$
6	2	$B(6)W_8^0 + B(14)W_8^2 + B(22)W_8^4 + B(30)W_8^6 + B(38)W_8^8 + B(46)W_8^{10} + B(54)W_8^{12} + B(62)W_8^{14}$
6	3	$B(6)W_8^0 + B(14)W_8^3 + B(22)W_8^6 + B(30)W_8^9 + B(38)W_8^{12} + B(46)W_8^{15} + B(54)W_8^{18} + B(62)W_8^{21}$
6	4	$B(6)W_8^0 + B(14)W_8^4 + B(22)W_8^8 + B(30)W_8^{12} + B(38)W_8^{16} + B(46)W_8^{20} + B(54)W_8^{24} + B(62)W_8^{28}$
6	5	$B(6)W_8^0 + B(14)W_8^5 + B(22)W_8^{10} + B(30)W_8^{15} + B(38)W_8^{20} + B(46)W_8^{25} + B(54)W_8^{30} + B(62)W_8^{35}$
6	6	$B(6)W_8^0 + B(14)W_8^6 + B(22)W_8^{12} + B(30)W_8^{18} + B(38)W_8^{24} + B(46)W_8^{30} + B(54)W_8^{36} + B(62)W_8^{42}$
6	7	$B(6)W_8^0 + B(14)W_8^7 + B(22)W_8^{14} + B(30)W_8^{21} + B(38)W_8^{28} + B(46)W_8^{35} + B(54)W_8^{42} + B(62)W_8^{49}$
7	0	$B(7)W_8^0 + B(15)W_8^0 + B(23)W_8^0 + B(31)W_8^0 + B(39)W_8^0 + B(47)W_8^0 + B(55)W_8^0 + B(63)W_8^0$
7	1	$B(7)W_8^0 + B(15)W_8^1 + B(23)W_8^2 + B(31)W_8^3 + B(39)W_8^4 + B(47)W_8^5 + B(55)W_8^6 + B(63)W_8^7$
7	2	$B(7)W_8^0 + B(15)W_8^2 + B(23)W_8^4 + B(31)W_8^6 + B(39)W_8^8 + B(47)W_8^{10} + B(55)W_8^{12} + B(63)W_8^{14}$
7	3	$B(7)W_8^0 + B(15)W_8^3 + B(23)W_8^6 + B(31)W_8^9 + B(39)W_8^{12} + B(47)W_8^{15} + B(55)W_8^{18} + B(63)W_8^{21}$
7	4	$B(7)W_8^0 + B(15)W_8^4 + B(23)W_8^8 + B(31)W_8^{12} + B(39)W_8^{16} + B(47)W_8^{20} + B(55)W_8^{24} + B(63)W_8^{28}$
7	5	$B(7)W_8^0 + B(15)W_8^5 + B(23)W_8^{10} + B(31)W_8^{15} + B(39)W_8^{20} + B(47)W_8^{25} + B(55)W_8^{30} + B(63)W_8^{35}$
7	6	$B(7)W_8^0 + B(15)W_8^6 + B(23)W_8^{12} + B(31)W_8^{18} + B(39)W_8^{24} + B(47)W_8^{30} + B(55)W_8^{36} + B(63)W_8^{42}$
7	7	$B(7)W_8^0 + B(15)W_8^7 + B(23)W_8^{14} + B(31)W_8^{21} + B(39)W_8^{28} + B(47)W_8^{35} + B(55)W_8^{42} + B(63)W_8^{49}$

Selanjutnya hasil kalkulasi FFT 8-titik tahap pertama ini disebut sebagai $[S(s)D(d)]$ dengan $S(s)$ menunjukkan nomor set dan $D(d)$ menunjukkan nomor data. Hasil kalkulasi tahap pertama akan masuk ke blok Interdimensional Constant Multiplier.

Tahap kedua FFT 64-titik dijelaskan sebagai berikut. Hasil perkalian $[S(s)D(d)]$ dengan twiddle faktor interdimensional diberikan sebagai berikut. Selain itu, nilai twiddle faktor interdimensional W_{64} diberikan pada tabel disampingnya.

Set	Data	Hasil Komputasi
0	0	$[S(0)D(0)]W_{64}^0$
0	1	$[S(0)D(1)]W_{64}^0$
0	2	$[S(0)D(2)]W_{64}^0$
0	3	$[S(0)D(3)]W_{64}^0$
0	4	$[S(0)D(4)]W_{64}^0$
0	5	$[S(0)D(5)]W_{64}^0$
0	6	$[S(0)D(6)]W_{64}^0$
0	7	$[S(0)D(7)]W_{64}^0$
1	0	$[S(1)D(0)]W_{64}^0$
1	1	$[S(1)D(1)]W_{64}^1$

W_{64}	Nilai
0	$1{,}000000000000000 + j0{,}000000000000000$
1	$0{,}995184726672197 - j0{,}098017140329561$
2	$0{,}980785280403230 - j0{,}195090322016128$
3	$0{,}956940335732209 - j0{,}290284677254462$
4	$0{,}923879532511287 - j0{,}382683432365090$
5	$0{,}881921264348355 - j0{,}471396736825998$
6	$0{,}831469612302545 - j0{,}555570233019602$
7	$0{,}773010453362737 - j0{,}634393284163645$
8	$0{,}707106781186548 - j0{,}707106781186547$
9	$0{,}634393284163645 - j0{,}773010453362737$

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

1	2	$[S(1)D(2)]W_{64}^{2}$
1	3	$[S(1)D(3)]W_{64}^{3}$
1	4	$[S(1)D(4)]W_{64}^{4}$
1	5	$[S(1)D(5)]W_{64}^{5}$
1	6	$[S(1)D(6)]W_{64}^{6}$
1	7	$[S(1)D(7)]W_{64}^{7}$
2	0	$[S(2)D(0)]W_{64}^{0}$
2	1	$[S(2)D(1)]W_{64}^{2}$
2	2	$[S(2)D(2)]W_{64}^{4}$
2	3	$[S(2)D(3)]W_{64}^{6}$
2	4	$[S(2)D(4)]W_{64}^{8}$
2	5	$[S(2)D(5)]W_{64}^{10}$
2	6	$[S(2)D(6)]W_{64}^{12}$
2	7	$[S(2)D(7)]W_{64}^{14}$
3	0	$[S(3)D(0)]W_{64}^{0}$
3	1	$[S(3)D(1)]W_{64}^{3}$
3	2	$[S(3)D(2)]W_{64}^{6}$
3	3	$[S(3)D(3)]W_{64}^{9}$
3	4	$[S(3)D(4)]W_{64}^{12}$
3	5	$[S(3)D(5)]W_{64}^{15}$
3	6	$[S(3)D(6)]W_{64}^{18}$
3	7	$[S(3)D(7)]W_{64}^{21}$
4	0	$[S(4)D(0)]W_{64}^{0}$
4	1	$[S(4)D(1)]W_{64}^{4}$
4	2	$[S(4)D(2)]W_{64}^{8}$
4	3	$[S(4)D(3)]W_{64}^{12}$
4	4	$[S(4)D(4)]W_{64}^{16}$
4	5	$[S(4)D(5)]W_{64}^{20}$
4	6	$[S(4)D(6)]W_{64}^{24}$
4	7	$[S(4)D(7)]W_{64}^{28}$
5	0	$[S(5)D(0)]W_{64}^{5}$
5	1	$[S(5)D(1)]W_{64}^{10}$
5	2	$[S(5)D(2)]W_{64}^{15}$
5	3	$[S(5)D(3)]W_{64}^{20}$
5	4	$[S(5)D(4)]W_{64}^{25}$
5	5	$[S(5)D(5)]W_{64}^{30}$
5	6	$[S(5)D(6)]W_{64}^{35}$
5	7	$[S(5)D(7)]W_{64}^{40}$
6	0	$[S(6)D(0)]W_{64}^{0}$
6	1	$[S(6)D(1)]W_{64}^{6}$
6	2	$[S(6)D(2)]W_{64}^{12}$
6	3	$[S(6)D(3)]W_{64}^{18}$
6	4	$[S(6)D(4)]W_{64}^{24}$
6	5	$[S(6)D(5)]W_{64}^{30}$
6	6	$[S(6)D(6)]W_{64}^{36}$
6	7	$[S(6)D(7)]W_{64}^{42}$
7	0	$[S(7)D(0)]W_{64}^{0}$
7	1	$[S(7)D(1)]W_{64}^{7}$
7	2	$[S(7)D(2)]W_{64}^{14}$
7	3	$[S(7)D(3)]W_{64}^{21}$
7	4	$[S(7)D(4)]W_{64}^{28}$
7	5	$[S(7)D(5)]W_{64}^{35}$

10	$0{,}555570233019602 - j0{,}831469612302545$
11	$0{,}471396736825998 - j0{,}881921264348355$
12	$0{,}382683432365090 - j0{,}923879532511287$
13	$0{,}290284677254462 - j0{,}956940335732209$
14	$0{,}195090322016128 - j0{,}980785280403230$
15	$0{,}098017140329561 - j0{,}995184726672197$
16	$0{,}000000000000000 - j1{,}000000000000000$
17	$-0{,}098017140329561 - j0{,}995184726672197$
18	$-0{,}195090322016128 - j0{,}980785280403230$
19	$-0{,}290284677254462 - j0{,}956940335732209$
20	$-0{,}382683432365090 - j0{,}923879532511287$
21	$-0{,}471396736825998 - j0{,}881921264348355$
22	$-0{,}555570233019602 - j0{,}831469612302545$
23	$-0{,}634393284163645 - j0{,}773010453362737$
24	$-0{,}707106781186547 - j0{,}707106781186548$
25	$-0{,}773010453362737 - j0{,}634393284163645$
26	$-0{,}831469612302545 - j0{,}555570233019602$
27	$-0{,}881921264348355 - j0{,}471396736825998$
28	$-0{,}923879532511287 - j0{,}382683432365090$
29	$-0{,}956940335732209 - j0{,}290284677254462$
30	$-0{,}980785280403230 - j0{,}195090322016129$
31	$-0{,}995184726672197 - j0{,}098017140329561$
32	$-1{,}000000000000000 + j0{,}000000000000000$
33	$-0{,}995184726672197 + j0{,}098017140329561$
34	$-0{,}980785280403230 + j0{,}195090322016128$
35	$-0{,}956940335732209 + j0{,}290284677254462$
36	$-0{,}923879532511287 + j0{,}382683432365090$
37	$-0{,}881921264348355 + j0{,}471396736825998$
38	$-0{,}831469612302545 + j0{,}555570233019602$
39	$-0{,}773010453362737 + j0{,}634393284163645$
40	$-0{,}707106781186548 + j0{,}707106781186547$
41	$-0{,}634393284163646 + j0{,}773010453362737$
42	$-0{,}555570233019602 + j0{,}831469612302545$
43	$-0{,}471396736825998 + j0{,}881921264348355$
44	$-0{,}382683432365090 + j0{,}923879532511287$
45	$-0{,}290284677254462 + j0{,}956940335732209$
46	$-0{,}195090322016129 + j0{,}980785280403230$
47	$-0{,}098017140329561 + j0{,}995184726672197$
48	$0{,}000000000000000 + j1{,}000000000000000$
49	$0{,}098017140329560 + j0{,}995184726672197$
50	$0{,}195090322016128 + j0{,}980785280403230$
51	$0{,}290284677254462 + j0{,}956940335732209$
52	$0{,}382683432365090 + j0{,}923879532511287$
53	$0{,}471396736825998 + j0{,}881921264348355$
54	$0{,}555570233019602 + j0{,}831469612302545$
55	$0{,}634393284163646 + j0{,}773010453362737$
56	$0{,}707106781186547 + j0{,}707106781186548$
57	$0{,}773010453362737 + j0{,}634393284163646$
58	$0{,}831469612302545 + j0{,}555570233019602$
59	$0{,}881921264348355 + j0{,}471396736825998$
60	$0{,}923879532511287 + j0{,}382683432365090$
61	$0{,}956940335732209 + j0{,}290284677254462$

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

7	6	$[S(7)D(6)]W_{64}^{42}$
7	7	$[S(7)D(7)]W_{64}^{49}$

62	$0{,}980785280403230 + j0{,}195090322016129$
63	$0{,}995184726672197 + j0{,}098017140329561$

Dari 64 buah Twiddle Factor Interdimensional tersebut, hanya akan digunakan sebagai 49 buah. Namun, ke-49 buah twiddle factor dapat diwakili oleh delapan buah twiddle factor paling awal, yaitu W_{64}^{0} , W_{64}^{1} , W_{64}^{2} , W_{64}^{3} , W_{64}^{4} , W_{64}^{5} , W_{64}^{6} , W_{64}^{7} , dan W_{64}^{8} dengan melakukan pembalikan tanda bagian real, melakukan pembalikan tanda pada bagian imaginer, atau menukar bilangan real dan bilangan imaginer. Hal ini akan dibahas lebih lanjut pada saat implementasi Blok Interdimensional Multiplier. Selanjutnya hasil perkalian dengan twiddle factor interdimensional ini disebut sebagai $[S'(s)D'(d)]$ dengan $S'(s)$ menunjukkan nomor set dan $D'(d)$ menunjukkan nomor data. Hasil kalkulasi ini kemudian memasuki FFT 8-titik tahap kedua untuk diproses lebih lanjut.

Tahap ketiga FFT 64-titik dijelaskan sebagai berikut. Komputasi dilakukan untuk data input $[S'(0)D'(0)]$hingga $[S'(7)D'(7)]$. Aturan komputasi menurut struktur blok input buffer dilakukan sebagai berikut.

Set	Input Data
0	$[S'(0)D'(0)], [S'(1)D'(0)], [S'(2)D'(0)], [S'(3)D'(0)], [S'(4)D'(0)], [S'(5)D'(0)], [S'(6)D'(0)], [S'(7)D'(0)]$
1	$[S'(0)D'(1)], [S'(1)D'(1)], [S'(2)D'(1)], [S'(3)D'(1)], [S'(4)D'(1)], [S'(5)D'(1)], [S'(6)D'(1)], [S'(7)D'(1)]$
2	$[S'(0)D'(2)], [S'(1)D'(2)], [S'(2)D'(2)], [S'(3)D'(2)], [S'(4)D'(2)], [S'(5)D'(2)], [S'(6)D'(2)], [S'(7)D'(2)]$
3	$[S'(0)D'(3)], [S'(1)D'(3)], [S'(2)D'(3)], [S'(3)D'(3)], [S'(4)D'(3)], [S'(5)D'(3)], [S'(6)D'(3)], [S'(7)D'(3)]$
4	$[S'(0)D'(4)], [S'(1)D'(4)], [S'(2)D'(4)], [S'(3)D'(4)], [S'(4)D'(4)], [S'(5)D'(4)], [S'(6)D'(4)], [S'(7)D'(4)]$
5	$[S'(0)D'(5)], [S'(1)D'(5)], [S'(2)D'(5)], [S'(3)D'(5)], [S'(4)D'(5)], [S'(5)D'(5)], [S'(6)D'(5)], [S'(7)D'(5)]$
6	$[S'(0)D'(6)], [S'(1)D'(6)], [S'(2)D'(6)], [S'(3)D'(6)], [S'(4)D'(6)], [S'(5)D'(6)], [S'(6)D'(6)], [S'(7)D'(6)]$
7	$[S'(0)D'(7)], [S'(1)D'(7)], [S'(2)D'(7)], [S'(3)D'(7)], [S'(4)D'(7)], [S'(5)D'(7)], [S'(6)D'(7)], [S'(7)D'(7)]$

Dengan mengingat bahwa $W_N^n = W_N^{n+mN}$ dengan $n \in \{0\,..\,N-1\}$ dan $m \in \mathbb{Z}$, berikut ini adalah hasil operasi yang dilakukan oleh FFT 8-titik tahap pertama.

A	Hasil Komputasi
$A[0]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^0 + [S'(2)D'(0)]W_8^0 + [S'(3)D'(0)]W_8^0 + [S'(4)D'(0)]W_8^0 + [S'(5)D'(0)]W_8^0 + [S'(6)D'(0)]W_8^0 + [S'(7)D'(0)]W_8^0$
$A[1]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^1 + [S'(2)D'(0)]W_8^2 + [S'(3)D'(0)]W_8^3 + [S'(4)D'(0)]W_8^4 + [S'(5)D'(0)]W_8^5 + [S'(6)D'(0)]W_8^6 + [S'(7)D'(0)]W_8^7$
$A[2]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^2 + [S'(2)D'(0)]W_8^4 + [S'(3)D'(0)]W_8^6 + [S'(4)D'(0)]W_8^8 + [S'(5)D'(0)]W_8^{10} + [S'(6)D'(0)]W_8^{12} + [S'(7)D'(0)]W_8^{14}$
$A[3]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^3 + [S'(2)D'(0)]W_8^6 + [S'(3)D'(0)]W_8^9 + [S'(4)D'(0)]W_8^{12} + [S'(5)D'(0)]W_8^{15} + [S'(6)D'(0)]W_8^{18} + [S'(7)D'(0)]W_8^{21}$
$A[4]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^4 + [S'(2)D'(0)]W_8^8 + [S'(3)D'(0)]W_8^{12} + [S'(4)D'(0)]W_8^{16} + [S'(5)D'(0)]W_8^{20} + [S'(6)D'(0)]W_8^{24} + [S'(7)D'(0)]W_8^{28}$
$A[5]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^5 + [S'(2)D'(0)]W_8^{10} + [S'(3)D'(0)]W_8^{15} + [S'(4)D'(0)]W_8^{20} + [S'(5)D'(0)]W_8^{25} + [S'(6)D'(0)]W_8^{30} + [S'(7)D'(0)]W_8^{35}$
$A[6]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^6 + [S'(2)D'(0)]W_8^{12} + [S'(3)D'(0)]W_8^{18} + [S'(4)D'(0)]W_8^{24} + [S'(5)D'(0)]W_8^{30} + [S'(6)D'(0)]W_8^{36} + [S'(7)D'(0)]W_8^{42}$
$A[7]$	$[S'(0)D'(0)]W_8^0 + [S'(1)D'(0)]W_8^7 + [S'(2)D'(0)]W_8^{14} + [S'(3)D'(0)]W_8^{21} + [S'(4)D'(0)]W_8^{28} + [S'(5)D'(0)]W_8^{35} + [S'(6)D'(0)]\ W_8^{42} + [S'(7)D'(0)]W_8^{49}$
$A[8]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^0 + [S'(2)D'(1)]W_8^0 + [S'(3)D'(1)]W_8^0 + [S'(4)D'(1)]W_8^0 + [S'(5)D'(1)]W_8^0 + [S'(6)D'(1)]W_8^0 + [S'(7)D'(1)]W_8^0$
$A[9]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^1 + [S'(2)D'(1)]W_8^2 + [S'(3)D'(1)]W_8^3 + [S'(4)D'(1)]W_8^4 + [S'(5)D'(1)]W_8^5 + [S'(6)D'(1)]W_8^6 + [S'(7)D'(1)]W_8^7$
$A[10]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^2 + [S'(2)D'(1)]W_8^4 + [S'(3)D'(1)]W_8^6 + [S'(4)D'(1)]W_8^8 + [S'(5)D'(1)]W_8^{10} + [S'(6)D'(1)]W_8^{12} + [S'(7)D'(1)]W_8^{14}$
$A[11]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^3 + [S'(2)D'(1)]W_8^6 + [S'(3)D'(1)]W_8^9 + [S'(4)D'(1)]W_8^{12} + [S'(5)D'(1)]W_8^{15} + [S'(6)D'(1)]W_8^{18} + [S'(7)D'(1)]W_8^{21}$
$A[12]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^4 + [S'(2)D'(1)]W_8^8 + [S'(3)D'(1)]W_8^{12} + [S'(4)D'(1)]W_8^{16} + [S'(5)D'(1)]W_8^{20} + [S'(6)D'(1)]W_8^{24} + [S'(7)D'(1)]W_8^{28}$
$A[13]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^5 + [S'(2)D'(1)]W_8^{10} + [S'(3)D'(1)]W_8^{15} + [S'(4)D'(1)]W_8^{20} + [S'(5)D'(1)]W_8^{25} + [S'(6)D'(1)]W_8^{30} + [S'(7)D'(1)]W_8^{35}$

$A[14]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^6 + [S'(2)D'(1)]W_8^{12} + [S'(3)D'(1)]W_8^{18} + [S'(4)D'(1)]W_8^{24} + [S'(5)D'(1)]W_8^{30} + [S'(6)D'(1)]W_8^{36} + [S'(7)D'(1)]W_8^{42}$
$A[15]$	$[S'(0)D'(1)]W_8^0 + [S'(1)D'(1)]W_8^7 + [S'(2)D'(1)]W_8^{14} + [S'(3)D'(1)]W_8^{21} + [S'(4)D'(1)]W_8^{28} + [S'(5)D'(1)]W_8^{35} + [S'(6)D'(1)]\ W_8^{42} + [S'(7)D'(1)]W_8^{49}$
$A[16]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^0 + [S'(2)D'(2)]W_8^0 + [S'(3)D'(2)]W_8^0 + [S'(4)D'(2)]W_8^0 + [S'(5)D'(2)]W_8^0 + [S'(6)D'(2)]W_8^0 + [S'(7)D'(2)]W_8^0$
$A[17]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^1 + [S'(2)D'(2)]W_8^2 + [S'(3)D'(2)]W_8^3 + [S'(4)D'(2)]W_8^4 + [S'(5)D'(2)]W_8^5 + [S'(6)D'(2)]W_8^6 + [S'(7)D'(2)]W_8^7$
$A[18]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^2 + [S'(2)D'(2)]W_8^4 + [S'(3)D'(2)]W_8^6 + [S'(4)D'(2)]W_8^8 + [S'(5)D'(2)]W_8^{10} + [S'(6)D'(2)]W_8^{12} + [S'(7)D'(2)]W_8^{14}$
$A[19]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^3 + [S'(2)D'(2)]W_8^6 + [S'(3)D'(2)]W_8^9 + [S'(4)D'(2)]W_8^{12} + [S'(5)D'(2)]W_8^{15} + [S'(6)D'(2)]W_8^{18} + [S'(7)D'(2)]W_8^{21}$
$A[20]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^4 + [S'(2)D'(2)]W_8^8 + [S'(3)D'(2)]W_8^{12} + [S'(4)D'(2)]W_8^{16} + [S'(5)D'(2)]W_8^{20} + [S'(6)D'(2)]W_8^{24} + [S'(7)D'(2)]W_8^{28}$
$A[21]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^5 + [S'(2)D'(2)]W_8^{10} + [S'(3)D'(2)]W_8^{15} + [S'(4)D'(2)]W_8^{20} + [S'(5)D'(2)]W_8^{25} + [S'(6)D'(2)]W_8^{30} + [S'(7)D'(2)]W_8^{35}$
$A[22]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^6 + [S'(2)D'(2)]W_8^{12} + [S'(3)D'(2)]W_8^{18} + [S'(4)D'(2)]W_8^{24} + [S'(5)D'(2)]W_8^{30} + [S'(6)D'(2)]W_8^{36} + [S'(7)D'(2)]W_8^{42}$
$A[23]$	$[S'(0)D'(2)]W_8^0 + [S'(1)D'(2)]W_8^7 + [S'(2)D'(2)]W_8^{14} + [S'(3)D'(2)]W_8^{21} + [S'(4)D'(2)]W_8^{28} + [S'(5)D'(2)]W_8^{35} + [S'(6)D'(2)]\ W_8^{42} + [S'(7)D'(2)]W_8^{49}$
$A[24]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^0 + [S'(2)D'(3)]W_8^0 + [S'(3)D'(3)]W_8^0 + [S'(4)D'(3)]W_8^0 + [S'(5)D'(3)]W_8^0 + [S'(6)D'(3)]W_8^0 + [S'(7)D'(3)]W_8^0$
$A[25]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^1 + [S'(2)D'(3)]W_8^2 + [S'(3)D'(3)]W_8^3 + [S'(4)D'(3)]W_8^4 + [S'(5)D'(3)]W_8^5 + [S'(6)D'(3)]W_8^6 + [S'(7)D'(3)]W_8^7$
$A[26]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^2 + [S'(2)D'(3)]W_8^4 + [S'(3)D'(3)]W_8^6 + [S'(4)D'(3)]W_8^8 + [S'(5)D'(3)]W_8^{10} + [S'(6)D'(3)]W_8^{12} + [S'(7)D'(3)]W_8^{14}$
$A[27]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^3 + [S'(2)D'(3)]W_8^6 + [S'(3)D'(3)]W_8^9 + [S'(4)D'(3)]W_8^{12} + [S'(5)D'(3)]W_8^{15} + [S'(6)D'(3)]W_8^{18} + [S'(7)D'(3)]W_8^{21}$
$A[28]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^4 + [S'(2)D'(3)]W_8^8 + [S'(3)D'(3)]W_8^{12} + [S'(4)D'(3)]W_8^{16} + [S'(5)D'(3)]W_8^{20} + [S'(6)D'(3)]W_8^{24} + [S'(7)D'(3)]W_8^{28}$
$A[29]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^5 + [S'(2)D'(3)]W_8^{10} + [S'(3)D'(3)]W_8^{15} + [S'(4)D'(3)]W_8^{20} + [S'(5)D'(3)]W_8^{25} + [S'(6)D'(3)]W_8^{30} + [S'(7)D'(3)]W_8^{35}$
$A[30]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^6 + [S'(2)D'(3)]W_8^{12} + [S'(3)D'(3)]W_8^{18} + [S'(4)D'(3)]W_8^{24} + [S'(5)D'(3)]W_8^{30} + [S'(6)D'(3)]W_8^{36} + [S'(7)D'(3)]W_8^{42}$
$A[31]$	$[S'(0)D'(3)]W_8^0 + [S'(1)D'(3)]W_8^7 + [S'(2)D'(3)]W_8^{14} + [S'(3)D'(3)]W_8^{21} + [S'(4)D'(3)]W_8^{28} + [S'(5)D'(3)]W_8^{35} + [S'(6)D'(3)]\ W_8^{42} + [S'(7)D'(3)]W_8^{49}$
$A[32]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^0 + [S'(2)D'(4)]W_8^0 + [S'(3)D'(4)]W_8^0 + [S'(4)D'(4)]W_8^0 + [S'(5)D'(4)]W_8^0 + [S'(6)D'(4)]W_8^0 + [S'(7)D'(4)]W_8^0$
$A[33]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^1 + [S'(2)D'(4)]W_8^2 + [S'(3)D'(4)]W_8^3 + [S'(4)D'(4)]W_8^4 + [S'(5)D'(4)]W_8^5 + [S'(6)D'(4)]W_8^6 + [S'(7)D'(4)]W_8^7$
$A[34]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^2 + [S'(2)D'(4)]W_8^4 + [S'(3)D'(4)]W_8^6 + [S'(4)D'(4)]W_8^8 + [S'(5)D'(4)]W_8^{10} + [S'(6)D'(4)]W_8^{12} + [S'(7)D'(4)]W_8^{14}$
$A[35]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^3 + [S'(2)D'(4)]W_8^6 + [S'(3)D'(4)]W_8^9 + [S'(4)D'(4)]W_8^{12} + [S'(5)D'(4)]W_8^{15} + [S'(6)D'(4)]W_8^{18} + [S'(7)D'(4)]W_8^{21}$
$A[36]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^4 + [S'(2)D'(4)]W_8^8 + [S'(3)D'(4)]W_8^{12} + [S'(4)D'(4)]W_8^{16} + [S'(5)D'(4)]W_8^{20} + [S'(6)D'(4)]W_8^{24} + [S'(7)D'(4)]W_8^{28}$
$A[37]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^5 + [S'(2)D'(4)]W_8^{10} + [S'(3)D'(4)]W_8^{15} + [S'(4)D'(4)]W_8^{20} + [S'(5)D'(4)]W_8^{25} + [S'(6)D'(4)]W_8^{30} + [S'(7)D'(4)]W_8^{35}$
$A[38]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^6 + [S'(2)D'(4)]W_8^{12} + [S'(3)D'(4)]W_8^{18} + [S'(4)D'(4)]W_8^{24} + [S'(5)D'(4)]W_8^{30} + [S'(6)D'(4)]W_8^{36} + [S'(7)D'(4)]W_8^{42}$
$A[39]$	$[S'(0)D'(4)]W_8^0 + [S'(1)D'(4)]W_8^7 + [S'(2)D'(4)]W_8^{14} + [S'(3)D'(4)]W_8^{21} + [S'(4)D'(4)]W_8^{28} + [S'(5)D'(4)]W_8^{35} + [S'(6)D'(4)]\ W_8^{42} + [S'(7)D'(4)]W_8^{49}$
$A[40]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^0 + [S'(2)D'(5)]W_8^0 + [S'(3)D'(5)]W_8^0 + [S'(4)D'(5)]W_8^0 + [S'(5)D'(5)]W_8^0 + [S'(6)D'(5)]W_8^0 + [S'(7)D'(5)]W_8^0$
$A[41]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^1 + [S'(2)D'(5)]W_8^2 + [S'(3)D'(5)]W_8^3 + [S'(4)D'(5)]W_8^4 + [S'(5)D'(5)]W_8^5 + [S'(6)D'(5)]W_8^6 + [S'(7)D'(5)]W_8^7$
$A[42]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^2 + [S'(2)D'(5)]W_8^4 + [S'(3)D'(5)]W_8^6 + [S'(4)D'(5)]W_8^8 + [S'(5)D'(5)]W_8^{10} + [S'(6)D'(5)]W_8^{12} + [S'(7)D'(5)]W_8^{14}$
$A[43]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^3 + [S'(2)D'(5)]W_8^6 + [S'(3)D'(5)]W_8^9 + [S'(4)D'(5)]W_8^{12} + [S'(5)D'(5)]W_8^{15} + [S'(6)D'(5)]W_8^{18} + [S'(7)D'(5)]W_8^{21}$
$A[44]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^4 + [S'(2)D'(5)]W_8^8 + [S'(3)D'(5)]W_8^{12} + [S'(4)D'(5)]W_8^{16} + [S'(5)D'(5)]W_8^{20} + [S'(6)D'(5)]W_8^{24} + [S'(7)D'(5)]W_8^{28}$
$A[45]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5))]W_8^5 + [S'(2)D'(5)]W_8^{10} + [S'(3)D'(5)]W_8^{15} + [S'(4)D'(5)]W_8^{20} + [S'(5)D'(5)]W_8^{25} + [S'(6)D'(5)]W_8^{30} + [S'(7)D'(5)]W_8^{35}$
$A[46]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^6 + [S'(2)D'(5)]W_8^{12} + [S'(3)D'(5)]W_8^{18} + [S'(4)D'(5)]W_8^{24} + [S'(5)D'(5)]W_8^{30} + [S'(6)D'(5)]W_8^{36} + [S'(7)D'(5)]W_8^{42}$
$A[47]$	$[S'(0)D'(5)]W_8^0 + [S'(1)D'(5)]W_8^7 + [S'(2)D'(5)]W_8^{14} + [S'(3)D'(5)]W_8^{21} + [S'(4)D'(5)]W_8^{28} + [S'(5)D'(5)]W_8^{35} + [S'(6)D'(5)]\ W_8^{42} + [S'(7)D'(5)]W_8^{49}$
$A[48]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^0 + [S'(2)D'(6)]W_8^0 + [S'(3)D'(6)]W_8^0 + [S'(4)D'(6)]W_8^0 + [S'(5)D'(6)]W_8^0 + [S'(6)D'(6)]W_8^0 + [S'(7)D'(6)]W_8^0$
$A[49]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^1 + [S'(2)D'(6)]W_8^2 + [S'(3)D'(6)]W_8^3 + [S'(4)D'(6)]W_8^4 + [S'(5)D'(6)]W_8^5 + [S'(6)D'(6)]W_8^6 + [S'(7)D'(6)]W_8^7$

$A[50]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^2 + [S'(2)D'(6)]W_8^4 + [S'(3)D'(6)]W_8^6 + [S'(4)D'(6)]W_8^8 + [S'(5)D'(6)]W_8^{10} + [S'(6)D'(6)]W_8^{12} + [S'(7)D'(6)]W_8^{14}$
$A[51]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^3 + [S'(2)D'(6)]W_8^6 + [S'(3)D'(6)]W_8^9 + [S'(4)D'(6)]W_8^{12} + [S'(5)D'(6)]W_8^{15} + [S'(6)D'(6)]W_8^{18} + [S'(7)D'(6)]W_8^{21}$
$A[52]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^4 + [S'(2)D'(6)]W_8^8 + [S'(3)D'(6)]W_8^{12} + [S'(4)D'(6)]W_8^{16} + [S'(5)D'(6)]W_8^{20} + [S'(6)D'(6)]W_8^{24} + [S'(7)D'(6)]W_8^{28}$
$A[53]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6))]W_8^5 + [S'(2)D'(6)]W_8^{10} + [S'(3)D'(6)]W_8^{15} + [S'(4)D'(6)]W_8^{20} + [S'(5)D'(6)]W_8^{25} + [S'(6)D'(6)]W_8^{30} + [S'(7)D'(6)]W_8^{35}$
$A[54]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^6 + [S'(2)D'(6)]W_8^{12} + [S'(3)D'(6)]W_8^{18} + [S'(4)D'(6)]W_8^{24} + [S'(5)D'(6)]W_8^{30} + [S'(6)D'(6)]W_8^{36} + [S'(7)D'(6)]W_8^{42}$
$A[55]$	$[S'(0)D'(6)]W_8^0 + [S'(1)D'(6)]W_8^7 + [S'(2)D'(6)]W_8^{14} + [S'(3)D'(6)]W_8^{21} + [S'(4)D'(6)]W_8^{28} + [S'(5)D'(6)]W_8^{35} + [S'(6)D'(6)]\ W_8^{42} + [S'(7)D'(6)]W_8^{49}$
$A[56]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^0 + [S'(2)D'(7)]W_8^0 + [S'(3)D'(7)]W_8^0 + [S'(4)D'(7)]W_8^0 + [S'(5)D'(7)]W_8^0 + [S'(6)D'(7)]W_8^0 + [S'(7)D'(7)]W_8^0$
$A[57]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^1 + [S'(2)D'(7)]W_8^2 + [S'(3)D'(7)]W_8^3 + [S'(4)D'(7)]W_8^4 + [S'(5)D'(7)]W_8^5 + [S'(6)D'(7)]W_8^6 + [S'(7)D'(7)]W_8^7$
$A[58]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^2 + [S'(2)D'(7)]W_8^4 + [S'(3)D'(7)]W_8^6 + [S'(4)D'(7)]W_8^8 + [S'(5)D'(7)]W_8^{10} + [S'(6)D'(7)]W_8^{12} + [S'(7)D'(7)]W_8^{14}$
$A[59]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^3 + [S'(2)D'(7)]W_8^6 + [S'(3)D'(7)]W_8^9 + [S'(4)D'(7)]W_8^{12} + [S'(5)D'(7)]W_8^{15} + [S'(6)D'(7)]W_8^{18} + [S'(7)D'(7)]W_8^{21}$
$A[60]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^4 + [S'(2)D'(7)]W_8^8 + [S'(3)D'(7)]W_8^{12} + [S'(4)D'(7)]W_8^{16} + [S'(5)D'(7)]W_8^{20} + [S'(6)D'(7)]W_8^{24} + [S'(7)D'(7)]W_8^{28}$
$A[61]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D(7)]W_8^5 + [S'(2)D'(7)]W_8^{10} + [S'(3)D'(7)]W_8^{15} + [S'(4)D'(7)]W_8^{20} + [S'(5)D'(7)]W_8^{25} + [S'(6)D'(7)]W_8^{30} + [S'(7)D'(7)]W_8^{35}$
$A[62]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^6 + [S'(2)D'(7)]W_8^{12} + [S'(3)D'(7)]W_8^{18} + [S'(4)D'(7)]W_8^{24} + [S'(5)D'(7)]W_8^{30} + [S'(6)D'(7)]W_8^{36} + [S'(7)D'(7)]W_8^{42}$
$A[63]$	$[S'(0)D'(7)]W_8^0 + [S'(1)D'(7)]W_8^7 + [S'(2)D'(7)]W_8^{14} + [S'(3)D'(7)]W_8^{21} + [S'(4)D'(7)]W_8^{28} + [S'(5)D'(7)]W_8^{35} + [S'(6)D'(7)]\ W_8^{42} + [S'(7)D'(7)]W_8^{49}$

Dengan demikian, terdapat 64 buah data yang akan diolah di dalam prosesor FFT/IFFT 64-titik untuk menghasilkan data baru. Dengan menggunakan hubungan matematika antara input dan output beserta arsitektur rangkaian yang akan digunakan, kita akan mendesain rangkaian prosesor FFT/IFFT 64-titik secara structural.

Selain itu, dari hubungan input dan output tersebut, kita dapat membuat program dalam MATLAB untuk melakukan perhitungan FFT 64-titik. Hasil perhitungan ini dapat digunakan untuk melakukan validasi terhadap fungsionalitas rangkaian secara keseluruhan.

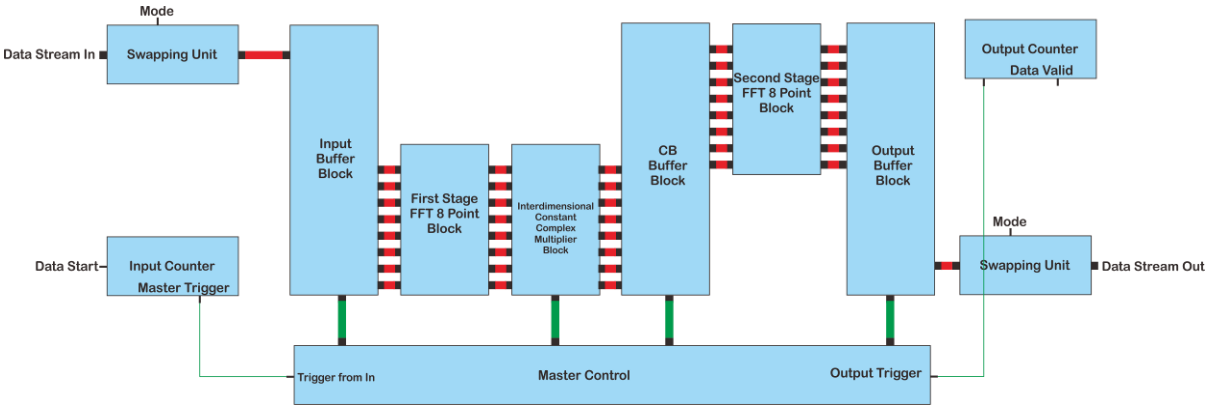
```
% FFT 64-point (C) Bagus Hanindhito (13211007)

% Define Mode %0 for FFT, 1 for IFFT
mode = 0;
% Twiddle Factor for 8 point FFT
Wn8 = zeros(1,64);
for i=1:1:64
       Wn8(i) = cos(-2*pi*(i-1)/8)+j*sin(-2*pi*(i-1)/8);
end;
% Wn8_1 = (sqrt(2)/2)-j*(sqrt(2)/2);
% Wn8_2 = 0 - j*1;
% Wn8_3 = -(sqrt(2)/2)-j*(sqrt(2)/2);

% Twiddle Factor for 64 point FFT
Wn64 = zeros(1,64);
for i=1:1:64
    Wn64(i)=cos(-2*pi*(i-1)/64)+j*sin(-2*pi*(i-1)/64);
end;

% Input Vector (Manual Input)
B64 = zeros(1,64);
B64(1)  = 0.001 + j*0.001;
B64(2)  = 0.002 + j*0.002;
B64(3)  = 0.003 + j*0.003;
B64(4)  = 0.004 + j*0.004;
B64(5)  = 0.005 + j*0.005;
B64(6)  = 0.006 + j*0.006;
```

```
B64(7)  = 0.007 + j*0.007;
B64(8)  = 0.008 + j*0.008;
B64(9)  = -0.001 + j*0.001;
B64(10) = -0.002 + j*0.002;
B64(11) = -0.003 + j*0.003;
B64(12) = -0.004 + j*0.004;
B64(13) = -0.005 + j*0.005;
B64(14) = -0.006 + j*0.006;
B64(15) = -0.007 + j*0.007;
B64(16) = -0.008 + j*0.008;
B64(17) = 0.001 - j*0.001;
B64(18) = 0.002 - j*0.002;
B64(19) = 0.003 - j*0.003;
B64(20) = 0.004 - j*0.004;
B64(21) = 0.005 - j*0.005;
B64(22) = 0.006 - j*0.006;
B64(23) = 0.007 - j*0.007;
B64(24) = 0.008 - j*0.008;
B64(25) = -0.001 - j*0.001;
B64(26) = -0.002 - j*0.002;
B64(27) = -0.003 - j*0.003;
B64(28) = -0.004 - j*0.004;
B64(29) = -0.005 - j*0.005;
B64(30) = -0.006 - j*0.006;
B64(31) = -0.007 - j*0.007;
B64(32) = -0.008 - j*0.008;
B64(33) = 0.001 + j*0.001;
B64(34) = 0.002 + j*0.002;
B64(35) = 0.003 + j*0.003;
B64(36) = 0.004 + j*0.004;
B64(37) = 0.005 + j*0.005;
B64(38) = 0.006 + j*0.006;
B64(39) = 0.007 + j*0.007;
B64(40) = 0.008 + j*0.008;
B64(41) = -0.001 + j*0.001;
B64(42) = -0.002 + j*0.002;
B64(43) = -0.003 + j*0.003;
B64(44) = -0.004 + j*0.004;
B64(45) = -0.005 + j*0.005;
B64(46) = -0.006 + j*0.006;
B64(47) = -0.007 + j*0.007;
B64(48) = -0.008 + j*0.008;
B64(49) = 0.001 - j*0.001;
B64(50) = 0.002 - j*0.002;
B64(51) = 0.003 - j*0.003;
B64(52) = 0.004 - j*0.004;
B64(53) = 0.005 - j*0.005;
B64(54) = 0.006 - j*0.006;
B64(55) = 0.007 - j*0.007;
B64(56) = 0.008 - j*0.008;
B64(57) = -0.001 - j*0.001;
B64(58) = -0.002 - j*0.002;
B64(59) = -0.003 - j*0.003;
B64(60) = -0.004 - j*0.004;
B64(61) = -0.005 - j*0.005;
B64(62) = -0.006 - j*0.006;
B64(63) = -0.007 - j*0.007;
B64(64) = -0.008 - j*0.008;

%Swapping Unit for IFFT
if(mode==1)
    for i=1:1:64
        B64(i) = imag(B64(i))+j*real(B64(i));
    end;
end;

% Convert to Fixed Point in Hexadecimal
B64_Fixed = cell(1,64);
```

```
for i=1:1:64
       B64_real_temp = fi(real(B64(i)),1,16,12);
       B64_imag_temp = fi(imag(B64(i)),1,16,12);
    B64_Fixed(i)  =
cellstr(dec2hex(bin2dec(strcat(B64_real_temp.bin,B64_imag_temp.bin)),8));
end;
B64_Fixed = transpose(B64_Fixed);

% Computing Equation (3)
A64 = zeros(1,64);
T64 = zeros(1,64);
T64_1 = zeros(1,64);
for(t=0:1:7)
       for(s=0:1:7)
              temp2 = 0;
              for(l=0:1:7)
                     temp =0;
                     for(m=0:1:7)
                            temp = temp+B64(l+8*m+1)*Wn8(s*m+1);
                     end;
             T64(s+8*l+1) = temp;
                     T64_1(s+8*l+1) = temp*Wn64(s*l+1);
                     temp2 = temp2 + temp*Wn64(s*l+1)*Wn8(l*t+1);
              end;
              A64(s+8*t+1) = temp2;
       end;
end;

% Intermediate Data
T64_Fixed = cell(1,64);
for i=1:1:64
       T64_real_temp = fi(real(T64(i)),1,16,12);
       T64_imag_temp = fi(imag(T64(i)),1,16,12);
    T64_Fixed(i)  =
cellstr(dec2hex(bin2dec(strcat(T64_real_temp.bin,T64_imag_temp.bin)),8));
end;
T64_Fixed = transpose(T64_Fixed);

T64_1_Fixed = cell(1,64);
for i=1:1:64
       T64_1_real_temp = fi(real(T64_1(i)),1,16,12);
       T64_1_imag_temp = fi(imag(T64_1(i)),1,16,12);
    T64_1_Fixed(i)  =
cellstr(dec2hex(bin2dec(strcat(T64_1_real_temp.bin,T64_1_imag_temp.bin)),8));
end;
T64_1_Fixed = transpose(T64_1_Fixed);

%Swapping Unit for IFFT
if(mode==1)
    for i=1:1:64
        A64(i) = imag(A64(i))+j*real(A64(i));
    end;
end;

% Convert Fixed Point in Hexadecimal
A64_Fixed = cell(1,64);
for i=1:1:64
       A64_real_temp = fi(real(A64(i)),1,16,12);
       A64_imag_temp = fi(imag(A64(i)),1,16,12);
    A64_Fixed(i)  =
cellstr(dec2hex(bin2dec(strcat(A64_real_temp.bin,A64_imag_temp.bin)),8));
end;
A64_Fixed = transpose(A64_Fixed);
```

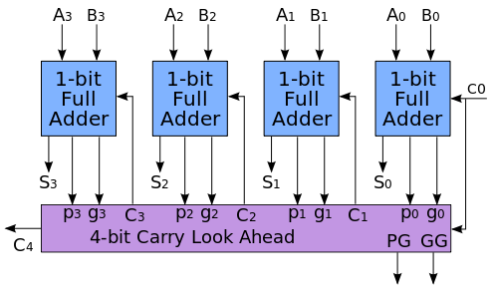
B. Arsitektur FFT/IFFT 64-titik dan Komponen Dasar

Berikut ini adalah diagram blok pada level tertinggi yang menggambarkan struktur rangkaian FFT/IFFT 64-titik yang akan dibuat. Rangkaian ini akan dibuat perbagian sebelum pada akhirnya semua bagian tersebut digabungkan. Hal ini dilakukan untuk menyederhanakan kompleksitas dari rangkaian dan mempermudah pengujian rangkaian dan sub rangkaian tersebut.



Untuk membuat rangkaian tersebut, terdapat beberapa komponen pembangun dasar dari rangkaian tersebut. Komponen pembangun dasar ini akan digunakan secara berulang-ulang untuk merealisasikan rangkaian yang lebih kompleks. Beberapa elemen dasar tersebut adalah sebagai berikut.

1. Implementasi Adder



Komponen dasar dari FFT 8-point ini adalah Adder. Realisasi adder dilakukan menggunakan Carry-Lookahead Adder. Terdapat dua buah adder yang akan direalisasikan dalam FFT 8-point ini yaitu Adder 16-bit yang digunakan secara umum dalam hardware dan Adder 32-bit yang digunakan pada complex multiplier. Berikut ini kode VHDL untuk Carry Lookahead Adder 1-bit, 16-bit, dan 32-bit.

adder_cla_1b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY adder_cla_1b IS
  PORT  (
            A         :  IN    STD_LOGIC;
            B         :  IN    STD_LOGIC;
            C_IN      :  IN    STD_LOGIC;
            R         :  OUT   STD_LOGIC;
            P         :  OUT   STD_LOGIC;
            G         :  OUT   STD_LOGIC
        );
END adder_cla_1b;
ARCHITECTURE structural of adder_cla_1b IS
BEGIN
  R     <=  A XOR B XOR C_IN;
  P     <=  A OR B;
  G     <=  A AND B;
END structural;
```

adder_cla_16b.vhd

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

```
ENTITY adder_cla_16b IS
  PORT  (
           A16      :  IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
           B16      :  IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
           R16      :  OUT   STD_LOGIC_VECTOR (15 DOWNTO 0);
           C_OUT16  :  OUT   STD_LOGIC
         );
END adder_cla_16b;
ARCHITECTURE structural OF adder_cla_16b IS
SIGNAL R_BUFFER : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL P_BUFFER : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL G_BUFFER : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL I_BUFFER : STD_LOGIC_VECTOR (16 DOWNTO 0);
COMPONENT adder_cla_1b
  PORT  (
           A        :  IN    STD_LOGIC;
           B        :  IN    STD_LOGIC;
           C_IN     :  IN    STD_LOGIC;
           R        :  OUT   STD_LOGIC;
           P        :  OUT   STD_LOGIC;
           G        :  OUT   STD_LOGIC
         );
END COMPONENT;
BEGIN
  C_OUT16           <= I_BUFFER(16);
  R16               <= R_BUFFER;
  I_BUFFER(0)       <= '0'; --Start Carry for Adder
  Full_Adder_Map       :
       for i in 0 to 15 GENERATE
        FAX :
           adder_cla_1b
                   PORT MAP
                      (
                           A       => A16(I),
                           B       => B16(I),
                           C_IN    => I_BUFFER(I),
                           R       => R_BUFFER(I),
                           P       => P_BUFFER(I),
                           G       => G_BUFFER(I)
                      );
           I_BUFFER(i+1)    <= G_BUFFER(I) OR (P_BUFFER(I) AND I_BUFFER(I));
       END GENERATE Full_Adder_Map;
END structural;
```

adder_cla_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY adder_cla_32b IS
  PORT  (
           A32      :  IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
           B32      :  IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
           R32      :  OUT   STD_LOGIC_VECTOR (31 DOWNTO 0);
           C_OUT32  :  OUT   STD_LOGIC
         );
END adder_cla_32b;
ARCHITECTURE structural OF adder_cla_32b IS
SIGNAL R_BUFFER : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL P_BUFFER : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL G_BUFFER : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL I_BUFFER : STD_LOGIC_VECTOR (32 DOWNTO 0);
COMPONENT adder_cla_1b
  PORT  (
           A        :  IN    STD_LOGIC;
           B        :  IN    STD_LOGIC;
           C_IN     :  IN    STD_LOGIC;
           R        :  OUT   STD_LOGIC;
           P        :  OUT   STD_LOGIC;
           G        :  OUT   STD_LOGIC
         );
```

```
END COMPONENT;
BEGIN
  C_OUT32           <= I_BUFFER(32);
  R32               <= R_BUFFER;
  I_BUFFER(0)       <= '0'; --Start Carry for Adder
  Full_Adder_Map      :
       for i in 0 to 31 GENERATE
        FAX :
           adder_cla_1b
                   PORT MAP
                      (
                            A       => A32(I),
                            B       => B32(I),
                            C_IN    => I_BUFFER(I),
                            R       => R_BUFFER(I),
                            P       => P_BUFFER(I),
                            G       => G_BUFFER(I)
                      );
           I_BUFFER(i+1)   <= G_BUFFER(I) OR (P_BUFFER(I) AND I_BUFFER(I));
       END GENERATE Full_Adder_Map;
END structural;
```

2. Implemenasi Substractor

Substractor dibuat menggunakan komponen yang sama dengan Adder. Hanya terdapat substractor berukuran 16-bit dalam rangkaian ini. Operan 1 tetap bernilai positif. Operan 2 akan dibalik tandanya menggunakan prinsip two's complement. Operan 2 harus masuk melalui gerbang not 16-bit sebelum masuk ke dalam adder. Kemudian, carry awal diset menjadi 1 agar merepresentasikan penambahan dengan bilangan 1. Kode VHDL diberikan sebagai berikut.

notgate_16b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY notgate_16b IS
  PORT  (
            DIN      :  IN    std_logic_vector (15 DOWNTO 0);
            DOUT     :  OUT   std_logic_vector (15 DOWNTO 0)
         );
END notgate_16b;
ARCHITECTURE structural of notgate_16b IS
SIGNAL data_buffer : std_logic_vector (15 DOWNTO 0);
BEGIN
  DOUT <= data_buffer;
  PROCESS (data_buffer, DIN)
  BEGIN
       FOR I in 0 to 15 LOOP
        data_buffer(I)<=NOT DIN(I);
       END LOOP;
  END PROCESS;
END structural;
```

substr_cla_16b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY subst_cla_16b IS
  PORT  (
            A16      :  IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
            B16      :  IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16      :  OUT   STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16  :  OUT   STD_LOGIC
         );
END subst_cla_16b;
ARCHITECTURE structural OF subst_cla_16b IS
SIGNAL R_BUFFER : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL P_BUFFER : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL G_BUFFER : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL I_BUFFER : STD_LOGIC_VECTOR (16 DOWNTO 0);
```

```
SIGNAL B16NEG   : STD_LOGIC_VECTOR (15 DOWNTO 0);
COMPONENT adder_cla_1b
  PORT  (
            A           :     IN      STD_LOGIC;
            B           :     IN      STD_LOGIC;
            C_IN        :     IN      STD_LOGIC;
            R           :     OUT     STD_LOGIC;
            P           :     OUT     STD_LOGIC;
            G           :     OUT     STD_LOGIC
         );
END COMPONENT;
COMPONENT notgate_16b IS
  PORT  (
            DIN     :  IN    std_logic_vector (15 DOWNTO 0);
            DOUT    :  OUT   std_logic_vector (15 DOWNTO 0)
         );
END COMPONENT;
BEGIN
  C_OUT16           <= I_BUFFER(16);
  R16               <= R_BUFFER;
  I_BUFFER(0)       <= '1'; --Start Carry for Substractor
  Not_Gate_Map          :
       notgate_16b
        PORT MAP
           (
                    DIN     =>      B16,
                    DOUT    =>      B16NEG
           );
  Full_Adder_Map        :
       for i in 0 to 15 GENERATE
        FAX :
           adder_cla_1b
                    PORT MAP
                       (
                            A       => A16(I),
                            B       => B16NEG(I),
                            C_IN    => I_BUFFER(I),
                            R       => R_BUFFER(I),
                            P       => P_BUFFER(I),
                            G       => G_BUFFER(I)
                       );
           I_BUFFER(i+1)    <= G_BUFFER(I) OR (P_BUFFER(I) AND I_BUFFER(I));
       END GENERATE Full_Adder_Map;
END structural;
```

3. Implementasi Complex Adder

Pada dasarnya, penjumlahan dua buah bilangan kompleks dalam bentuk kartesian dilakukan dengan menjumlahkan bagian real dengan bagian real dan bagian imaginary dengan bagian imaginary. Dengan demikian, complex adder dapat direalisasikan menggunakan dua buah Adder 16-bit sebagai berikut.

complex_adder_cla_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY complex_adder_cla_32b IS
  PORT  (
            A32      : IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            B32      : IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32      : OUT  STD_LOGIC_VECTOR (31 DOWNTO 0);
            C_OUT32  : OUT  STD_LOGIC_VECTOR (1 DOWNTO 0)
         );
END complex_adder_cla_32b;
ARCHITECTURE structural OF complex_adder_cla_32b IS
SIGNAL REAL_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_B32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```
SIGNAL IMAG_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_B32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
COMPONENT adder_cla_16b IS
  PORT  (
           A16     : IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
           B16     : IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
           R16     : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
           C_OUT16 : OUT STD_LOGIC
         );
END COMPONENT;
BEGIN
  REAL_A32            <= A32(31 DOWNTO 16);
  REAL_B32            <= B32(31 DOWNTO 16);
  IMAG_A32            <= A32(15 DOWNTO 0);
  IMAG_B32            <= B32(15 DOWNTO 0);
  R32(31 DOWNTO 16)   <= REAL_R32;
  R32(15 DOWNTO 0)    <= IMAG_R32;
  ADDER_REAL        :
       adder_cla_16b
         PORT MAP
           (
                   A16     => REAL_A32,
                   B16     => REAL_B32,
                   R16     => REAL_R32,
                   C_OUT16 => C_OUT32(1)
           );
  ADDER_IMAG        :
       adder_cla_16b
         PORT MAP
           (
                   A16     => IMAG_A32,
                   B16     => IMAG_B32,
                   R16     => IMAG_R32,
                   C_OUT16 => C_OUT32(0)
           );
END structural;
```

4. Implementasi Complex Substractor

Sama halnya dengan penjumlahan bilangan kompleks, pengurangan bilangan kompleks dalam bentuk kartesian dilakukan dengan mengurangkan bagian real dengan bagian real dan bagian imaginary dengan bagian imaginary. Dengan demikian, complex substractor dapat disusun menggunakan dua buah substractor 16-bit masing-masing mengurangkan bilangan real dengan bilangan real serta bilangan imaginary dengan bilangan imaginary. Berikut ini adalah implementasi Complex Substractor.

complex_subst_cla_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY complex_subst_cla_32b IS
  PORT   (
            A32     :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            B32     :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32     :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            C_OUT32 :  OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
          );
END complex_subst_cla_32b;
ARCHITECTURE structural OF complex_subst_cla_32b IS
SIGNAL REAL_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_B32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_B32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
COMPONENT subst_cla_16b IS
```

```
  PORT   (
            A16      :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            B16      :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16      :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16  :  OUT STD_LOGIC
         );
END COMPONENT;
BEGIN
  REAL_A32           <= A32(31 DOWNTO 16);
  REAL_B32           <= B32(31 DOWNTO 16);
  IMAG_A32           <= A32(15 DOWNTO 0);
  IMAG_B32           <= B32(15 DOWNTO 0);
  R32(31 DOWNTO 16)  <= REAL_R32;
  R32(15 DOWNTO 0)   <= IMAG_R32;
  ADDER_REAL :
        subst_cla_16b
         PORT MAP
            (
                  A16      => REAL_A32,
                  B16      => REAL_B32,
                  R16      => REAL_R32,
                  C_OUT16  => C_OUT32(1)
            );
  ADDER_IMAG :
        subst_cla_16b
         PORT MAP
            (
                  A16      => IMAG_A32,
                  B16      => IMAG_B32,
                  R16      => IMAG_R32,
                  C_OUT16  => C_OUT32(0)
            );
END structural;
```

5. Implementasi Sign Inverter

Sign Inverter digunakan untuk membalik tanda dari positif menjadi negatif ataupun dari negatif menjadi positif. Pada dasarnya, sign inverter dapat diimplementasikan dengan menggunakan substractor dengan memberikan input operan pertama adalah nol dan input operan kedua adalah bilangan yang akan dibalik tandanya. Dengan demikian, sign inverter dapat direalisasikan menggunakan substractor satu input sehingga dapat dioptimisasi lebih baik. Berikut ini adalah implementasi Sign Inverter 1-bit dan Sign Inverter 16-bit.

sgninv_1b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY sgninv_1b IS
  PORT  (
          A     :   IN    STD_LOGIC;
          C_IN  :   IN    STD_LOGIC;
          R     :   OUT   STD_LOGIC;
          P     :   OUT   STD_LOGIC
        );
END sgninv_1b;
ARCHITECTURE structural of sgninv_1b IS
BEGIN
  R     <=      A XOR C_IN;
  P     <=      A AND C_IN;
END structural;
```

sgninv_16b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY sgninv_16b IS
  PORT  (
          A16      :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
          R16      :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```
          C_OUT16 :  OUT STD_LOGIC
        );
END sgninv_16b;
ARCHITECTURE structural OF sgninv_16b IS
SIGNAL A16NEG     : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL P_BUFFER   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL R_BUFFER   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL I_BUFFER   : STD_LOGIC_VECTOR (16 DOWNTO 0);
COMPONENT sgninv_1b
  PORT  (
          A        :  IN  STD_LOGIC;
          C_IN     :  IN  STD_LOGIC;
          R        :  OUT STD_LOGIC;
          P        :  OUT STD_LOGIC
        );
END COMPONENT;
COMPONENT notgate_16b IS
  PORT  (
          DIN      :  IN  std_logic_vector (15 DOWNTO 0);
          DOUT     :  OUT std_logic_vector (15 DOWNTO 0)
        );
END COMPONENT;
BEGIN
  C_OUT16          <= I_BUFFER(16);
  R16              <= R_BUFFER;
  I_BUFFER(0)      <= '1'; --Start Carry for Substractor
  Not_Gate_Map       :
       notgate_16b
         PORT MAP
           (
                   DIN    => A16,
                   DOUT   => A16NEG
          );
  Full_Adder_Map     :
       for i in 0 to 15 GENERATE
         FAX :
           sgninv_1b
                   PORT MAP
                      (
                          A         => A16NEG(I),
                          C_IN      => I_BUFFER(I),
                          R         => R_BUFFER(I),
                          P         => P_BUFFER(I)
                      );
          I_BUFFER(i+1)  <= (P_BUFFER(I) AND I_BUFFER(I));
       END GENERATE Full_Adder_Map;
END structural;
```

6. Multiplexer

Multiplexer digunakan untuk memilih data yang akan dialirkan ke tahap selanjutnya. Dalam rangkaian ini digunakan multiplexer 2 ke 1 dengan lebar data 16-bit, multiplexer 2 ke 1 dengan lebar data 32-bit, dan multiplexer 8 ke 1 dengan lebar data 32-bit.

mux_2to1_16b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux_2to1_16b IS
  PORT  (
          D1   :  IN    std_logic_vector (15 DOWNTO 0);
          D2   :  IN    std_logic_vector (15 DOWNTO 0);
          Y    :  OUT   std_logic_vector (15 DOWNTO 0);
          S    :  IN    std_logic
        );
END mux_2to1_16b;
ARCHITECTURE behavioural of mux_2to1_16b IS
BEGIN
  WITH S SELECT
```

```
        Y <=   D1 WHEN '0',
               D2 WHEN '1';
END behavioural;
```

mux_2to1_32b.vhd

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

ENTITY mux_2to1_32b IS
   PORT   (
            D1 :      IN     std_logic_vector (31 DOWNTO 0);
            D2 :      IN     std_logic_vector (31 DOWNTO 0);
            Y  :      OUT    std_logic_vector (31 DOWNTO 0);
            S  :      IN     std_logic
          );
END mux_2to1_32b;

ARCHITECTURE behavioural of mux_2to1_32b IS
   BEGIN
       WITH S SELECT
          Y <=       D1 WHEN '0',
                     D2 WHEN '1',
                     D1 WHEN OTHERS;
   END behavioural;
```

mux_8to1_32b.vhd

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

ENTITY mux_8to1_32b IS
   PORT  (
               D1    : IN     std_logic_vector (31 DOWNTO 0);
               D2    : IN     std_logic_vector (31 DOWNTO 0);
               D3    : IN     std_logic_vector (31 DOWNTO 0);
               D4    : IN     std_logic_vector (31 DOWNTO 0);
               D5    : IN     std_logic_vector (31 DOWNTO 0);
               D6    : IN     std_logic_vector (31 DOWNTO 0);
               D7    : IN     std_logic_vector (31 DOWNTO 0);
               D8    : IN     std_logic_vector (31 DOWNTO 0);
               Y     : OUT    std_logic_vector (31 DOWNTO 0);
               S     : IN     std_logic_vector (2 DOWNTO 0)
          );
END mux_8to1_32b;

ARCHITECTURE behavioural of mux_8to1_32b IS
BEGIN
   WITH S SELECT
         Y <= D1 WHEN "000",
              D2 WHEN "001",
              D3 WHEN "010",
              D4 WHEN "011",
              D5 WHEN "100",
              D6 WHEN "101",
              D7 WHEN "110",
              D8 WHEN "111",
              D1 WHEN OTHERS;
END behavioural;
```

7. Complex Multiplier

Berbeda dengan penjumlahan dan pengurangan pada bilangan kompleks, perkalian pada bilangan kompleks dalam bentuk kartesian lebih sulit dilakukan karena setiap bagian real maupun imaginary akan dikalikan dengan bagian real maupun imaginary dari bilangan lain. Untuk menyederhanakan permasalahan tersebut, diberikan complex multiplier yang efisien berdasarkan rumus matematika sebagai berikut.

Diketahui dua bilangan kompleks yang akan dikalikan menjadi satu bilangan kompleks.

$A = X + jY$

$B = C + jS$

$H = A \cdot B = (X + jY) \cdot (P + jQ) = XP + jXQ + jY - YQ = R + jI$

Misalkan kita memiliki lima buah bilangan yang merupakan hasil operasi dari komponen bilangan A dan bilangan B sebagai berikut.

$G_1 = C$

$G_2 = C + S$

$G_3 = C - S$

$E = X - Y$

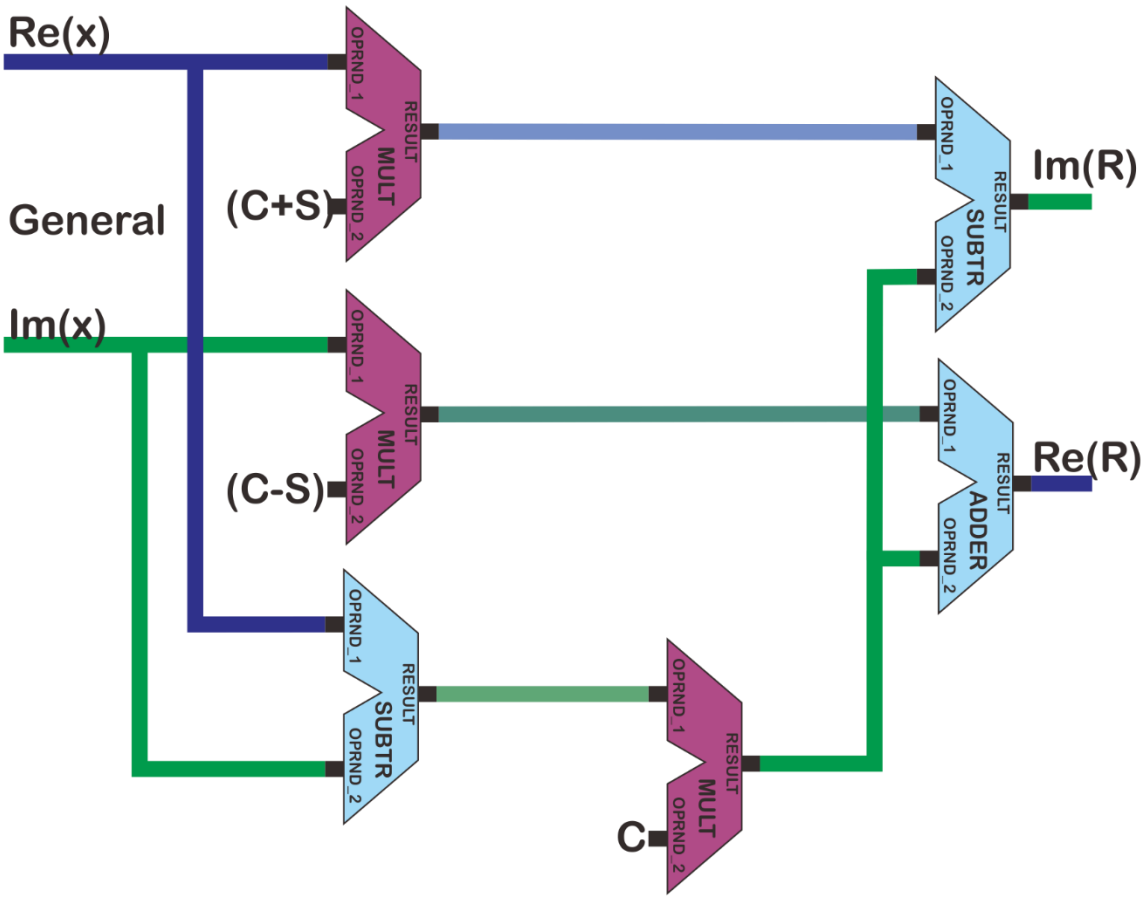
$Z = C \cdot E$

Hasil akhir diperoleh dengan sebagai berikut.

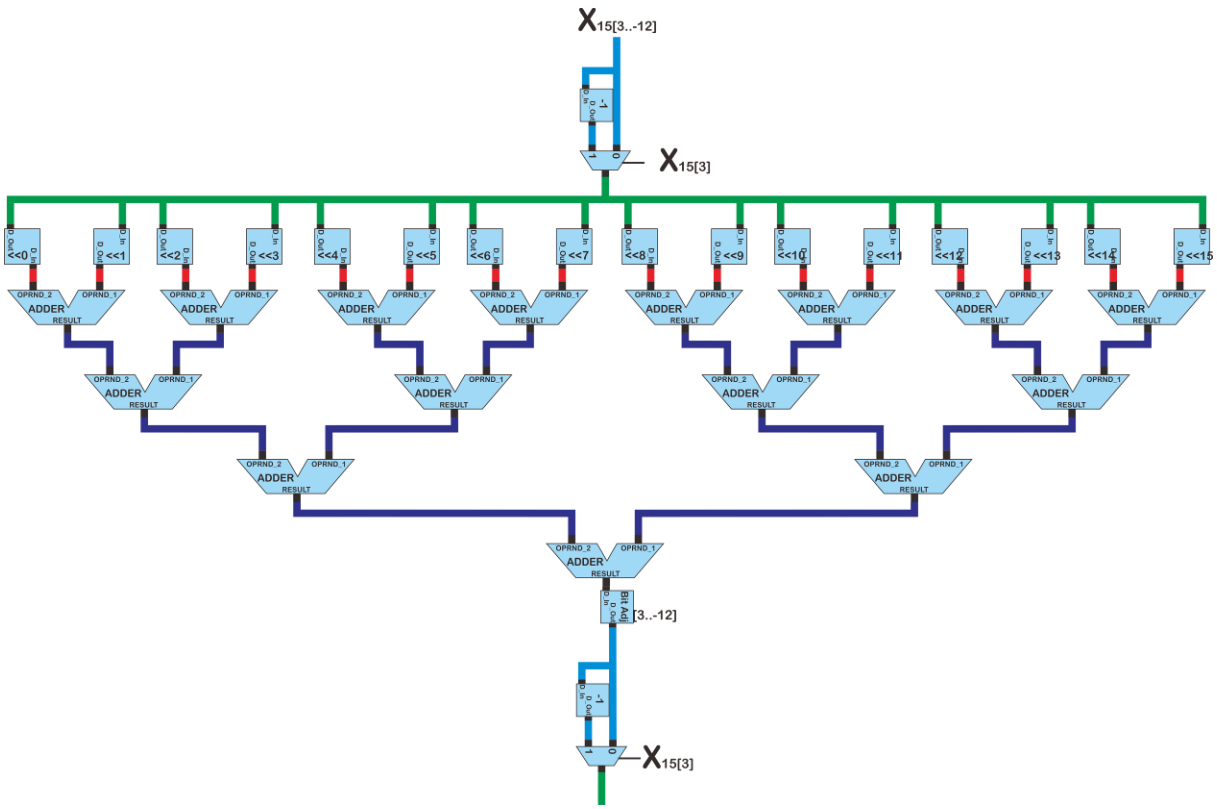
$R = G_3 \cdot Y + Z = (C - S) \cdot Y + C \cdot (X - Y)$

$I = G_2 \cdot Y - Z = (C + S) \cdot X - C \cdot (X - Y)$

Selanjutnya, kita dapat mengimplementasikan perkalian dua bilangan kompleks dengan rangkaian yang lebih sederhana sebagai berikut. Perkalian bilangan kompleks akan dijumpai pada masing-masing tahap FFT 8-titik dan pada perkalian dengan konstanta twiddle factor interdimensional. Perkalian kompleks yang akan dilakukan merupakan perkalian bilangan kompleks berupa variable dengan bilangan kompleks berupa konstanta (dalam hal ini twiddle factor). Hal ini akan mengurangi kompleksitas rangkaian dan akan mengurangi jumlah komponen yang dibutuhkan dalam realisasi complex multiplier karena complex multiplier hanya akan digunakan untuk mengalikan konstanta dengan variabel.



Secara umum, setiap blok MULT akan direalisasikan sebagai berikut. Untuk perkalian dengan bilangan konstan, blok MULT yang dihasilkan akan dapat dibuat lebih sederhana karena hanya bagian-bagian yang digunakan saja yang akan direalisasikan.



Dengan demikian, untuk mengimplementasikan blok MULT secara umum tersebut, kita memerlukan *left shifter* mulai dari *left shifter* nol kali hingga *left shifter* lima belas kali, lima belas buah adder 32-bit, dua buah sign inverter, dua buah multiplexer, dan satu bit adjustment. Perkalian dilakukan dengan mengambil nilai absolut pada input data untuk kemudian dikembalikan tandanya yang sesuai di akhir perkalian. Selain itu, bit adjustment berguna untuk melakukan pemotongan (*truncation*) terhadap hasil perkalian. Seperti yang kita ketahui, perkalian dua bilangan 16-bit akan menghasilkan bilangan 32-bit. Pemotongan dilakukan berdasarkan posisi fraksionalnya

Bit adjustment digunakan untuk memotong (*trucantion*) hasil perkalian agar diperoleh kembali bilangan 16-bit (32-bit kompleks). Selain pemotongan, bit adjustment juga melakukan rounding up atau rounding down, bergantung pada nilai bit pada bagian yang dipotong. Bit adjustment ini merupakan revisi dari bit adjustment pada struktur sebelumnya yang diimplementasikan tanpa rounding (pembulatan).

bit_adj_32b_to_16b.vhd

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

ENTITY bit_adj_32b_to_16b IS
  PORT  (
          Data_In      :     IN     std_logic_vector (31 DOWNTO 0);
          Data_Out     :     OUT    std_logic_vector (15 DOWNTO 0)
         );
END bit_adj_32b_to_16b;

ARCHITECTURE structural of bit_adj_32b_to_16b IS
COMPONENT adder_cla_16b IS
  PORT  (
          A16          :     IN   STD_LOGIC_VECTOR (15 DOWNTO 0);
          B16          :     IN   STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```
          R16          :     OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
          C_OUT16      :     OUT STD_LOGIC
         );
END COMPONENT;
SIGNAL TRUNCATE_TEMP : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL ADDER_OUT     : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL ROUNDING_TEMP : STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
       Data_Out     <= ADDER_OUT;
       TRUNCATE_TEMP<= Data_In(27 DOWNTO 12);
       ROUNDING_TEMP<= "000000000000000" & Data_In(11);
       adder :
         adder_cla_16b
          PORT MAP (
                     A16      => TRUNCATE_TEMP,
                     B16      => ROUNDING_TEMP,
                     R16      => ADDER_OUT
                   );
END structural;
```

lshift_0_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_0_16b_to_32b IS
  PORT   (
             Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
             Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_0_16b_to_32b;
ARCHITECTURE structural of lshift_0_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 16)    <= "0000000000000000";
        Data_Out(15 DOWNTO 0)     <= Data_In;
END structural;
```

lshift_1_16b_to_32b.vhd

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

ENTITY lshift_1_16b_to_32b IS
  PORT   (
             Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
             Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
             );
END lshift_1_16b_to_32b;

ARCHITECTURE structural of lshift_1_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 17)    <= "000000000000000";
        Data_Out(16 DOWNTO 1)     <= Data_In;
        Data_Out(0 DOWNTO 0)      <= "0";
END structural;
```

lshift_2_16b_to_32b.vhd

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

ENTITY lshift_2_16b_to_32b IS
  PORT   (
             Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
             Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
         );
END lshift_2_16b_to_32b;

ARCHITECTURE structural of lshift_2_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 18)<= "00000000000000";
        Data_Out(17 DOWNTO 2) <= Data_In;
        Data_Out(1 DOWNTO 0) <= "00";
```

```
END structural;
```

lshift_3_16b_to_32b.vhd

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

ENTITY lshift_3_16b_to_32b IS
  PORT   (
            Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
            Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
        );
END lshift_3_16b_to_32b;

ARCHITECTURE structural of lshift_3_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 19)    <= "0000000000000";
        Data_Out(18 DOWNTO 3)     <= Data_In;
        Data_Out(2 DOWNTO 0)      <= "000";
END structural;
```

lshift_4_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_4_16b_to_32b IS
  PORT   (
            Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
            Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
         );
END lshift_4_16b_to_32b;
ARCHITECTURE structural of lshift_4_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 20)    <= "000000000000";
        Data_Out(19 DOWNTO 4)     <= Data_In;
        Data_Out(3 DOWNTO 0)      <= "0000";
END structural;
```

lshift_5_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_5_16b_to_32b IS
  PORT   (
            Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
            Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
         );
END lshift_5_16b_to_32b;
ARCHITECTURE structural of lshift_5_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 21)    <= "00000000000";
        Data_Out(20 DOWNTO 5)     <= Data_In;
        Data_Out(4 DOWNTO 0)      <= "00000";
END structural;
```

lshift_6_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_6_16b_to_32b IS
  PORT   (
            Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
            Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_6_16b_to_32b;
ARCHITECTURE structural of lshift_6_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 22)    <= "0000000000";
        Data_Out(21 DOWNTO 6)     <= Data_In;
        Data_Out(5 DOWNTO 0)      <= "000000";
END structural;
```

lshift_7_16b_to_32b.vhd

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
ENTITY lshift_7_16b_to_32b IS
  PORT   (
              Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
              Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_7_16b_to_32b;
ARCHITECTURE structural of lshift_7_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 23)    <= "000000000";
        Data_Out(22 DOWNTO 7)     <= Data_In;
        Data_Out(6 DOWNTO 0)      <= "0000000";
END structural;
```

lshift_8_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_8_16b_to_32b IS
  PORT   (
              Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
              Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_8_16b_to_32b;
ARCHITECTURE structural of lshift_8_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 24)    <= "00000000";
        Data_Out(23 DOWNTO 8)     <= Data_In;
        Data_Out(7 DOWNTO 0)      <= "00000000";
END structural;
```

lshift_9_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_9_16b_to_32b IS
  PORT   (
              Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
              Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_9_16b_to_32b;
ARCHITECTURE structural of lshift_9_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 25)    <= "0000000";
        Data_Out(24 DOWNTO 9)     <= Data_In;
        Data_Out(8 DOWNTO 0)      <= "000000000";
END structural;
```

lshift_10_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_10_16b_to_32b IS
  PORT   (
              Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
              Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_10_16b_to_32b;
ARCHITECTURE structural of lshift_10_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 26)    <= "000000";
        Data_Out(25 DOWNTO 10)    <= Data_In;
        Data_Out(9 DOWNTO 0)      <= "0000000000";
END structural;
```

lshift_11_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_11_16b_to_32b IS
  PORT   (
              Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
              Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_11_16b_to_32b;
ARCHITECTURE structural of lshift_11_16b_to_32b IS
```

```
BEGIN
        Data_Out(31 DOWNTO 27)    <= "00000";
        Data_Out(26 DOWNTO 11)    <= Data_In;
        Data_Out(10 DOWNTO 0)     <= "00000000000";
END structural;
```

lshift_12_16b_to_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lshift_12_16b_to_32b IS
  PORT   (
            Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
            Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
         );
END lshift_12_16b_to_32b;
ARCHITECTURE structural of lshift_12_16b_to_32b IS
BEGIN
        Data_Out(31 DOWNTO 28)    <= "0000";
        Data_Out(27 DOWNTO 12)    <= Data_In;
        Data_Out(11 DOWNTO 0)     <= "000000000000";
END structural;
```

lshift_13_16b_to_32b.vhd

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

ENTITY lshift_13_16b_to_32b IS
  PORT    (
             Data_In    :   IN     std_logic_vector (15 DOWNTO 0);
             Data_Out   :   OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_13_16b_to_32b;

ARCHITECTURE structural of lshift_13_16b_to_32b IS
BEGIN
          Data_Out(31 DOWNTO 29)  <= "000";
          Data_Out(28 DOWNTO 13)  <= Data_In;
          Data_Out(12 DOWNTO 0)   <= "0000000000000";
END structural;
```

lshift_14_16b_to_32b.vhd

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

ENTITY lshift_14_16b_to_32b IS
   PORT    (
             Data_In     :  IN     std_logic_vector (15 DOWNTO 0);
             Data_Out    :  OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_14_16b_to_32b;

ARCHITECTURE structural of lshift_14_16b_to_32b IS
BEGIN
          Data_Out(31 DOWNTO 30)  <= "00";
          Data_Out(29 DOWNTO 14)  <= Data_In;
          Data_Out(13 DOWNTO 0)   <= "00000000000000";
END structural;
```

lshift_15_16b_to_32b.vhd

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

ENTITY lshift_15_16b_to_32b IS
   PORT    (
             Data_In     :  IN     std_logic_vector (15 DOWNTO 0);
             Data_Out    :  OUT    std_logic_vector (31 DOWNTO 0)
          );
END lshift_15_16b_to_32b;

ARCHITECTURE structural of lshift_15_16b_to_32b IS
BEGIN
          Data_Out(31 DOWNTO 31)  <= "0";
```

```
        Data_Out(30 DOWNTO 15)  <= Data_In;
        Data_Out(14 DOWNTO 0)   <= "000000000000000";
END structural;
```

Selanjutnya, dengan menggabungkan semua komponen tersebut, kita akan memperoleh general multiplier. Agar dapat disesuaikan dengan konstanta yang akan dikalikan, implementasi general multiplier dilakukan dengan menggunakan IF GENERATE yang diatur menggunakan variabel GENERIC sehingga hanya komponen yang dibutuhkan saja yang akan disintesis. Hal ini akan mempermudah optimisasi karena kita tidak perlu merancang constant multiplier yang berbeda-beda untuk setiap jenis konstanta.

Namun, untuk complex multiplier pada masing-masing FFT 8-titik, kita tetap menggunakan implementasi yang telah dilakukan sebelumnya, yaitu multiplier konstan square of two dan multiplier konstan half square of two.

generic_mult_16b.vhd

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

ENTITY generic_mult_16b IS
  GENERIC (
            Const     :   STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000000000000"
          );
  PORT    (
            Data_In   :   IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out  :   OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
          );
END generic_mult_16b;

ARCHITECTURE structural of generic_mult_16b IS
COMPONENT adder_cla_32b IS
  PORT    (
            A32       :   IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            B32       :   IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32       :   OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            C_OUT32   :   OUT STD_LOGIC
          );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT    (
            A16       :   IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16       :   OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16   :   OUT STD_LOGIC
          );
END COMPONENT;
COMPONENT bit_adj_32b_to_16b IS
  PORT    (
            Data_In   :   IN  std_logic_vector (31 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (15 DOWNTO 0)
          );
END COMPONENT;
COMPONENT mux_2to1_16b IS
  PORT    (
            D1        :   IN  std_logic_vector (15 DOWNTO 0);
            D2        :   IN  std_logic_vector (15 DOWNTO 0);
            Y         :   OUT std_logic_vector (15 DOWNTO 0);
            S         :   IN  std_logic
          );
END COMPONENT;
COMPONENT lshift_0_16b_to_32b IS
  PORT    (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
          );
END COMPONENT;
```

```
COMPONENT lshift_1_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_2_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_3_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_4_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_5_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_6_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_7_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_8_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_9_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_10_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_11_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_12_16b_to_32b IS
  PORT   (
```

```
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_13_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_14_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_15_16b_to_32b IS
  PORT   (
            Data_In   :   IN  std_logic_vector (15 DOWNTO 0);
            Data_Out  :   OUT std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;

SIGNAL mux_0_out      : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL mux_1_out      : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninv_0_out   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninv_1_out   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL lshift_0_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_1_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_2_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_3_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_4_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_5_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_6_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_7_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_8_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_9_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_10_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_11_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_12_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_13_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_14_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_15_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_0_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_1_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_2_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_3_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_4_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_5_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_6_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_7_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_8_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_9_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_10_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_11_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_12_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_13_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_14_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL bitadj_out     : STD_LOGIC_VECTOR (15 DOWNTO 0);

BEGIN
  Data_Out   <= mux_1_out;
  -- Port Mapping
  sgninv_0   :
        sgninv_16b
         PORT MAP (
                          A16 =>  Data_In,
                          R16 =>  sgninv_0_out
```

```
                    );
mux_0   :
       mux_2to1_16b
        PORT MAP (
                        D1  =>  Data_In,
                        D2  =>  sgninv_0_out,
                        Y   =>  mux_0_out,
                        S   =>  Data_In(15)
                    );
lshift_0_GEN : IF (Const(0)='1') GENERATE
                    BEGIN
                     lshift_0   :
                      lshift_0_16b_to_32b
                      PORT MAP (
                                  Data_In       =>     mux_0_out,
                                  Data_Out      =>     lshift_0_out
                                );
                    END GENERATE lshift_0_GEN;
lshift_1_GEN : IF (Const(1)='1') GENERATE
                    BEGIN
                     lshift_1   :
                      lshift_1_16b_to_32b
                      PORT MAP (
                                  Data_In       =>     mux_0_out,
                                  Data_Out      =>     lshift_1_out
                                );
                    END GENERATE lshift_1_GEN;
lshift_2_GEN : IF (Const(2)='1') GENERATE
                    BEGIN
                     lshift_2   :
                      lshift_2_16b_to_32b
                      PORT MAP (
                                  Data_In       =>     mux_0_out,
                                  Data_Out      =>     lshift_2_out
                                );
                    END GENERATE lshift_2_GEN;
lshift_3_GEN : IF (Const(3)='1') GENERATE
                    BEGIN
                     lshift_3   :
                      lshift_3_16b_to_32b
                      PORT MAP (
                                  Data_In       =>     mux_0_out,
                                  Data_Out      =>     lshift_3_out
                                );
                    END GENERATE lshift_3_GEN;
lshift_4_GEN : IF (Const(4)='1') GENERATE
                    BEGIN
                     lshift_4   :
                      lshift_4_16b_to_32b
                      PORT MAP (
                                  Data_In       =>     mux_0_out,
                                  Data_Out      =>     lshift_4_out
                                );
                    END GENERATE lshift_4_GEN;
lshift_5_GEN : IF (Const(5)='1') GENERATE
                    BEGIN
                     lshift_5   :
                      lshift_5_16b_to_32b
                      PORT MAP (
                                  Data_In       =>     mux_0_out,
                                  Data_Out      =>     lshift_5_out
                                );
                    END GENERATE lshift_5_GEN;
lshift_6_GEN : IF (Const(6)='1') GENERATE
                    BEGIN
                     lshift_6   :
                      lshift_6_16b_to_32b
                      PORT MAP (
```

```
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_6_out
                              );
                    END GENERATE lshift_6_GEN;
lshift_7_GEN : IF (Const(7)='1') GENERATE
                    BEGIN
                     lshift_7   :
                      lshift_7_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_7_out
                              );
                    END GENERATE lshift_7_GEN;
lshift_8_GEN : IF (Const(8)='1') GENERATE
                    BEGIN
                     lshift_8   :
                      lshift_8_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_8_out
                              );
                    END GENERATE lshift_8_GEN;
lshift_9_GEN : IF (Const(9)='1') GENERATE
                    BEGIN
                     lshift_9   :
                      lshift_9_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_9_out
                              );
                    END GENERATE lshift_9_GEN;
lshift_10_GEN : IF (Const(10)='1') GENERATE
                    BEGIN
                     lshift_10  :
                      lshift_10_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_10_out
                              );
                    END GENERATE lshift_10_GEN;
lshift_11_GEN : IF (Const(11)='1') GENERATE
                    BEGIN
                     lshift_11  :
                      lshift_11_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_11_out
                              );
                    END GENERATE lshift_11_GEN;
lshift_12_GEN : IF (Const(12)='1') GENERATE
                    BEGIN
                     lshift_12  :
                      lshift_12_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_12_out
                              );
                    END GENERATE lshift_12_GEN;
lshift_13_GEN : IF (Const(13)='1') GENERATE
                    BEGIN
                     lshift_13  :
                      lshift_13_16b_to_32b
                      PORT MAP (
                                Data_In       =>     mux_0_out,
                                Data_Out      =>     lshift_13_out
                              );
                    END GENERATE lshift_13_GEN;
lshift_14_GEN : IF (Const(14)='1') GENERATE
```

```
                    BEGIN
                     lshift_14  :
                      lshift_14_16b_to_32b
                      PORT MAP (
                                 Data_In       =>     mux_0_out,
                                 Data_Out      =>     lshift_14_out
                               );
                    END GENERATE lshift_14_GEN;
lshift_15_GEN : IF (Const(15)='1') GENERATE
                    BEGIN
                     lshift_15  :
                      lshift_15_16b_to_32b
                      PORT MAP (
                                 Data_In       =>     mux_0_out,
                                 Data_Out      =>     lshift_15_out
                               );
                    END GENERATE lshift_15_GEN;
adder_0_BOTH: IF (Const(0)='1' AND Const(1)='1') GENERATE
                    BEGIN
                     adder_0    :
                      adder_cla_32b
                      PORT MAP (
                                 A32    =>     lshift_0_out,
                                 B32    =>     lshift_1_out,
                                 R32    =>     adder_0_out
                               );
                    END GENERATE adder_0_BOTH;
adder_0_LEFT: IF (Const(0)='1' AND Const(1)='0') GENERATE
                    BEGIN
                     adder_0_out <= lshift_0_out;
                    END GENERATE adder_0_LEFT;
adder_0_RIGHT: IF (Const(0)='0' AND Const(1)='1') GENERATE
                    BEGIN
                     adder_0_out <= lshift_1_out;
                    END GENERATE adder_0_RIGHT;
adder_1_BOTH: IF (Const(2)='1' AND Const(3)='1') GENERATE
                    BEGIN
                     adder_1    :
                      adder_cla_32b
                      PORT MAP (
                                 A32    =>     lshift_2_out,
                                 B32    =>     lshift_3_out,
                                 R32    =>     adder_1_out
                               );
                    END GENERATE adder_1_BOTH;
adder_1_LEFT: IF (Const(2)='1' AND Const(3)='0') GENERATE
                    BEGIN
                     adder_1_out <= lshift_2_out;
                    END GENERATE adder_1_LEFT;
adder_1_RIGHT: IF (Const(2)='0' AND Const(3)='1') GENERATE
                    BEGIN
                     adder_1_out <= lshift_3_out;
                    END GENERATE adder_1_RIGHT;

adder_2_BOTH: IF (Const(4)='1' AND Const(5)='1') GENERATE
                    BEGIN
                     adder_2    :
                      adder_cla_32b
                      PORT MAP (
                                 A32    =>     lshift_4_out,
                                 B32    =>     lshift_5_out,
                                 R32    =>     adder_2_out
                               );
                    END GENERATE adder_2_BOTH;
adder_2_LEFT: IF (Const(4)='1' AND Const(5)='0') GENERATE
                    BEGIN
                     adder_2_out <= lshift_4_out;
                    END GENERATE adder_2_LEFT;
```

```
adder_2_RIGHT: IF (Const(4)='0' AND Const(5)='1') GENERATE
                    BEGIN
                      adder_2_out <= lshift_5_out;
                    END GENERATE adder_2_RIGHT;
adder_3_BOTH: IF (Const(6)='1' AND Const(7)='1') GENERATE
                    BEGIN
                      adder_3    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     lshift_6_out,
                                   B32    =>     lshift_7_out,
                                   R32    =>     adder_3_out
                                 );
                    END GENERATE adder_3_BOTH;
adder_3_LEFT: IF (Const(6)='1' AND Const(7)='0') GENERATE
                    BEGIN
                      adder_3_out <= lshift_6_out;
                    END GENERATE adder_3_LEFT;
adder_3_RIGHT: IF (Const(6)='0' AND Const(7)='1') GENERATE
                    BEGIN
                      adder_3_out <= lshift_7_out;
                    END GENERATE adder_3_RIGHT;
adder_4_BOTH: IF (Const(8)='1' AND Const(9)='1') GENERATE
                    BEGIN
                      adder_4    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     lshift_8_out,
                                   B32    =>     lshift_9_out,
                                   R32    =>     adder_4_out
                                 );
                    END GENERATE adder_4_BOTH;
adder_4_LEFT: IF (Const(8)='1' AND Const(9)='0') GENERATE
                    BEGIN
                      adder_4_out <= lshift_8_out;
                    END GENERATE adder_4_LEFT;
adder_4_RIGHT: IF (Const(8)='0' AND Const(9)='1') GENERATE
                    BEGIN
                      adder_4_out <= lshift_9_out;
                    END GENERATE adder_4_RIGHT;
adder_5_BOTH: IF (Const(10)='1' AND Const(11)='1') GENERATE
                    BEGIN
                      adder_5    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     lshift_10_out,
                                   B32    =>     lshift_11_out,
                                   R32    =>     adder_5_out
                                 );
                    END GENERATE adder_5_BOTH;
adder_5_LEFT: IF (Const(10)='1' AND Const(11)='0') GENERATE
                    BEGIN
                      adder_5_out <= lshift_10_out;
                    END GENERATE adder_5_LEFT;
adder_5_RIGHT: IF (Const(10)='0' AND Const(11)='1') GENERATE
                    BEGIN
                      adder_5_out <= lshift_11_out;
                    END GENERATE adder_5_RIGHT;
adder_6_BOTH: IF (Const(12)='1' AND Const(13)='1') GENERATE
                    BEGIN
                      adder_6    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     lshift_12_out,
                                   B32    =>     lshift_13_out,
                                   R32    =>     adder_6_out
                                 );
                    END GENERATE adder_6_BOTH;
```

```
adder_6_LEFT: IF (Const(12)='1' AND Const(13)='0') GENERATE
                    BEGIN
                      adder_6_out <= lshift_12_out;
                    END GENERATE adder_6_LEFT;
adder_6_RIGHT: IF (Const(12)='0' AND Const(13)='1') GENERATE
                    BEGIN
                      adder_6_out <= lshift_13_out;
                    END GENERATE adder_6_RIGHT;
adder_7_BOTH: IF (Const(14)='1' AND Const(15)='1') GENERATE
                    BEGIN
                      adder_7    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     lshift_14_out,
                                   B32    =>     lshift_15_out,
                                   R32    =>     adder_7_out
                                 );
                    END GENERATE adder_7_BOTH;
adder_7_LEFT: IF (Const(14)='1' AND Const(15)='0') GENERATE
                    BEGIN
                      adder_7_out <= lshift_14_out;
                    END GENERATE adder_7_LEFT;
adder_7_RIGHT: IF (Const(14)='0' AND Const(15)='1') GENERATE
                    BEGIN
                      adder_7_out <= lshift_15_out;
                    END GENERATE adder_7_RIGHT;
adder_8_BOTH: IF ((Const(0)='1' or Const(1)='1') and (Const(2)='1' or
                    Const(3)='1')) GENERATE
                    BEGIN
                      adder_8    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     adder_0_out,
                                   B32    =>     adder_1_out,
                                   R32    =>     adder_8_out
                                 );
                    END GENERATE adder_8_BOTH;
adder_8_LEFT: IF ((Const(0)='1' or Const(1)='1') and (Const(2)='0' and
                    Const(3)='0')) GENERATE
                    BEGIN
                      adder_8_out <= adder_0_out;
                    END GENERATE adder_8_LEFT;
adder_8_RIGHT: IF ((Const(0)='0' and Const(1)='0') and (Const(2)='1' or
                    Const(3)='1')) GENERATE
                    BEGIN
                      adder_8_out <= adder_1_out;
                    END GENERATE adder_8_RIGHT;
adder_9_BOTH: IF ((Const(4)='1' or Const(5)='1') and (Const(6)='1' or
                    Const(7)='1')) GENERATE
                    BEGIN
                      adder_9    :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     adder_2_out,
                                   B32    =>     adder_3_out,
                                   R32    =>     adder_9_out
                                 );
                    END GENERATE adder_9_BOTH;
adder_9_LEFT: IF ((Const(4)='1' or Const(5)='1') and (Const(6)='0' and
                    Const(7)='0')) GENERATE
                    BEGIN
                      adder_9_out <= adder_2_out;
                    END GENERATE adder_9_LEFT;
adder_9_RIGHT: IF ((Const(4)='0' and Const(5)='0') and (Const(6)='1' or
                    Const(7)='1')) GENERATE
                    BEGIN
                      adder_9_out <= adder_3_out;
                    END GENERATE adder_9_RIGHT;
```

```
adder_10_BOTH: IF ((Const(8)='1' or Const(9)='1') and (Const(10)='1' or
                   Const(11)='1')) GENERATE
                   BEGIN
                    adder_10    :
                     adder_cla_32b
                     PORT MAP (
                               A32    =>     adder_4_out,
                               B32    =>     adder_5_out,
                               R32    =>     adder_10_out
                              );
                   END GENERATE adder_10_BOTH;
adder_10_LEFT: IF ((Const(8)='1' or Const(9)='1') and (Const(10)='0' and
                   Const(11)='0')) GENERATE
                   BEGIN
                    adder_10_out<= adder_4_out;
                   END GENERATE adder_10_LEFT;
adder_10_RIGHT: IF ((Const(8)='0' and Const(9)='0') and (Const(10)='1' or
                   Const(11)='1')) GENERATE
                   BEGIN
                    adder_10_out<= adder_5_out;
                   END GENERATE adder_10_RIGHT;
adder_11_BOTH: IF ((Const(12)='1' or Const(13)='1') and (Const(14)='1' or
                   Const(15)='1')) GENERATE
                   BEGIN
                    adder_11    :
                     adder_cla_32b
                     PORT MAP (
                               A32    =>     adder_6_out,
                               B32    =>     adder_7_out,
                               R32    =>     adder_11_out
                              );
                   END GENERATE adder_11_BOTH;
adder_11_LEFT: IF ((Const(12)='1' or Const(13)='1') and (Const(14)='0' and
                   Const(15)='0')) GENERATE
                   BEGIN
                    adder_11_out<= adder_6_out;
                   END GENERATE adder_11_LEFT;
adder_11_RIGHT: IF ((Const(12)='0' and Const(13)='0') and (Const(14)='1' or
                   Const(15)='1')) GENERATE
                   BEGIN
                    adder_11_out<= adder_7_out;
                   END GENERATE adder_11_RIGHT;

adder_12_BOTH: IF ( (Const(0)='1' or Const(1)='1' or Const(2)='1' or
                   Const(3)='1') and (Const(4)='1' or Const(5)='1' or
                   Const(6)='1' or Const(7)='1') ) GENERATE
                   BEGIN
                    adder_12    :
                     adder_cla_32b
                     PORT MAP (
                               A32    =>     adder_8_out,
                               B32    =>     adder_9_out,
                               R32    =>     adder_12_out
                              );
                   END GENERATE adder_12_BOTH;
adder_12_LEFT: IF ( (Const(0)='1' or Const(1)='1' or Const(2)='1' or
                   Const(3)='1') and (Const(4)='0' and Const(5)='0' and
                   Const(6)='0' and Const(7)='0') ) GENERATE
                   BEGIN
                    adder_12_out<= adder_8_out;
                   END GENERATE adder_12_LEFT;
adder_12_RIGHT: IF ( (Const(0)='0' and Const(1)='0' and Const(2)='0' and
                   Const(3)='0') and (Const(4)='1' or Const(5)='1' or
                   Const(6)='1' or Const(7)='1') ) GENERATE
                   BEGIN
                    adder_12_out<= adder_9_out;
                   END GENERATE adder_12_RIGHT;
```

```
adder_13_BOTH: IF ( (Const(8)='1' or Const(9)='1' or Const(10)='1' or
                    Const(11)='1') and (Const(12)='1' or Const(13)='1' or
                    Const(14)='1' or Const(15)='1') ) GENERATE
                    BEGIN
                      adder_13     :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     adder_10_out,
                                   B32    =>     adder_11_out,
                                   R32    =>     adder_13_out
                                 );
                    END GENERATE adder_13_BOTH;
adder_13_LEFT: IF ( (Const(8)='1' or Const(9)='1' or Const(10)='1' or
                    Const(11)='1') and (Const(12)='0' and Const(13)='0' and
                    Const(14)='0' and Const(15)='0') ) GENERATE
                    BEGIN
                      adder_13_out<= adder_10_out;
                    END GENERATE adder_13_LEFT;
adder_13_RIGHT: IF ( (Const(8)='0' and Const(9)='0' and Const(10)='0' and
                    Const(11)='0') and (Const(12)='1' or Const(13)='1' or
                    Const(14)='1' or Const(15)='1') ) GENERATE
                    BEGIN
                      adder_13_out<= adder_11_out;
                    END GENERATE adder_13_RIGHT;
adder_14_BOTH: IF ( (Const(0)='1' or Const(1)='1' or Const(2)='1' or
                    Const(3)='1' or Const(4)='1' or Const(5)='1' or
                    Const(6)='1' or Const(7)='1') and (Const(8)='1' or
                    Const(9)='1' or Const(10)='1' or Const(11)='1' or
                    Const(12)='1' or Const(13)='1' or Const(14)='1' or
                    Const(15)='1')) GENERATE
                    BEGIN
                      adder_14     :
                        adder_cla_32b
                        PORT MAP (
                                   A32    =>     adder_12_out,
                                   B32    =>     adder_13_out,
                                   R32    =>     adder_14_out
                                 );
                    END GENERATE adder_14_BOTH;
adder_14_LEFT: IF ( (Const(0)='1' or Const(1)='1' or Const(2)='1' or
                    Const(3)='1' or Const(4)='1' or Const(5)='1' or
                    Const(6)='1' or Const(7)='1') and (Const(8)='0' and
                    Const(9)='0' and Const(10)='0' and Const(11)='0' and
                    Const(12)='0' and Const(13)='0' and Const(14)='0' and
                    Const(15)='0')) GENERATE
                    BEGIN
                      adder_14_out<= adder_12_out;
                    END GENERATE adder_14_LEFT;
adder_14_RIGHT: IF ((Const(0)='0' and Const(1)='0' and Const(2)='0' and
                    Const(3)='0' and Const(4)='0' and Const(5)='0' and
                    Const(6)='0' and Const(7)='0') and (Const(8)='1' or
                    Const(9)='1' or Const(10)='1' or Const(11)='1' or
                    Const(12)='1' or Const(13)='1' or Const(14)='1' or
                    Const(15)='1')) GENERATE
                    BEGIN
                      adder_14_out<= adder_13_out;
                    END GENERATE adder_14_RIGHT;

adder_NONE:         IF ((Const(0)='0' and Const(1)='0' and Const(2)='0' and
                    Const(3)='0' and Const(4)='0' and Const(5)='0' and
                    Const(6)='0' and Const(7)='0') and (Const(8)='0' and
                    Const(9)='0' and Const(10)='0' and Const(11)='0' and
                    Const(12)='0' and Const(13)='0' and Const(14)='0' and
                    Const(15)='0')) GENERATE
                    BEGIN
                      adder_14_out<= "00000000000000000000000000000000";
                    END GENERATE adder_NONE;
bitadj   :
```

```
        bit_adj_32b_to_16b
         PORT MAP (
                    Data_In    =>      adder_14_out,
                    Data_Out   =>      bitadj_out
                  );
  sgninv_1   :
        sgninv_16b
         PORT MAP (
                    A16     => bitadj_out,
                    R16     => sgninv_1_out
                  );
  mux_1    :
        mux_2to1_16b
         PORT MAP (
                    D1      => bitadj_out,
                    D2      => sgninv_1_out,
                    Y       => mux_1_out,
                    S       => Data_In(15)
                  );
END structural;
```

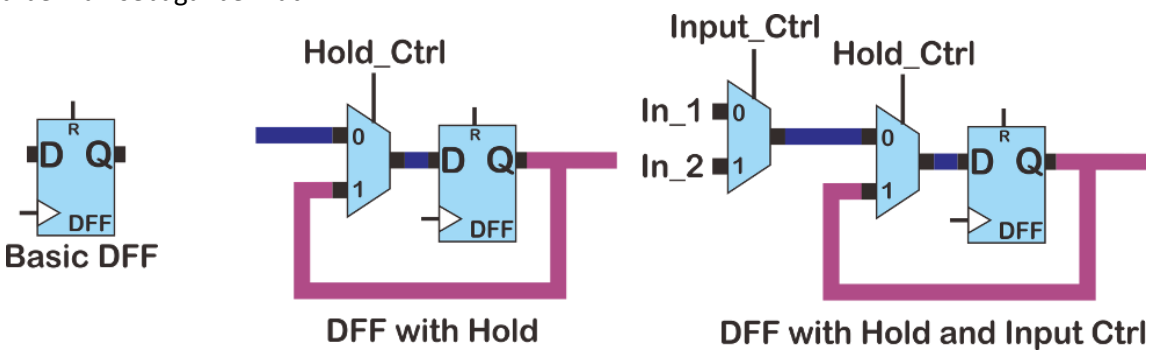
generic_complex_mult_16b.vhd

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

ENTITY generic_complex_mult_16b IS
   GENERIC (
              C_PLUS_S: STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
              C_ONLY  : STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
              C_MIN_S : STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000"

            );
   PORT     (
              REAL_A32:     IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
              IMAG_A32:     IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
              REAL_R32:     OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
              IMAG_R32:     OUT    STD_LOGIC_VECTOR (15 DOWNTO 0)
            );
END generic_complex_mult_16b;

ARCHITECTURE structural OF generic_complex_mult_16b IS
COMPONENT generic_mult_16b IS
   GENERIC (
              Const  :      STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000000000000"
            );
   PORT     (
              Data_In :     IN     STD_LOGIC_VECTOR(15 DOWNTO 0);
              Data_Out:     OUT    STD_LOGIC_VECTOR(15 DOWNTO 0)
            );
END COMPONENT;
COMPONENT adder_cla_16b IS
   PORT     (
              A16           :      IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
              B16           :      IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
              R16           :      OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
              C_OUT16       :      OUT    STD_LOGIC
            );
END COMPONENT;
COMPONENT subst_cla_16b IS
   PORT     (
              A16           :      IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
              B16           :      IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
              R16           :      OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
              C_OUT16       :      OUT    STD_LOGIC
            );
END COMPONENT;

SIGNAL MULT_C_Plus_S_Out    : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MULT_C_Min_S_Out     : STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```
SIGNAL MULT_C_Out          : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SUBSTR_0_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL ADDER_1_OUT         : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SUBSTR_1_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);

BEGIN
   REAL_R32          <= ADDER_1_OUT;
   IMAG_R32          <= SUBSTR_1_OUT;

   -- Port Mapping
   MULT_C_PLUS_S :
           generic_mult_16b
            GENERIC MAP
              (
                     Const         => C_PLUS_S
              )
            PORT MAP
              (
                     Data_In       => REAL_A32,
                     Data_Out      => MULT_C_Plus_S_Out
              );
   MULT_C_MIN_S :
           generic_mult_16b
            GENERIC MAP
              (
                     Const         => C_MIN_S
              )
            PORT MAP
              (
                     Data_In       => IMAG_A32,
                     Data_Out      => MULT_C_Min_S_Out
              );
   SUBSTR_0 :
           subst_cla_16b
            PORT MAP
              (
                     A16           => REAL_A32,
                     B16           => IMAG_A32,
                     R16           => SUBSTR_0_OUT
              );
   MULT_C_ONLY :
           generic_mult_16b
            GENERIC MAP
              (
                     Const         => C_ONLY
              )
            PORT MAP
              (
                     Data_In       => SUBSTR_0_OUT,
                     Data_Out      => MULT_C_Out
              );
   SUBSTR_1 :
           subst_cla_16b
            PORT MAP
              (
                     A16    => MULT_C_Plus_S_Out,
                     B16    => MULT_C_Out,
                     R16    => SUBSTR_1_OUT
              );
   ADDER_1 :
           adder_cla_16b
            PORT MAP
              (
                     A16    => MULT_C_Min_S_Out,
                     B16    => MULT_C_Out,
                     R16    => ADDER_1_OUT
              );
END structural;
```

8. Register

Untuk mengimplementasikan elemen penyimpan, kita memerlukan komponen register. Setidaknya, terdapat tiga jenis register yang akan kita gunakan. Ketiga jenis register tersebut diberikan sebagai berikut.



Basic DFF merupakan D Flip-flop yang paling dasar. Basic DFF memiliki positive edge triggered dan reset yang sinkron. DFF with Hold memberikan D Flip-flop kemampuan untuk mempertahankan data yang disimpannya walaupun nilai input memiliki kemungkinan berubah. DFF with Hold akan digunakan secara intensif dalam Input Unit, CB Unit, dan Output Unit. DFF with Hold and Input Ctrl merupakan ekstensi dari DFF with Hold sehingga DFF memiliki dua buah pilihan input. DFF with Hold and Input Ctrl ini juga akan digunakan secara intensif dalam Input Unit, CB Unit, dan Output Unit.

basic_dff_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY basic_dff_32b IS
    PORT   (
                D     :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
                clk   :      IN     STD_LOGIC;
                rst   :      IN     STD_LOGIC;
                Q     :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
               );
END basic_dff_32b;

ARCHITECTURE behavioral of basic_dff_32b IS
SIGNAL Q_temp : STD_LOGIC_VECTOR (31 DOWNTO 0);
    BEGIN
    Q      <= Q_temp;
    PROCESS(D, clk, rst)
    BEGIN
           IF ((clk'EVENT) AND (clk='1')) THEN
              IF (rst='1') THEN
                Q_temp <= std_logic_vector(to_unsigned(0,32));
              ELSE
                Q_temp <= D;
              END IF;
           END IF;
    END PROCESS;
    END behavioral;
```

dff_with_hold_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dff_with_hold_32b IS
```

```
    PORT    (
                D         :  IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
                clk       :  IN    STD_LOGIC;
                hold      :  IN    STD_LOGIC;
                rst       :  IN    STD_LOGIC;
                Q         :  OUT   STD_LOGIC_VECTOR (31 DOWNTO 0)
              );
END dff_with_hold_32b;

ARCHITECTURE structural of dff_with_hold_32b IS
COMPONENT basic_dff_32b IS
    PORT    (
                D         :  IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
                clk       :  IN    STD_LOGIC;
                rst       :  IN    STD_LOGIC;
                Q         :  OUT   STD_LOGIC_VECTOR (31 DOWNTO 0)
              );
END COMPONENT;
COMPONENT mux_2to1_32b IS
    PORT    (
                D1     :  IN      std_logic_vector (31 DOWNTO 0);
                D2     :  IN      std_logic_vector (31 DOWNTO 0);
                Y      :  OUT     std_logic_vector (31 DOWNTO 0);
                S      :  IN      std_logic
              );
END COMPONENT;
SIGNAL DFF_0_OUT:      STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL MUX_0_OUT:      STD_LOGIC_VECTOR (31 DOWNTO 0);
    BEGIN
    Q       <= DFF_0_OUT;
    -- Port Map
    MUX_0 :
            mux_2to1_32b
               PORT MAP (
                          D1 =>D,
                          D2 =>DFF_0_OUT,
                          Y  =>MUX_0_OUT,
                          S  =>hold
                        );
    DFF_0 :
            basic_dff_32b
               PORT MAP (
                          D  =>MUX_0_OUT,
                          clk =>clk,
                          rst =>rst,
                          Q   =>DFF_0_OUT
                        );
    END structural;
```

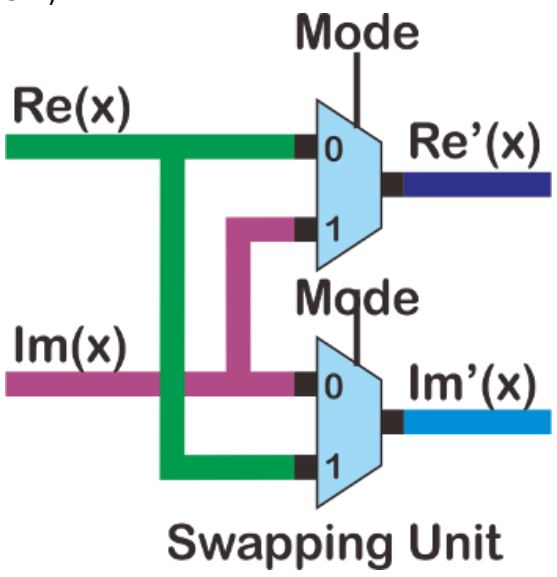
dff_with_hold_input_ctrl.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dff_with_hold_input_ctrl IS
    PORT  (
              D1        :   IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
              D2        :   IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk       :   IN     STD_LOGIC;
              hold      :   IN     STD_LOGIC;
              in_sel    :   IN     STD_LOGIC;
              rst       :   IN     STD_LOGIC;
              Q         :   OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
           );
END dff_with_hold_input_ctrl;

ARCHITECTURE structural of dff_with_hold_input_ctrl IS
COMPONENT dff_with_hold_32b IS
    PORT  (
```

```
            D         :   IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk       :   IN     STD_LOGIC;
            hold      :   IN     STD_LOGIC;
            rst       :   IN     STD_LOGIC;
            Q         :   OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT mux_2to1_32b IS
    PORT (
            D1      :IN  std_logic_vector (31 DOWNTO 0);
            D2      :IN  std_logic_vector (31 DOWNTO 0);
            Y       :OUT std_logic_vector (31 DOWNTO 0);
            S       :IN  std_logic
        );
END COMPONENT;
SIGNAL DFF_1_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL MUX_1_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
    BEGIN
    Q    <=  DFF_1_OUT;
    -- Port Map
    MUX_1 :
         mux_2to1_32b
          PORT MAP (
                    D1  =>D1,
                    D2  =>D2,
                    Y   =>MUX_1_OUT,
                    S   =>in_sel
                   );
    DFF_1 :
         dff_with_hold_32b
          PORT MAP (
                    D   =>MUX_1_OUT,
                    clk =>clk,
                    hold=>hold,
                    rst =>rst,
                    Q   =>DFF_1_OUT
                  );
    END structural;
```

9. Real-Imaginary Swapping / Interchange

Blok ini digunakan untuk menukar posisi bilangan real dan posisi bilangan imaginary. Blok ini akan digunakan pada bagian awal dari input buffer dan pada bagian akhir dari output buffer dan digunakan untuk mengatur operasi yang dilakukan oleh prosesor. Prosesor dapat melakukan FFT (mode=0) atau IFFT (mode=1).

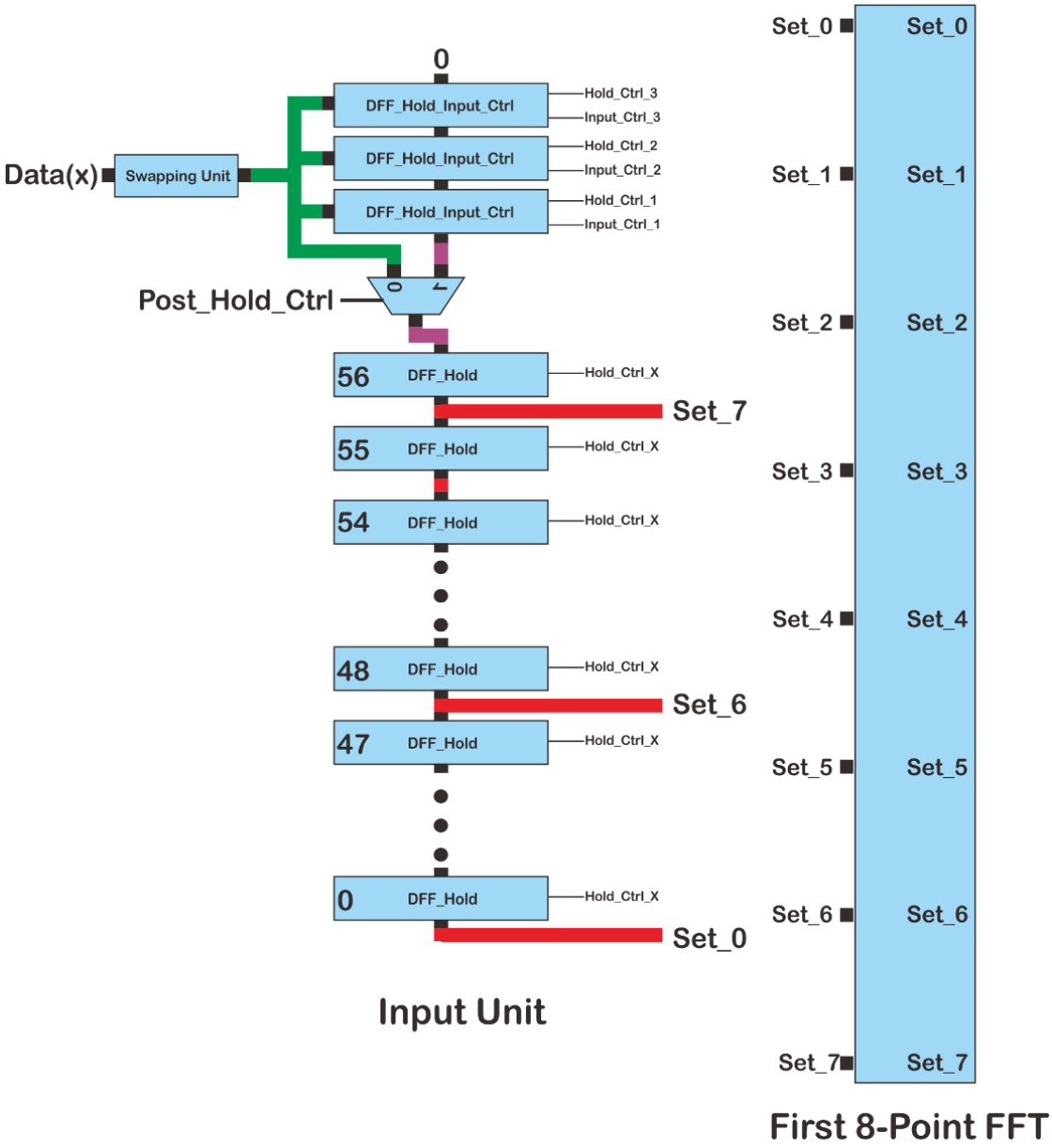


real_imaginary_interchange.vhd

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

ENTITY real_imaginary_interchange IS
    PORT (
              A32    :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
              Swap   :      IN     STD_LOGIC;
              R32    :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END real_imaginary_interchange;

ARCHITECTURE structural OF real_imaginary_interchange IS
COMPONENT mux_2to1_16b IS
    PORT (
              D1     :      IN     std_logic_vector (15 DOWNTO 0);
              D2     :      IN     std_logic_vector (15 DOWNTO 0);
              Y      :      OUT    std_logic_vector (15 DOWNTO 0);
              S      :      IN     std_logic
          );
END COMPONENT;

SIGNAL REAL_A32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_0_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_1_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);

BEGIN
    REAL_A32                <= A32(31 DOWNTO 16);
    IMAG_A32                <= A32(15 DOWNTO 0);
    REAL_R32                <= MUX_0_OUT;
    IMAG_R32                <= MUX_1_OUT;
    R32(31 DOWNTO 16)       <= REAL_R32;
    R32(15 DOWNTO 0)        <= IMAG_R32;


    -- Port Mapping
    MUX_0 :
         mux_2to1_16b
          PORT MAP
             (
             D1     =>REAL_A32,
             D2     =>IMAG_A32,
             Y      =>MUX_0_OUT,
             S      =>Swap
             );
    MUX_1 :
         mux_2to1_16b
          PORT MAP
             (
             D1     =>IMAG_A32,
             D2     =>REAL_A32,
             Y      =>MUX_1_OUT,
             S      =>Swap
             );
END structural;
```

C. Blok Input Buffer (I/P Unit)

Input buffer memiliki tiga fungsi utama. Fungsi pertama adalah melakukan buffering data input. Fungsi kedua adalah melakukan *alignment* data yang akan diproses. Fungsi ketiga adalah membuat serial data input menjadi paralel data input. Berikut ini adalah struktur Input Buffer.



Input Buffer memiliki 56 buah DFF_Hold register yang terbagi ke dalam tujuh buah segmen. Segmen paling bawah adalah segmen ke-0 (0..7) dan segmen paling atas adalah segmen ke-6 (48..55). Selain itu, diakhir dari segmen ke-7 terdapat satu buah DFF_Hold register (56). Terdapat tiga buah DFF_Hold_Input_Ctrl register yang digunakan untuk melakukan buffering input data ketika terjadi delay operasi pada bagian interdimensional multiplier (akan dijelaskan kemudian). Multiplexer digunakan untuk memilih jalur input data. Pada bagian awal rangkaian terdapat rangkaian swapping yang berfungsi untuk menukar posisi bilangan real dan bilangan imaginer. Hal ini dilakukan sehingga FFT dan IFFT dapat dilakukan dalam struktur prosesor yang sama. Untuk melakukan FFT, mode diberikan sinyal LOW sedangkan untuk melakukan IFFT, mode diberikan sinyal HIGH. Sinyal kontrol akan diatur oleh Master Control.

dff_segment_for_input.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dff_segment_for_input IS
    PORT   (
```

```
            D       :       IN          STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk     :       IN      STD_LOGIC;
            hold    :       IN      STD_LOGIC;
            rst     :       IN      STD_LOGIC;
            Q       :       OUT     STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END dff_segment_for_input;

ARCHITECTURE structural of dff_segment_for_input IS
COMPONENT dff_with_hold_32b IS
    PORT   (
            D       :       IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk     :       IN      STD_LOGIC;
            hold    :       IN      STD_LOGIC;
            rst     :       IN      STD_LOGIC;
            Q       :       OUT     STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END COMPONENT;
SIGNAL DFF_0_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_1_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_2_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_3_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_4_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_5_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_6_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_7_OUT:   STD_LOGIC_VECTOR (31 DOWNTO 0);
    BEGIN
    Q       <= DFF_0_OUT;
    -- Port Map
    DFF_7 :
           dff_with_hold_32b
            PORT MAP (
                            D       =>D,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_7_OUT
                    );

    DFF_6 :
           dff_with_hold_32b
            PORT MAP (
                            D       =>DFF_7_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_6_OUT
                    );
    DFF_5 :
           dff_with_hold_32b
            PORT MAP (
                            D       =>DFF_6_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_5_OUT
                    );
    DFF_4 :
           dff_with_hold_32b
            PORT MAP (
                            D       =>DFF_5_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_4_OUT
                    );
    DFF_3 :
           dff_with_hold_32b
```

```
            PORT MAP (
                            D       =>DFF_4_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_3_OUT
                    );
    DFF_2 :
            dff_with_hold_32b
            PORT MAP (
                            D       =>DFF_3_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_2_OUT
                    );
    DFF_1 :
            dff_with_hold_32b
            PORT MAP (
                            D       =>DFF_2_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_1_OUT
                    );
    DFF_0 :
            dff_with_hold_32b
            PORT MAP (
                            D       =>DFF_1_OUT,
                            clk     =>clk,
                            hold    =>hold,
                            rst     =>rst,
                            Q       =>DFF_0_OUT
                    );
    END structural;
```

Input_circuit.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY input_circuit IS
    PORT   (
                D               :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                clk             :  IN  STD_LOGIC;
                hold_all_seg    :  IN  STD_LOGIC;
                hold_buf_1      :  IN  STD_LOGIC;
                hold_buf_2      :  IN  STD_LOGIC;
                hold_buf_3      :  IN  STD_LOGIC;
                in_ctrl_buf_1   :  IN  STD_LOGIC;
                in_ctrl_buf_2   :  IN  STD_LOGIC;
                in_ctrl_buf_3   :  IN  STD_LOGIC;
                pos_hold_ctrl   :  IN  STD_LOGIC;
                mode            :  IN  STD_LOGIC;
                rst             :  IN  STD_LOGIC;
                Q0              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q1              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q2              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q3              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q4              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q5              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q6              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q7              :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
              );
END input_circuit;

ARCHITECTURE structural of input_circuit IS
COMPONENT dff_segment_for_input IS
    PORT   (
```

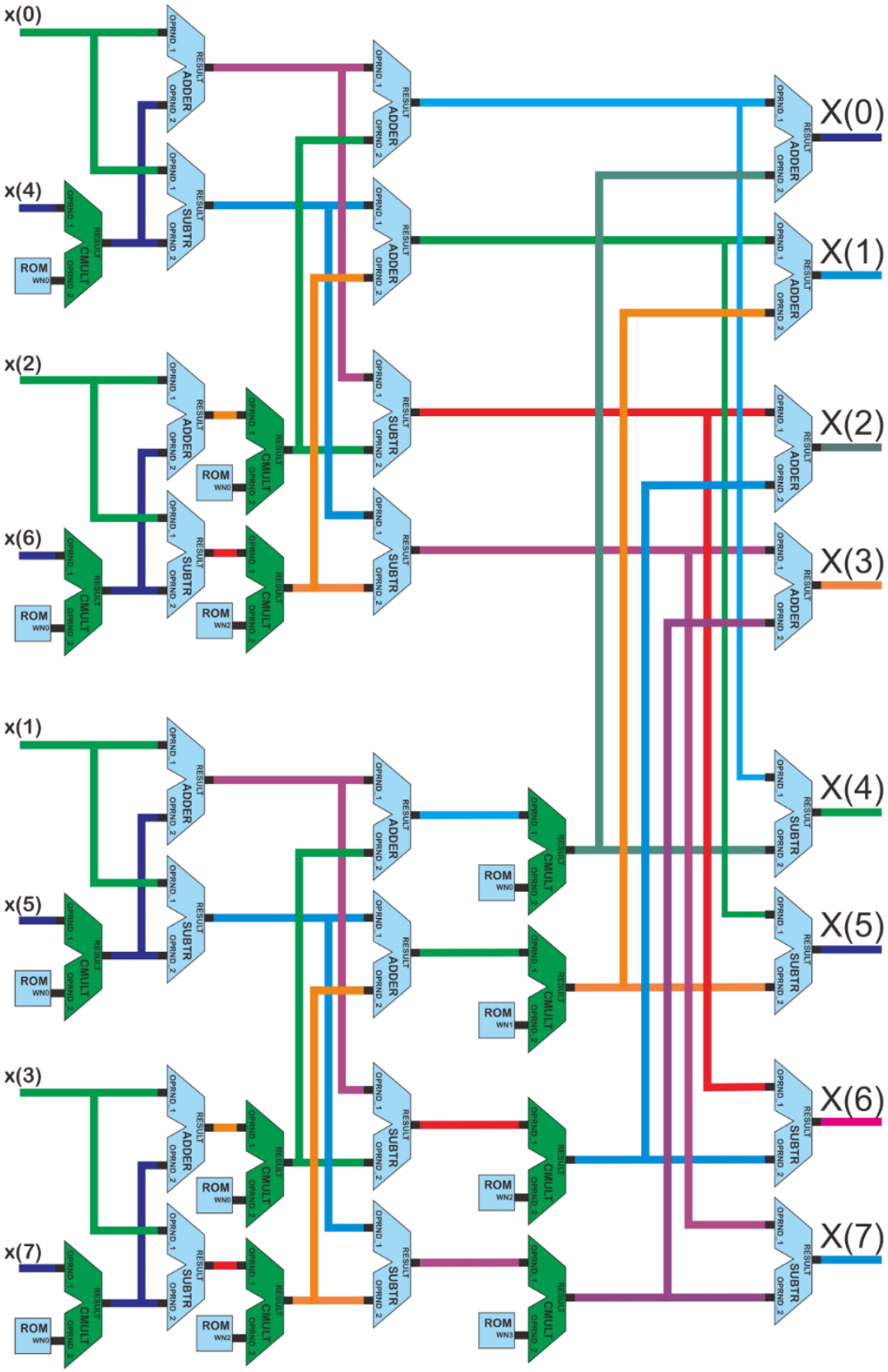
```
            D               :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk             :  IN   STD_LOGIC;
            hold            :  IN   STD_LOGIC;
            rst             :  IN   STD_LOGIC;
            Q               :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
COMPONENT dff_with_hold_32b IS
    PORT   (
            D               :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk             :  IN   STD_LOGIC;
            hold            :  IN   STD_LOGIC;
            rst             :  IN   STD_LOGIC;
            Q               :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
COMPONENT dff_with_hold_input_ctrl IS
    PORT   (
            D1              :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            D2              :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk             :  IN   STD_LOGIC;
            hold            :  IN   STD_LOGIC;
            in_sel          :  IN   STD_LOGIC;
            rst             :  IN   STD_LOGIC;
            Q               :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
COMPONENT mux_2to1_32b IS
    PORT   (
            D1              :  IN   std_logic_vector (31 DOWNTO 0);
            D2              :  IN   std_logic_vector (31 DOWNTO 0);
            Y               :  OUT  std_logic_vector (31 DOWNTO 0);
            S               :  IN   std_logic
          );
END COMPONENT;
COMPONENT real_imaginary_interchange IS
    PORT   (
            A32             :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            Swap            :  IN   STD_LOGIC;
            R32             :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;

SIGNAL SWAP_OUT         :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL BUF_1_OUT        :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL BUF_2_OUT        :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL BUF_3_OUT        :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL MUX_OUT          :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL LEAD_BUF_OUT     :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_0_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_1_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_2_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_3_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_4_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_5_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_6_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_7_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
    BEGIN
    Q0                  <=      SEGMENT_0_OUT;
    Q1                  <=      SEGMENT_1_OUT;
    Q2                  <=      SEGMENT_2_OUT;
    Q3                  <=      SEGMENT_3_OUT;
    Q4                  <=      SEGMENT_4_OUT;
    Q5                  <=      SEGMENT_5_OUT;
    Q6                  <=      SEGMENT_6_OUT;
    Q7                  <=      SEGMENT_7_OUT;
    SEGMENT_7_OUT     <=    LEAD_BUF_OUT;
    -- Port Map
```

```
 SWAP :
       real_imaginary_interchange
          PORT MAP (
                              A32 =>D,
                              Swap=>mode,
                              R32 =>SWAP_OUT
                            );
 BUF_3 :
       dff_with_hold_input_ctrl
          PORT MAP (
                              D1     =>std_logic_vector(to_unsigned(0,32)),
                              D2     =>SWAP_OUT,
                              clk    =>clk,
                              hold   =>hold_buf_3,
                              in_sel =>in_ctrl_buf_3,
                              rst    =>rst,
                              Q      =>BUF_3_OUT
                            );
 BUF_2 :
       dff_with_hold_input_ctrl
          PORT MAP (
                              D1     =>BUF_3_OUT,
                              D2     =>SWAP_OUT,
                              clk    =>clk,
                              hold   =>hold_buf_2,
                              in_sel =>in_ctrl_buf_2,
                              rst    =>rst,
                              Q      =>BUF_2_OUT
                            );
 BUF_1 :
       dff_with_hold_input_ctrl
          PORT MAP (
                              D1     =>BUF_2_OUT,
                              D2     =>SWAP_OUT,
                              clk    =>clk,
                              hold   =>hold_buf_1,
                              in_sel =>in_ctrl_buf_1,
                              rst    =>rst,
                              Q      =>BUF_1_OUT
                            );
 MUX :
       mux_2to1_32b
          PORT MAP (
                              D1  =>SWAP_OUT,
                              D2  =>BUF_1_OUT,
                              Y   =>MUX_OUT,
                              S   =>pos_hold_ctrl
                            );
 LEAD_BUF :
       dff_with_hold_32b
          PORT MAP (
                              D   =>MUX_OUT,
                              clk =>clk,
                              hold=>hold_all_seg,
                              rst =>rst,
                              Q   =>LEAD_BUF_OUT
                            );
 SEGMEN_6 :
       dff_segment_for_input
          PORT MAP (
                              D   =>  LEAD_BUF_OUT,
                              clk =>  clk,
                              hold=>  hold_all_seg,
                              rst =>  rst,
                              Q   =>  SEGMENT_6_OUT
                            );
 SEGMEN_5 :
       dff_segment_for_input
```

```
            PORT MAP (
                                D    =>   SEGMENT_6_OUT,
                                clk  =>  clk,
                                hold=>  hold_all_seg,
                                rst  =>  rst,
                                Q    =>  SEGMENT_5_OUT
                              );
    SEGMEN_4 :
          dff_segment_for_input
            PORT MAP (
                                D    =>  SEGMENT_5_OUT,
                                clk  =>  clk,
                                hold=>  hold_all_seg,
                                rst  =>  rst,
                                Q    =>  SEGMENT_4_OUT
                              );
    SEGMEN_3 :
          dff_segment_for_input
            PORT MAP (
                                D    =>  SEGMENT_4_OUT,
                                clk  =>  clk,
                                hold=>  hold_all_seg,
                                rst  =>  rst,
                                Q    =>  SEGMENT_3_OUT
                              );
    SEGMEN_2 :
          dff_segment_for_input
            PORT MAP (
                                D    =>  SEGMENT_3_OUT,
                                clk  =>  clk,
                                hold=>  hold_all_seg,
                                rst  =>  rst,
                                Q    =>  SEGMENT_2_OUT
                              );
    SEGMEN_1 :
          dff_segment_for_input
            PORT MAP (
                                D    =>  SEGMENT_2_OUT,
                                clk  =>  clk,
                                hold=>  hold_all_seg,
                                rst  =>  rst,
                                Q    =>  SEGMENT_1_OUT
                              );
    SEGMEN_0 :
          dff_segment_for_input
            PORT MAP (
                                D    =>  SEGMENT_1_OUT,
                                clk  =>  clk,
                                hold=>  hold_all_seg,
                                rst  =>  rst,
                                Q    =>  SEGMENT_0_OUT
                              );
    END structural;
```

D. Blok FFT 8-titik

Perancangan FFT 8-point dilakukan dengan mengikuti hubungan *input* dan *output* tersebut yang telah disederhanakan dari sisi pemakaian hardware dengan menggunakan *signal flow graph* yang telah diberikan pada halaman sebelumnya. Gambar rangkaian keseluruhan dari FFT 8-point ini diberikan sebagai berikut.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

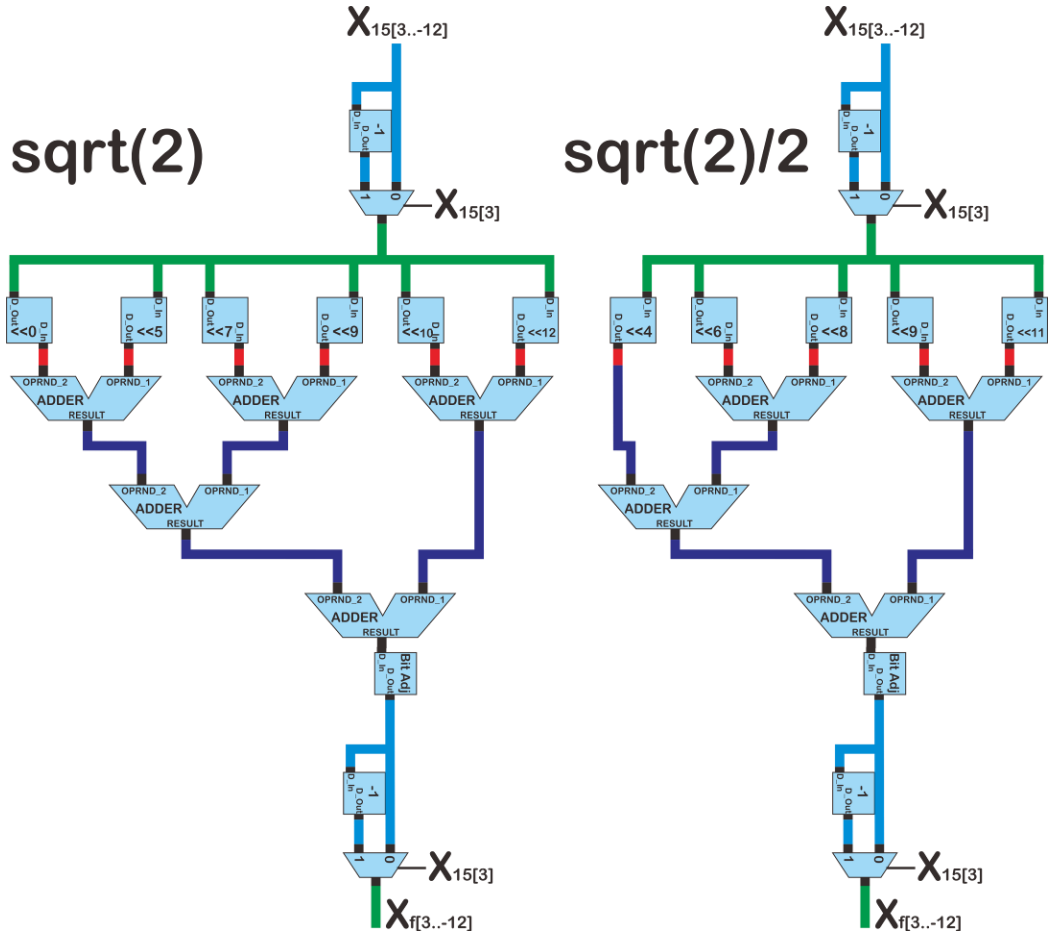
Pada dasarnya terdapat empat elemen dasar dari FFT 8-point yang akan diimplementasikan.

1. Complex Adder, direalisasikan menggunakan dua buah Carry-Lookahead Adder.
2. Complex Substractor, direalisasikan menggunakan dua buah Carry-Lookahead Substractor. Pada dasarnya, substractor dapat diperoleh dengan melakukan modifikasi pada Adder menggunakan prinsip Two's Complement.
3. Sign Inverter, direalisasikan menggunakan Substractor dengan salah satu operan bernilai nol.
4. Complex Multiplier, digunakan untuk melakukan perkalian dengan *twiddle factor*.

Pembuatan Complex Multiplier dilakukan dengan cara yang berbeda, yaitu melakukan konfigurasi secara manual, mengikuti dengan yang dilakukan pada perancangan FFT 8-titik ini.

Twiddle Factor	**Nilai ($C + jS$)**	C	$C + S$	$C - S$
${W_n}^0$	$1 + j0$	1	1	1
${W_n}^1$	$\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	0	$\sqrt{2}$
${W_n}^2$	$0 - j1$	0	-1	1
${W_n}^3$	$-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{2}}{2}$	$-\sqrt{2}$	0

Dengan demikian, kita memerlukan satu buah multiplier saja, yaitu multiplier dengan $\sqrt{2}$. Komputasi perkalian $\frac{\sqrt{2}}{2}$ dapat dilakukan dengan menggunakan *right shifter* satu kali. Namun untuk mengefisiensikan implementasi, perkalian $\frac{\sqrt{2}}{2}$ juga akan diimplementasikan terpisah. Kedua multiplier tersebut dapat dilihat pada diagram berikut.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Implementasi tersebut menggunakan left shifter 16-bit ke 32-bit, 32-bit Carry-Lookahead Adder, sign inverter 16-bit, multiplexer 2-to-1 16-bit, dan bit adjustment 32-bit ke 16-bit. Perkalian dilakukan dengan mengambil nilai absolut pada input data untuk kemudian dikembalikan tandanya yang sesuai di akhir perkalian. Selain itu, bit adjustment berguna untuk melakukan pemotongan (*truncation*) terhadap hasil perkalian. Seperti yang kita ketahui, perkalian dua bilangan 16-bit akan menghasilkan bilangan 32-bit. Pemotongan dilakukan berdasarkan posisi fraksionalnya. Implementasi dalam kode VHDL diberikan sebagai berikut.

const_mult_cla_16b_halfsqrtof2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY const_mult_cla_16b_halfsqrtof2 IS
  PORT  (
            Data_In     :     IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out    :     OUT   STD_LOGIC_VECTOR(15 DOWNTO 0)
         );
END const_mult_cla_16b_halfsqrtof2;
ARCHITECTURE structural of const_mult_cla_16b_halfsqrtof2 IS
COMPONENT adder_cla_32b IS
  PORT  (
            A32         :     IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
            B32         :     IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32         :     OUT   STD_LOGIC_VECTOR (31 DOWNTO 0);
            C_OUT32     :     OUT   STD_LOGIC
         );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT  (
            A16         :     IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16         :     OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16     :     OUT STD_LOGIC
         );
END COMPONENT;
COMPONENT bit_adj_32b_to_16b IS
  PORT  (
            Data_In     :     IN    std_logic_vector (31 DOWNTO 0);
            Data_Out    :     OUT   std_logic_vector (15 DOWNTO 0)
         );
END COMPONENT;
COMPONENT mux_2to1_16b IS
  PORT  (
            D1          :     IN    std_logic_vector (15 DOWNTO 0);
            D2          :     IN    std_logic_vector (15 DOWNTO 0);
            Y           :     OUT   std_logic_vector (15 DOWNTO 0);
            S           :     IN    std_logic
         );
END COMPONENT;
COMPONENT lshift_4_16b_to_32b IS
  PORT  (
            Data_In     :     IN    std_logic_vector (15 DOWNTO 0);
            Data_Out    :     OUT   std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_6_16b_to_32b IS
  PORT  (
            Data_In     :     IN    std_logic_vector (15 DOWNTO 0);
            Data_Out    :     OUT   std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_8_16b_to_32b IS
  PORT  (
            Data_In     :     IN    std_logic_vector (15 DOWNTO 0);
            Data_Out    :     OUT   std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
```

```
COMPONENT lshift_9_16b_to_32b IS
  PORT  (
           Data_In    :     IN    std_logic_vector (15 DOWNTO 0);
           Data_Out   :     OUT   std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT lshift_11_16b_to_32b IS
  PORT  (
           Data_In    :     IN    std_logic_vector (15 DOWNTO 0);
           Data_Out   :     OUT   std_logic_vector (31 DOWNTO 0)
         );
END COMPONENT;
SIGNAL mux_0_out      : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL mux_1_out      : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninv_0_out   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninv_1_out   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL lshift_4_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_6_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_8_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_9_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_11_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_0_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_1_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_2_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_3_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL bitadj_out     : STD_LOGIC_VECTOR  (15 DOWNTO 0);

BEGIN
  Data_Out <= mux_1_out;
  -- Port Mapping
  sgninv_0 :
       sgninv_16b
         PORT MAP (
                             A16   =>       Data_In,
                             R16 => sgninv_0_out
           );
  mux_0  :
       mux_2to1_16b
         PORT MAP (
                             D1    =>       Data_In,
                             D2    =>       sgninv_0_out,
                             Y     =>       mux_0_out,
                             S     =>       Data_In(15)
           );
  lshift_4  :
       lshift_4_16b_to_32b
         PORT MAP (
                       Data_In    => mux_0_out,
                       Data_Out   => lshift_4_out
           );
  lshift_6  :
       lshift_6_16b_to_32b
         PORT MAP (
                       Data_In    => mux_0_out,
                       Data_Out   => lshift_6_out
           );
  lshift_8  :
       lshift_8_16b_to_32b
         PORT MAP (
                       Data_In    => mux_0_out,
                       Data_Out   => lshift_8_out
           );
  lshift_9  :
       lshift_9_16b_to_32b
         PORT MAP (
                       Data_In    => mux_0_out,
                       Data_Out   => lshift_9_out
           );
```

```
  lshift_11:
       lshift_11_16b_to_32b
        PORT MAP (
                       Data_In    => mux_0_out,
                       Data_Out   => lshift_11_out
            );
  adder_0   :
       adder_cla_32b
        PORT MAP (
                       A32   =>    lshift_6_out,
                       B32   =>    lshift_8_out,
                       R32   =>    adder_0_out
            );
  adder_1   :
       adder_cla_32b
        PORT MAP (
                       A32   =>    lshift_9_out,
                       B32   =>    lshift_11_out,
                       R32   =>    adder_1_out
            );
  adder_2   :
       adder_cla_32b
        PORT MAP (
                       A32   =>    lshift_4_out,
                       B32   =>    adder_0_out,
                       R32   =>    adder_2_out
            );
  adder_3   :
       adder_cla_32b
        PORT MAP (
                       A32   =>    adder_2_out,
                       B32   =>    adder_1_out,
                       R32   =>    adder_3_out
            );
  bitadj    :
       bit_adj_32b_to_16b
        PORT MAP (
                       Data_In    => adder_3_out,
                       Data_Out   => bitadj_out
            );
  sgninv_1  :
       sgninv_16b
        PORT MAP (
                       A16   =>    bitadj_out,
                       R16 =>      sgninv_1_out
            );
  mux_1  :
       mux_2to1_16b
        PORT MAP (
                       D1    =>    bitadj_out,
                       D2    =>    sgninv_1_out,
                       Y     =>    mux_1_out,
                       S     =>    Data_In(15)
            );
END structural;
```

const_mult_cla_16b_sqrtof2.vhd

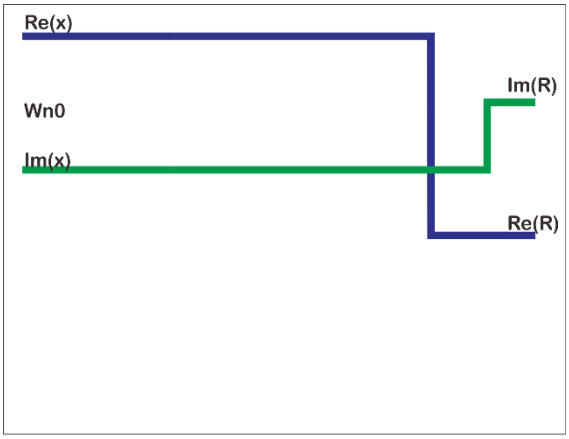
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY const_mult_cla_16b_sqrtof2 IS
  PORT  (
            Data_In    :  IN   STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out   :  OUT  STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
END const_mult_cla_16b_sqrtof2;
ARCHITECTURE structural of const_mult_cla_16b_sqrtof2 IS
COMPONENT adder_cla_32b IS
  PORT  (
            A32         :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
```

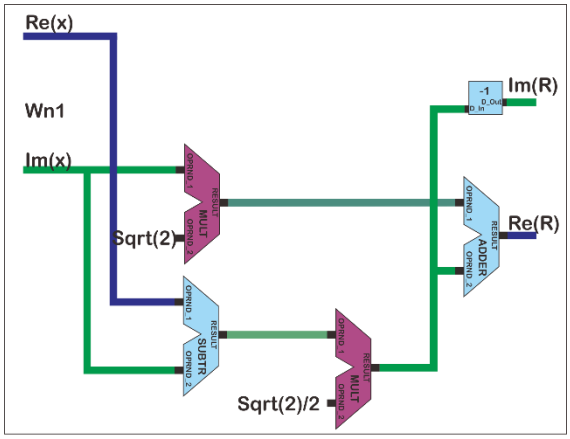
```
            B32         :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32         :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            C_OUT32     :  OUT STD_LOGIC
        );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT  (
            A16         :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16         :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16     :  OUT STD_LOGIC
        );
END COMPONENT;
COMPONENT bit_adj_32b_to_16b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (31 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (15 DOWNTO 0)
        );
END COMPONENT;
COMPONENT mux_2to1_16b IS
  PORT  (
            D1          :  IN  std_logic_vector (15 DOWNTO 0);
            D2          :  IN  std_logic_vector (15 DOWNTO 0);
            Y           :  OUT std_logic_vector (15 DOWNTO 0);
            S           :  IN  std_logic
        );
END COMPONENT;
COMPONENT lshift_0_16b_to_32b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (15 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT lshift_5_16b_to_32b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (15 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT lshift_7_16b_to_32b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (15 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT lshift_9_16b_to_32b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (15 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT lshift_10_16b_to_32b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (15 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT lshift_12_16b_to_32b IS
  PORT  (
            Data_In     :  IN  std_logic_vector (15 DOWNTO 0);
            Data_Out    :  OUT std_logic_vector (31 DOWNTO 0)
        );
END COMPONENT;

SIGNAL mux_0_out      : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL mux_1_out      : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninv_0_out   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninv_1_out   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL lshift_0_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
```

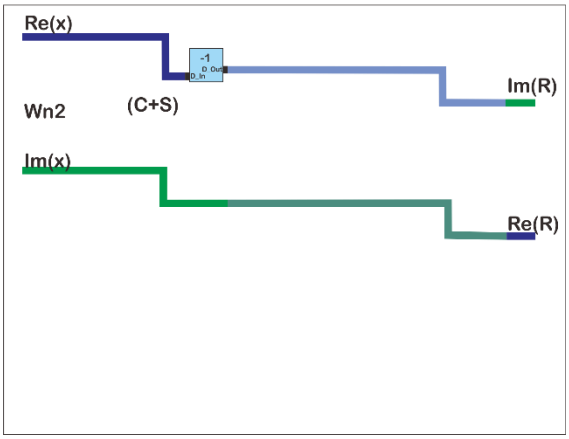
```
SIGNAL lshift_5_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_7_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_9_out   : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_10_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL lshift_12_out  : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_0_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_1_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_2_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_3_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL adder_4_out    : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL bitadj_out     : STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
 Data_Out <= mux_1_out;
 -- Port Mapping
 sgninv_0 :
      sgninv_16b
       PORT MAP (
                        A16 =>    Data_In,
                        R16 =>    sgninv_0_out
          );
 mux_0  :
      mux_2to1_16b
       PORT MAP (
                        D1  =>    Data_In,
                        D2  =>    sgninv_0_out,
                        Y   =>    mux_0_out,
                        S   =>    Data_In(15)
          );
 lshift_0 :
      lshift_0_16b_to_32b
       PORT MAP (
                        Data_In   =>     mux_0_out,
                        Data_Out  =>     lshift_0_out
          );
 lshift_5 :
      lshift_5_16b_to_32b
       PORT MAP (
                        Data_In   =>     mux_0_out,
                        Data_Out  =>     lshift_5_out
          );
 lshift_7 :
      lshift_7_16b_to_32b
       PORT MAP (
                        Data_In   =>     mux_0_out,
                        Data_Out  =>     lshift_7_out
          );
 lshift_9 :
      lshift_9_16b_to_32b
       PORT MAP (
                        Data_In   =>     mux_0_out,
                        Data_Out  =>     lshift_9_out
          );
 lshift_10:
      lshift_10_16b_to_32b
       PORT MAP (
                        Data_In   =>     mux_0_out,
                        Data_Out  =>     lshift_10_out
          );
 lshift_12:
      lshift_12_16b_to_32b
       PORT MAP (
                        Data_In   =>     mux_0_out,
                        Data_Out  =>     lshift_12_out
          );
 adder_0  :
      adder_cla_32b
       PORT MAP (
                        A32 =>    lshift_0_out,
```

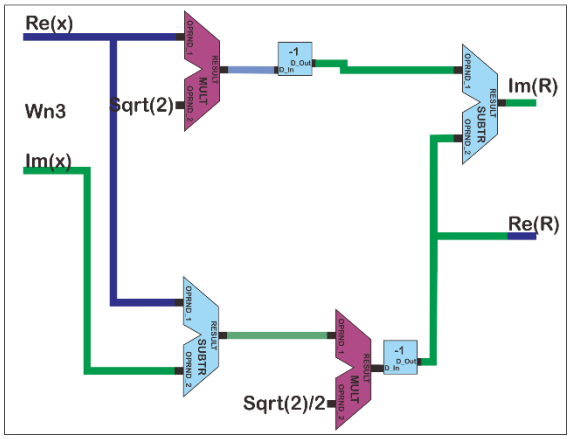
```
                        B32 =>    lshift_5_out,
                        R32 =>    adder_0_out
            );
  adder_1   :
        adder_cla_32b
         PORT MAP (
                        A32 =>    lshift_7_out,
                        B32 =>    lshift_9_out,
                        R32 =>    adder_1_out
            );
  adder_2   :
        adder_cla_32b
         PORT MAP (
                        A32 =>    lshift_10_out,
                        B32 =>    lshift_12_out,
                        R32 =>    adder_2_out
                      );
  adder_3   :
        adder_cla_32b
         PORT MAP (
                        A32 =>    adder_0_out,
                        B32 =>    adder_1_out,
                        R32 =>    adder_3_out
                      );
  adder_4   :
        adder_cla_32b
         PORT MAP (
                        A32 =>    adder_3_out,
                        B32 =>    adder_2_out,
                        R32 =>    adder_4_out
            );
  bitadj    :
        bit_adj_32b_to_16b
         PORT MAP (
                        Data_In   =>     adder_4_out,
                        Data_Out  =>     bitadj_out
            );
  sgninv_1  :
        sgninv_16b
         PORT MAP (
                        A16 =>    bitadj_out,
                        R16 =>    sgninv_1_out
            );
  mux_1  :
        mux_2to1_16b
         PORT MAP (
                        D1  =>    bitadj_out,
                        D2  =>    sgninv_1_out,
                        Y   =>    mux_1_out,
                        S   =>    Data_In(15)
            );
END structural;
```

Selanjutnya, kita dapat mengimplementasikan perkalian dengan masing-masing *twiddle factor* sebagai berikut. Komponen MULT akan direalisasikan sesuai dengan perkalian yang akan dilakukan, apakah dengan akar dua atau dengan setengah akar dua. Khusus untuk *twiddle factor* yang trivial (bernilai 1), perkalian dengan *twiddle factor* hanya meneruskan sinyal saja. Diagram blok perkalian dengan masing-masing *twiddle factor* diberikan pada halaman selanjutnya.









Dengan menggunakan diagram tersebut, realisasi kode VHDL untuk masing-masing perkalian *twiddle factor* diberikan sebagai berikut.

complex_mult_twiddle_wn0_32b.vhd

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

ENTITY complex_mult_twiddle_wn0_32b IS
  PORT   (
              A32     :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
              R32     :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END complex_mult_twiddle_wn0_32b;
ARCHITECTURE structural OF complex_mult_twiddle_wn0_32b IS
SIGNAL REAL_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
  REAL_A32           <= A32(31 DOWNTO 16);
  IMAG_A32           <= A32(15 DOWNTO 0);
  R32(31 DOWNTO 16)  <= REAL_A32;
  R32(15 DOWNTO 0)   <= IMAG_A32;
END structural;
```

complex_mult_twiddle_wn1_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY complex_mult_twiddle_wn1_32b IS
  PORT  (
            A32      :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32      :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END complex_mult_twiddle_wn1_32b;
ARCHITECTURE structural OF complex_mult_twiddle_wn1_32b IS
COMPONENT const_mult_cla_16b_sqrtof2 IS
  PORT  (
```

```
            Data_In :  IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out:  OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
END COMPONENT;
COMPONENT const_mult_cla_16b_halfsqrtof2 IS
  PORT  (
            Data_In :  IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out:  OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
END COMPONENT;
COMPONENT adder_cla_16b IS
  PORT  (
            A16     :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            B16     :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16     :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16 :  OUT STD_LOGIC
        );
END COMPONENT;
COMPONENT subst_cla_16b IS
  PORT  (
            A16     :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            B16     :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16     :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16 :  OUT STD_LOGIC
        );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT  (
            A16     :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16     :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16 :  OUT STD_LOGIC
        );
END COMPONENT;
SIGNAL REAL_A32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MULT_0_OUT     : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL HALFMULT_0_OUT: STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SUBSTR_0_OUT   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL ADDER_0_OUT    : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_0_OUT   : STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
  REAL_A32          <= A32(31 DOWNTO 16);
  IMAG_A32          <= A32(15 DOWNTO 0);
  REAL_R32          <= ADDER_0_OUT;
  IMAG_R32          <= SGNINV_0_OUT;
  R32(31 DOWNTO 16)<= REAL_R32;
  R32(15 DOWNTO 0) <= IMAG_R32;
  -- Port Mapping
  MULT_0 :
       const_mult_cla_16b_sqrtof2
         PORT MAP
            (
                 Data_In  => IMAG_A32,
                 Data_Out => MULT_0_OUT
            );
  SUBSTR_0 :
       subst_cla_16b
         PORT MAP
            (
                 A16  => REAL_A32,
                 B16  => IMAG_A32,
                 R16  => SUBSTR_0_OUT
            );
  HALFMULT_0 :
       const_mult_cla_16b_halfsqrtof2
         PORT MAP
```

```
            (
                  Data_In  => SUBSTR_0_OUT,
                  Data_Out => HALFMULT_0_OUT
            );
  ADDER_0 :
        adder_cla_16b
         PORT MAP
            (
                  A16   => MULT_0_OUT,
                  B16   => HALFMULT_0_OUT,
                  R16   => ADDER_0_OUT
            );
  SGNINV_0 :
        sgninv_16b
         PORT MAP
            (
                  A16   => HALFMULT_0_OUT,
                  R16   => SGNINV_0_OUT
            );
END structural;
```

complex_mult_twiddle_wn2_32b.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY complex_mult_twiddle_wn2_32b IS
  PORT  (
            A32       :  IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32       :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END complex_mult_twiddle_wn2_32b;
ARCHITECTURE structural OF complex_mult_twiddle_wn2_32b IS
COMPONENT sgninv_16b IS
  PORT  (
            A16       :  IN   STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16       :  OUT  STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16   :  OUT  STD_LOGIC
         );
END COMPONENT;
SIGNAL REAL_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL sgninvout: STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
  REAL_A32           <= A32(31 DOWNTO 16);
  IMAG_A32           <= A32(15 DOWNTO 0);
  REAL_R32           <= IMAG_A32;
  IMAG_R32           <= sgninvout;
  R32(31 DOWNTO 16)  <= REAL_R32;
  R32(15 DOWNTO 0)   <= IMAG_R32;
  -- Port Map
  sgninv :
        sgninv_16b
         PORT MAP (
                     A16    =>REAL_A32,
                     R16    =>sgninvout
                  );
END structural;
```

complex_mult_twiddle_wn3_32b

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY complex_mult_twiddle_wn3_32b IS
  PORT  (
            A32       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            R32       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END complex_mult_twiddle_wn3_32b;
ARCHITECTURE structural OF complex_mult_twiddle_wn3_32b IS
COMPONENT const_mult_cla_16b_sqrtof2 IS
```

```
  PORT  (
            Data_In   : IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out  : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0)
         );
END COMPONENT;
COMPONENT const_mult_cla_16b_halfsqrtof2 IS
  PORT  (
            Data_In   : IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
            Data_Out  : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0)
         );
END COMPONENT;
COMPONENT subst_cla_16b IS
  PORT  (
            A16       : IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
            B16       : IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16       : OUT   STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16   : OUT   STD_LOGIC
         );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT  (
            A16       : IN    STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16       : OUT   STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16   : OUT   STD_LOGIC
         );
END COMPONENT;
SIGNAL REAL_A32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32       : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MULT_0_OUT     : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL HALFMULT_0_OUT: STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SUBSTR_0_OUT   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SUBSTR_1_OUT   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_0_OUT   : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_1_OUT   : STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
  REAL_A32           <= A32(31 DOWNTO 16);
  IMAG_A32           <= A32(15 DOWNTO 0);
  REAL_R32           <= SGNINV_1_OUT;
  IMAG_R32           <= SUBSTR_1_OUT;
  R32(31 DOWNTO 16)  <= REAL_R32;
  R32(15 DOWNTO 0)   <= IMAG_R32;
  -- Port Mapping
  MULT_0 :
       const_mult_cla_16b_sqrtof2
         PORT MAP
            (
                Data_In   => REAL_A32,
                Data_Out  => MULT_0_OUT
            );
  SUBSTR_0 :
       subst_cla_16b
         PORT MAP
            (
                A16   => REAL_A32,
                B16   => IMAG_A32,
                R16   => SUBSTR_0_OUT
            );
  HALFMULT_0 :
       const_mult_cla_16b_halfsqrtof2
         PORT MAP
            (
                Data_In   => SUBSTR_0_OUT,
                Data_Out  => HALFMULT_0_OUT
            );
  SGNINV_0 :
       sgninv_16b
```

```
        PORT MAP
           (
               A16   => MULT_0_OUT,
               R16   => SGNINV_0_OUT
           );
  SGNINV_1 :
        sgninv_16b
        PORT MAP
           (
               A16   => HALFMULT_0_OUT,
               R16   => SGNINV_1_OUT
           );
  SUBSTR_1 :
        subst_cla_16b
        PORT MAP
           (
               A16   => SGNINV_0_OUT,
               B16   => SGNINV_1_OUT,
               R16   => SUBSTR_1_OUT
           );
END structural;
```

Dengan demikian, lengkap sudah komponen dasar yang diperlukan untuk merealisasikan FFT 8-point. Berikut ini adalah implementasi Top Level FFT 8-point dalam bahasa VHDL.

fft_8p_16b_top.vhd

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

ENTITY fft_8p_16b_top IS
   PORT  (
               xt0    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt1    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt2    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt3    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt4    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt5    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt6    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xt7    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf0    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf1    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf2    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf3    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf4    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf5    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf6    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
               xf7    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0)
           );
END fft_8p_16b_top;

ARCHITECTURE structural of fft_8p_16b_top IS
COMPONENT complex_mult_twiddle_wn0_32b IS
   PORT  (
               A32          :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               R32          :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
           );
END COMPONENT;
COMPONENT complex_mult_twiddle_wn1_32b IS
   PORT  (
               A32          :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               R32          :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
           );
END COMPONENT;
COMPONENT complex_mult_twiddle_wn2_32b IS
   PORT  (
               A32          :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               R32          :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
```

```
            );
END COMPONENT;
COMPONENT complex_mult_twiddle_wn3_32b IS
   PORT  (
               A32           :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               R32           :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
END COMPONENT;
COMPONENT complex_adder_cla_32b IS
   PORT  (
               A32           :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               B32           :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               R32           :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
               C_OUT32       :      OUT    STD_LOGIC_VECTOR (1 DOWNTO 0)
            );
END COMPONENT;
COMPONENT complex_subst_cla_32b IS
   PORT  (
               A32           :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               B32           :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
               R32           :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
               C_OUT32       :      OUT    STD_LOGIC_VECTOR (1 DOWNTO 0)
            );
END COMPONENT;
SIGNAL CMULT_WN0_0_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN0_1_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN0_2_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN0_3_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN0_4_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN0_5_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN0_6_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN1_0_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN2_0_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN2_1_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN2_2_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CMULT_WN3_0_OUT     :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_0_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_1_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_2_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_3_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_4_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_5_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_6_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_7_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_8_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_9_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_10_OUT       :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CADDER_11_OUT       :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_0_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_1_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_2_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_3_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_4_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_5_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_6_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_7_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_8_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_9_OUT        :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_10_OUT       :      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CSUBTR_11_OUT       :      STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
   xf0   <= CADDER_8_OUT;
   xf1   <= CADDER_9_OUT;
   xf2   <= CADDER_10_OUT;
   xf3   <= CADDER_11_OUT;
   xf4   <= CSUBTR_8_OUT;
   xf5   <= CSUBTR_9_OUT;
   xf6   <= CSUBTR_10_OUT;
```

```
   xf7   <= CSUBTR_11_OUT;
CMULT_WN0_0 :
   complex_mult_twiddle_wn0_32b
        PORT MAP (
                    A32     => xt4,
                    R32     => CMULT_WN0_0_OUT
                );
CMULT_WN0_1 :
   complex_mult_twiddle_wn0_32b
        PORT MAP (
                    A32     => xt6,
                    R32     => CMULT_WN0_1_OUT
                );
CMULT_WN0_2 :
   complex_mult_twiddle_wn0_32b
        PORT MAP (
                    A32     => xt5,
                    R32     => CMULT_WN0_2_OUT
                );
CMULT_WN0_3 :
   complex_mult_twiddle_wn0_32b
        PORT MAP (
                    A32     => xt7,
                    R32     => CMULT_WN0_3_OUT
                );
CADDER_0 :
   complex_adder_cla_32b
        PORT MAP (
                    A32     => xt0,
                    B32     => CMULT_WN0_0_OUT,
                    R32     => CADDER_0_OUT
                );
CSUBTR_0 :
   complex_subst_cla_32b
        PORT MAP (
                    A32     => xt0,
                    B32     => CMULT_WN0_0_OUT,
                    R32     => CSUBTR_0_OUT
                );
CADDER_1 :
   complex_adder_cla_32b
        PORT MAP (
                    A32     => xt2,
                    B32     => CMULT_WN0_1_OUT,
                    R32     => CADDER_1_OUT
                );
CSUBTR_1 :
   complex_subst_cla_32b
        PORT MAP (
                    A32     => xt2,
                    B32     => CMULT_WN0_1_OUT,
                    R32     => CSUBTR_1_OUT
                );
CADDER_2 :
   complex_adder_cla_32b
        PORT MAP (
                    A32     => xt1,
                    B32     => CMULT_WN0_2_OUT,
                    R32     => CADDER_2_OUT
                );
CSUBTR_2 :
   complex_subst_cla_32b
        PORT MAP (
                    A32     => xt1,
                    B32     => CMULT_WN0_2_OUT,
                    R32     => CSUBTR_2_OUT
                );
CADDER_3 :
```

```
   complex_adder_cla_32b
         PORT MAP (
                     A32    => xt3,
                     B32    => CMULT_WN0_3_OUT,
                     R32    => CADDER_3_OUT
                );
CSUBTR_3 :
   complex_subst_cla_32b
         PORT MAP (
                     A32    => xt3,
                     B32    => CMULT_WN0_3_OUT,
                     R32    => CSUBTR_3_OUT
                );
CMULT_WN0_4 :
   complex_mult_twiddle_wn0_32b
         PORT MAP (
                     A32    => CADDER_1_OUT,
                     R32    => CMULT_WN0_4_OUT
                );
CMULT_WN2_0 :
   complex_mult_twiddle_wn2_32b
         PORT MAP (
                     A32    => CSUBTR_1_OUT,
                     R32    => CMULT_WN2_0_OUT
                );
CMULT_WN0_5 :
   complex_mult_twiddle_wn0_32b
         PORT MAP (
                     A32    => CADDER_3_OUT,
                     R32    => CMULT_WN0_5_OUT
                );
CMULT_WN2_1 :
   complex_mult_twiddle_wn2_32b
         PORT MAP (
                     A32    => CSUBTR_3_OUT,
                     R32    => CMULT_WN2_1_OUT
                );
CADDER_4 :
   complex_adder_cla_32b
         PORT MAP (
                     A32    => CADDER_0_OUT,
                     B32    => CMULT_WN0_4_OUT,
                     R32    => CADDER_4_OUT
                );
CADDER_5 :
   complex_adder_cla_32b
         PORT MAP (
                     A32    => CSUBTR_0_OUT,
                     B32    => CMULT_WN2_0_OUT,
                     R32    => CADDER_5_OUT
                );
CSUBTR_4 :
   complex_subst_cla_32b
         PORT MAP (
                     A32    => CADDER_0_OUT,
                     B32    => CMULT_WN0_4_OUT,
                     R32    => CSUBTR_4_OUT
                );
CSUBTR_5 :
   complex_subst_cla_32b
         PORT MAP (
                     A32    => CSUBTR_0_OUT,
                     B32    => CMULT_WN2_0_OUT,
                     R32    => CSUBTR_5_OUT
                );
CADDER_6 :
   complex_adder_cla_32b
         PORT MAP (
```

```
                    A32     => CADDER_2_OUT,
                    B32     => CMULT_WN0_5_OUT,
                    R32     => CADDER_6_OUT
                );
CADDER_7 :
   complex_adder_cla_32b
         PORT MAP (
                    A32     => CSUBTR_2_OUT,
                    B32     => CMULT_WN2_1_OUT,
                    R32     => CADDER_7_OUT
                );
CSUBTR_6 :
   complex_subst_cla_32b
         PORT MAP (
                    A32     => CADDER_2_OUT,
                    B32     => CMULT_WN0_5_OUT,
                    R32     => CSUBTR_6_OUT
                );
CSUBTR_7 :
   complex_subst_cla_32b
         PORT MAP (
                    A32     => CSUBTR_2_OUT,
                    B32     => CMULT_WN2_1_OUT,
                    R32     => CSUBTR_7_OUT
                );
CMULT_WN0_6 :
   complex_mult_twiddle_wn0_32b
         PORT MAP (
                    A32     => CADDER_6_OUT,
                    R32     => CMULT_WN0_6_OUT
                );
CMULT_WN1_0 :
   complex_mult_twiddle_wn1_32b
         PORT MAP (
                    A32     => CADDER_7_OUT,
                    R32     => CMULT_WN1_0_OUT
                );
CMULT_WN2_2 :
   complex_mult_twiddle_wn2_32b
         PORT MAP (
                    A32     => CSUBTR_6_OUT,
                    R32     => CMULT_WN2_2_OUT
                );
CMULT_WN3_0 :
   complex_mult_twiddle_wn3_32b
         PORT MAP (
                    A32     => CSUBTR_7_OUT,
                    R32     => CMULT_WN3_0_OUT
                );
CADDER_8 :
   complex_adder_cla_32b
         PORT MAP (
                    A32     => CADDER_4_OUT,
                    B32     => CMULT_WN0_6_OUT,
                    R32     => CADDER_8_OUT
                );
CADDER_9 :
   complex_adder_cla_32b
         PORT MAP (
                    A32     => CADDER_5_OUT,
                    B32     => CMULT_WN1_0_OUT,
                    R32     => CADDER_9_OUT
                );
CADDER_10 :
   complex_adder_cla_32b
         PORT MAP (
                    A32     => CSUBTR_4_OUT,
                    B32     => CMULT_WN2_2_OUT,
```

```
                    R32     => CADDER_10_OUT
                );
CADDER_11 :
   complex_adder_cla_32b
         PORT MAP (
                    A32     => CSUBTR_5_OUT,
                    B32     => CMULT_WN3_0_OUT,
                    R32     => CADDER_11_OUT
                );
CSUBTR_8 :
   complex_subst_cla_32b
         PORT MAP (
                    A32     => CADDER_4_OUT,
                    B32     => CMULT_WN0_6_OUT,
                    R32     => CSUBTR_8_OUT
                );
CSUBTR_9 :
   complex_subst_cla_32b
         PORT MAP (
                    A32     => CADDER_5_OUT,
                    B32     => CMULT_WN1_0_OUT,
                    R32     => CSUBTR_9_OUT
                );
CSUBTR_10 :
   complex_subst_cla_32b
         PORT MAP (
                    A32     => CSUBTR_4_OUT,
                    B32     => CMULT_WN2_2_OUT,
                    R32     => CSUBTR_10_OUT
                );
CSUBTR_11 :
   complex_subst_cla_32b
         PORT MAP (
                    A32     => CSUBTR_5_OUT,
                    B32     => CMULT_WN3_0_OUT,
                    R32     => CSUBTR_11_OUT
                );
END structural;
```

E. Blok Interdimensional Multiplier

Blok ini merupakan blok yang sangat kompleks dalam rangkaian ini. Blok ini digunakan untuk melakukan perkalian dengan konstanta twiddle faktor interdimensional (W_{64}). Dalam pembahasan sebelumnya, hanya 49 buah twiddle faktor interdimensional yang digunakan. Namun, kita tidak akan mengimplementasikannya satu persatu karena blok multiplier merupakan blok yang memakan area yang sangat besar. Oleh karena itu, diusahakan blok multiplier ini dapat digunakan secara berulang-ulang.

Untuk mendesain Interdimensional Multiplier, kita perlu melihat dulu sifat matematika dari perkalian bilangan kompleks sebagai berikut sebagai tambahan dari sebelumnya telah dijelaskan mengenai perancangan perkalian bilangan kompleks dengan rangkaian yang sederhana. Diketahui tabel berikut yang merupakan perkalian bilangan kompleks variabel tetap dengan bilangan kompleks konstanta dengan tanda dan posisi real-imaginary yang diubah-ubah.

Type	Input Operand 1	Constant Operand	Result
1	$A + jB$	$C + jD$	$AC - BD + j(AD + BC)$
2	$A + jB$	$C - jD$	$AC + BD + j(-AD + BC)$
3	$A + jB$	$-C + jD$	$-AC - BD + j(AD - BC)$
4	$A + jB$	$-C - jD$	$-AC + BD + j(-AD - BC)$
5	$A + jB$	$D + jC$	$AD - BC + j(AC + BD)$
6	$A + jB$	$D - jC$	$AD + BC + j(-AC + BD)$

7	$A + jB$	$-D + jC$	$-AD - BC + j(AC - BD)$
8	$A + jB$	$-D - jC$	$-AD + BC + j(-AC - BD)$

Seandainya Constant Operand memiliki tanda dan posisi yang tetap, namun Input Operand 1 kita ubah-ubah tanda dan posisi real-imaginary, maka untuk memperoleh hasil yang sama, diperlukan pemrosesan sebagai berikut.

Type	Input Operand	Pre-Processing	Constant Operand	Result	Post Processing
1	$A + jB$	NONE $A + jB$	$C + jD$	$AC - BD + j(AD + BC)$	NONE $AC - BD + j(AD + BC)$
2	$A + jB$	IM_INVERT $A - jB$	$C + jD$	$AC + BD + j(AD - BC)$	IM_INVERT $AC + BD + j(-AD + BC)$
3	$A + jB$	RE_INVERT $-A + jB$	$C + jD$	$-AC - BD + j(-AD + BC)$	IM_INVERT $-AC - BD + j(AD - BC)$
4	$A + jB$	RE_INVERT IM_INVERT $-A - jB$	$C + jD$	$-AC + BD + j(-AD - BC)$	NONE $-AC + BD + j(-AD - BC)$
5	$A + jB$	RE-IM SWAP $B + jA$	$C + jD$	$-AD + BC + j(AC + BD)$	RE_INVERT $AD - BC + j(AC + BD)$
6	$A + jB$	RE-IM SWAP IM_INVERT $B - jA$	$C + jD$	$AD + BC + j(-AC + BD)$	NONE $AD + BC + j(-AC + BD)$
7	$A + jB$	RE-IM SWAP RE_INVERT $-B + jA$	$C + jD$	$-AD - BC + j(AC - BD)$	NONE $-AD - BC + j(AC - BD)$
8	$A + jB$	RE-IM SWAP RE_INVERT IM_INVERT $-B - jA$	$C + jD$	$AD - BC + j(-AC - BD)$	RE_INVERT $-AD + BC + j(-AC - BD)$

Dengan demikian, kita dapat menetapkan konstanta standar yang akan dijadikan perkalian tetap untuk beberapa twiddle factor dengan mengubah tanda atau posisi real-imaginary. Terdapat sembilan konstanta tetap (satu berupa konstanta trivial) yang ditetapkan sebagai standar berikut.

Constant	**Nilai**
0	$1{,}000000000000000 + j0{,}000000000000000$
1	$0{,}995184726672197 + j0{,}098017140329561$
2	$0{,}980785280403230 + j0{,}195090322016128$
3	$0{,}956940335732209 + j0{,}290284677254462$
4	$0{,}923879532511287 + j0{,}382683432365090$
5	$0{,}881921264348355 + j0{,}471396736825998$
6	$0{,}831469612302545 + j0{,}555570233019602$
7	$0{,}773010453362737 + j0{,}634393284163645$
8	$0{,}707106781186548 + j0{,}707106781186547$

Dengan demikian, sebelum dioperasikan menggunakan perkalian bilangan kompleks, kita harus mengatur terlebih dahulu (pre-process) input bilangan kompleks variabel sesuai dengan tabel sebelumnya. Selain itu, terdapat post-process yang juga harus dilakukan setelah perkalian bilangan kompleks dilakukan. Dengan demikian terdapat delapan buah tipe pre-process dan post-process yang harus dilakukan.

TYPE	Pre Processing			Post Processing			Control Signal
	RE-IM Interchange	RE	IM	RE-IM Interchange	RE	IM	
0	NO	POS	POS	NO	POS	POS	000
1	NO	POS	NEG	NO	POS	NEG	001
2	NO	NEG	POS	NO	POS	NEG	010
3	No	NEG	NEG	NO	POS	POS	011
4	YES	POS	POS	NO	NEG	POS	100
5	YES	POS	NEG	NO	POS	POS	101
6	YES	NEG	POS	NO	POS	POS	110
7	YES	NEG	NEG	NO	NEG	POS	111

Dengan demikian, kita dapat menuliskan kembali Twiddle Factor Interdimensional beserta konstanta yang berhubungan dan operasi yang harus dilakukan.

W_{64}	Nilai	Constant dan Type
0	$1{,}000000000000000 + j0{,}000000000000000$	CONSTANT_0_TYPE_1
1	$0{,}995184726672197 - j0{,}098017140329561$	CONSTANT_1_TYPE_1
2	$0{,}980785280403230 - j0{,}195090322016128$	CONSTANT_2_TYPE_1
3	$0{,}956940335732209 - j0{,}290284677254462$	CONSTANT_3_TYPE_1
4	$0{,}923879532511287 - j0{,}382683432365090$	CONSTANT_4_TYPE_1
5	$0{,}881921264348355 - j0{,}471396736825998$	CONSTANT_5_TYPE_1
6	$0{,}831469612302545 - j0{,}555570233019602$	CONSTANT_6_TYPE_1
7	$0{,}773010453362737 - j0{,}634393284163645$	CONSTANT_7_TYPE_1
8	$0{,}707106781186548 - j0{,}707106781186547$	CONSTANT_8_TYPE_1
9	$0{,}634393284163645 - j0{,}773010453362737$	CONSTANT_7_TYPE_5
10	$0{,}555570233019602 - j0{,}831469612302545$	CONSTANT_6_TYPE_5
11	$0{,}471396736825998 - j0{,}881921264348355$	CONSTANT_5_TYPE_5
12	$0{,}382683432365090 - j0{,}923879532511287$	CONSTANT_4_TYPE_5
13	$0{,}290284677254462 - j0{,}956940335732209$	CONSTANT_3_TYPE_5
14	$0{,}195090322016128 - j0{,}980785280403230$	CONSTANT_2_TYPE_5
15	$0{,}098017140329561 - j0{,}995184726672197$	CONSTANT_1_TYPE_5
16	$0{,}000000000000000 - j1{,}000000000000000$	CONSTANT_0_TYPE_5
17	$-0{,}098017140329561 - j0{,}995184726672197$	CONSTANT_1_TYPE_7
18	$-0{,}195090322016128 - j0{,}980785280403230$	CONSTANT_2_TYPE_7
19	$-0{,}290284677254462 - j0{,}956940335732209$	CONSTANT_3_TYPE_7
20	$-0{,}382683432365090 - j0{,}923879532511287$	CONSTANT_4_TYPE_7
21	$-0{,}471396736825998 - j0{,}881921264348355$	CONSTANT_5_TYPE_7
22	$-0{,}555570233019602 - j0{,}831469612302545$	CONSTANT_6_TYPE_7
23	$-0{,}634393284163645 - j0{,}773010453362737$	CONSTANT_7_TYPE_7
24	$-0{,}707106781186547 - j0{,}707106781186548$	CONSTANT_8_TYPE_7
25	$-0{,}773010453362737 - j0{,}634393284163645$	CONSTANT_7_TYPE_3
26	$-0{,}831469612302545 - j0{,}555570233019602$	CONSTANT_6_TYPE_3
27	$-0{,}881921264348355 - j0{,}471396736825998$	CONSTANT_5_TYPE_3
28	$-0{,}923879532511287 - j0{,}382683432365090$	CONSTANT_4_TYPE_3
29	$-0{,}956940335732209 - j0{,}290284677254462$	CONSTANT_3_TYPE_3
30	$-0{,}980785280403230 - j0{,}195090322016129$	CONSTANT_2_TYPE_3
31	$-0{,}995184726672197 - j0{,}098017140329561$	CONSTANT_1_TYPE_3
32	$-1{,}000000000000000 + j0{,}000000000000000$	CONSTANT_0_TYPE_2
33	$-0{,}995184726672197 + j0{,}098017140329561$	CONSTANT_1_TYPE_2

34	$-0{,}980785280403230 + j0{,}195090322016128$	CONSTANT_2_TYPE_2
35	$-0{,}956940335732209 + j0{,}290284677254462$	CONSTANT_3_TYPE_2
36	$-0{,}923879532511287 + j0{,}382683432365090$	CONSTANT_4_TYPE_2
37	$-0{,}881921264348355 + j0{,}471396736825998$	CONSTANT_5_TYPE_2
38	$-0{,}831469612302545 + j0{,}555570233019602$	CONSTANT_6_TYPE_2
39	$-0{,}773010453362737 + j0{,}634393284163645$	CONSTANT_7_TYPE_2
40	$-0{,}707106781186548 + j0{,}707106781186547$	CONSTANT_8_TYPE_2
41	$-0{,}634393284163646 + j0{,}773010453362737$	CONSTANT_7_TYPE_6
42	$-0{,}555570233019602 + j0{,}831469612302545$	CONSTANT_6_TYPE_6
43	$-0{,}471396736825998 + j0{,}881921264348355$	CONSTANT_5_TYPE_6
44	$-0{,}382683432365090 + j0{,}923879532511287$	CONSTANT_4_TYPE_6
45	$-0{,}290284677254462 + j0{,}956940335732209$	CONSTANT_3_TYPE_6
46	$-0{,}195090322016129 + j0{,}980785280403230$	CONSTANT_2_TYPE_6
47	$-0{,}098017140329561 + j0{,}995184726672197$	CONSTANT_1_TYPE_6
48	$0{,}000000000000000 + j1{,}000000000000000$	CONSTANT_0_TYPE_6
49	$0{,}098017140329560 + j0{,}995184726672197$	CONSTANT_1_TYPE_4
50	$0{,}195090322016128 + j0{,}980785280403230$	CONSTANT_2_TYPE_4
51	$0{,}290284677254462 + j0{,}956940335732209$	CONSTANT_3_TYPE_4
52	$0{,}382683432365090 + j0{,}923879532511287$	CONSTANT_4_TYPE_4
53	$0{,}471396736825998 + j0{,}881921264348355$	CONSTANT_5_TYPE_4
54	$0{,}555570233019602 + j0{,}831469612302545$	CONSTANT_6_TYPE_4
55	$0{,}634393284163646 + j0{,}773010453362737$	CONSTANT_7_TYPE_4
56	$0{,}707106781186547 + j0{,}707106781186548$	CONSTANT_8_TYPE_4
57	$0{,}773010453362737 + j0{,}634393284163646$	CONSTANT_7_TYPE_0
58	$0{,}831469612302545 + j0{,}555570233019602$	CONSTANT_6_TYPE_0
59	$0{,}881921264348355 + j0{,}471396736825998$	CONSTANT_5_TYPE_0
60	$0{,}923879532511287 + j0{,}382683432365090$	CONSTANT_4_TYPE_0
61	$0{,}956940335732209 + j0{,}290284677254462$	CONSTANT_3_TYPE_0
62	$0{,}980785280403230 + j0{,}195090322016129$	CONSTANT_2_TYPE_0
63	$0{,}995184726672197 + j0{,}098017140329561$	CONSTANT_1_TYPE_0

Berdasarkan analisis matematika yang telah dijelaskan sebelumnya, terdapat beberapa data dalam satu set data yang menggunakan CONSTANT yang sama. Hal ini di atasi dengan melakukan perkalian dalam beberapa clock cycle. Berikut ini adalah pemakaian CONSTANT untuk setiap set data.

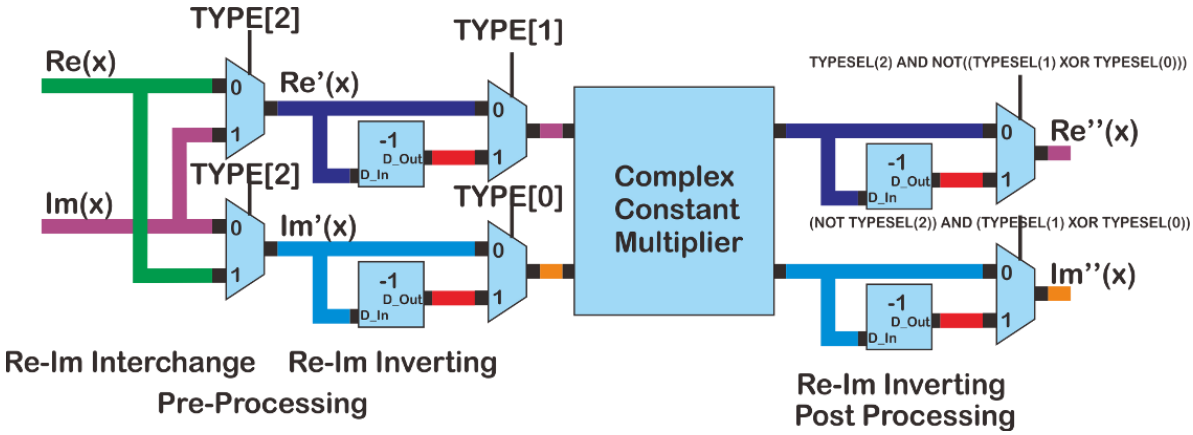
SET	DATA0		DATA1		DATA2		DATA3		DATA4		DATA5		DATA6		DATA7	
	CONST	TY	CONST	TY	CONST	TY	CONST	TY	CONST	TY	CONST	TY	CONST	TY	CONST	TY
SET 0	BYPASS	1	BYPASS	1	BYPASS	1	BYPASS	1	BYPASS	1	BYPASS	1	BYPASS	1	BYPASS	1
SET 1	BYPASS	1	1	1	2	1	3	1	4	1	5	1	6	1	7	1
SET 2	BYPASS	1	2	1	4	1	6	1	8	1	HOLD	HL	HOLD		HOLD	HL
	HOLD	1	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	6	5	4	5	2	5
SET 3	BYPASS	1	3	1	6	1	7	5	4	5	1	5	2	7	5	7
SET 4	BYPASS	1	4	1	8	1	HOLD	HL	BYPASS	5	HOLD	HL	HOLD	HL	HOLD	HL
	HOLD	HL	HOLD	HL	HOLD	HL	4	5	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL
	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	4	7	8	7	HOLD	HL
	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	4	3
SET 5	BYPASS	1	5	1	6	5	1	5	4	7	7	3	2	3	3	2
SET 6	BYPASS	1	6	1	4	5	2	7	8	7	HOLD	HL	HOLD	HL	HOLD	HL
	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	HOLD	HL	2	3	4	2	6	6
SET 7	BYPASS	1	7	1	2	5	5	7	4	3	3	2	6	6	1	

Berikut ini adalah nilai C+S, C-S, dan C dari masing-masing konstanta baik dalam desimal maupun dalam format Q4.12 Fixed Point.

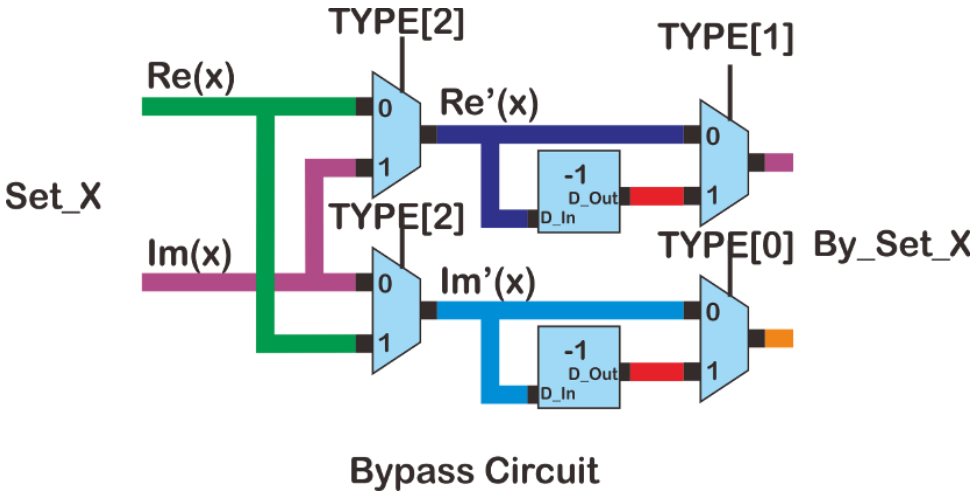
Constant	C+jS	C+S	C	C-S
Constant 0	1,000000000000000 + *j*0,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
Constant 1	0,995184726672197 + *j*0,098017140329561	1,093201867001760	0,995184726672197	0,897167586342636
Constant 2	0,980785280403230 + *j*0,195090322016128	1,175875602419360	0,980785280403230	0,785694958387102
Constant 3	0,956940335732209 + *j*0,290284677254462	1,247225012986670	0,956940335732209	0,666655658477747
Constant 4	0,923879532511287 + *j*0,382683432365090	1,306562964876380	0,923879532511287	0,541196100146197
Constant 5	0,881921264348355 + *j*0,471396736825998	1,353318001174350	0,881921264348355	0,410524527522357
Constant 6	0,831469612302545 + *j*0,555570233019602	1,387039845322150	0,831469612302545	0,275899379282943
Constant 7	0,773010453362737 + *j*0,634393284163645	1,407403737526380	0,773010453362737	0,138617169199092
Constant 8	0,707106781186547 + *j*0,707106781186547	1,414213562373090	0,707106781186547	0,000000000000000

Constant	C+jS	C+S	C	C-S
Constant 0	1,000000000000000 + *j*0,000000000000000	0001000000000000	0001000000000000	0001000000000000
Constant 1	0,995184726672197 + *j*0,098017140329561	0001000101111110	0000111111101100	0000111001011011
Constant 2	0,980785280403230 + *j*0,195090322016128	0001001011010000	0000111110110001	0000110010010010
Constant 3	0,956940335732209 + *j*0,290284677254462	0001001111110101	0000111101010000	0000101010101011
Constant 4	0,923879532511287 + *j*0,382683432365090	0001010011101000	0000111011001000	0000100010101001
Constant 5	0,881921264348355 + *j*0,471396736825998	0001010110100111	0000111000011100	0000011010010010
Constant 6	0,831469612302545 + *j*0,555570233019602	0001011000110001	0000110101001110	0000010001101010
Constant 7	0,773010453362737 + *j*0,634393284163645	0001011010000101	0000110001011110	0000001000111000
Constant 8	0,707106781186547 + *j*0,707106781186547	0001011010100001	0000101101010000	0000000000000000

Dengan demikian, struktur multiplier yang digunakan adalah sebagai berikut.

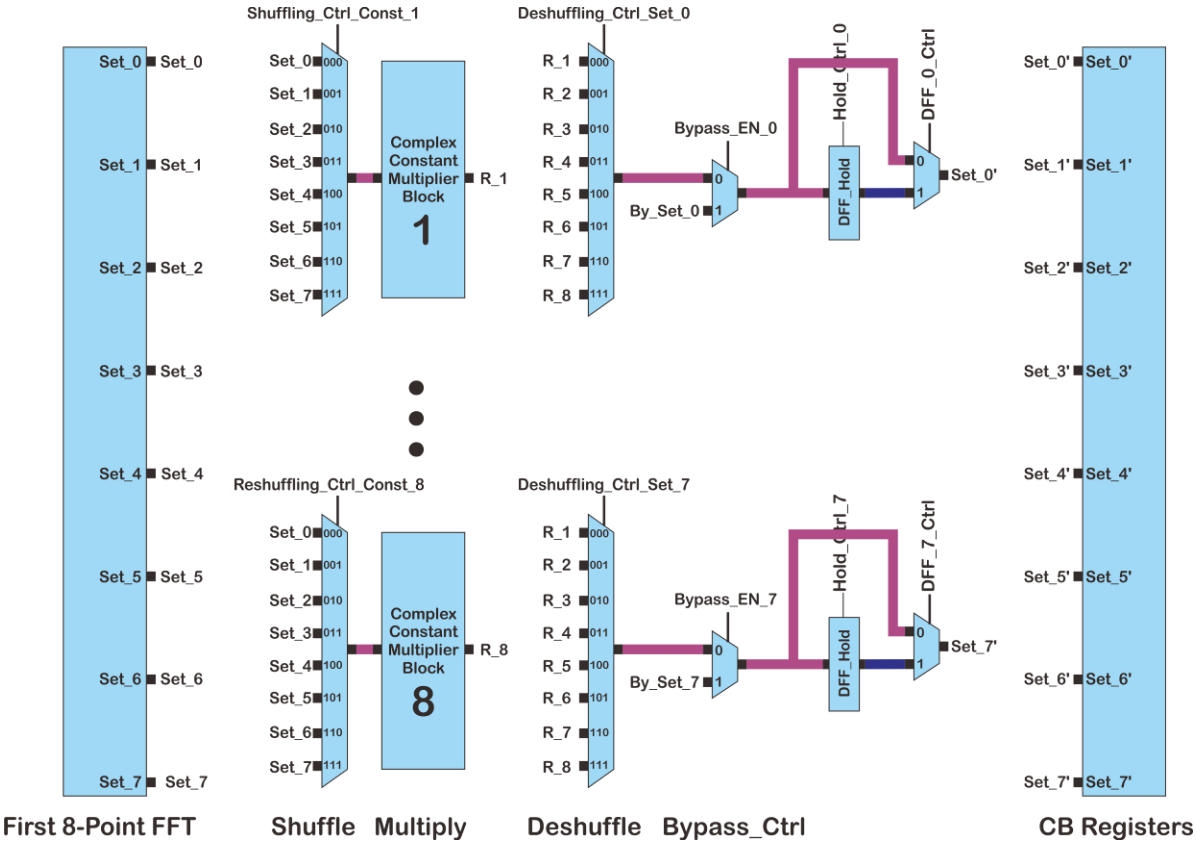


Untuk CONSTANT_0, operasi yang dilakukan hanya *bypassing* saja sehingga rangkaian menjadi seperti berikut.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Blok Interdimensional Constant Multiplier dapat digambarkan sebagai berikut.



Terdapat register DFF_Hold diakhir operasi. Register ini digunakan untuk menyimpan data hasil perkalian ketika sebuah konstanta digunakan secara berulang untuk memproses set data yang sama seperti yang telah dijelaskan sebelumnya.

bypass_32b.vhd

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

ENTITY bypass_32b IS
  PORT  (
            A32     :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            TYPESEL :  IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            R32     :  OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END bypass_32b;

ARCHITECTURE structural OF bypass_32b IS
COMPONENT mux_2to1_16b IS
  PORT  (
            D1      :  IN  std_logic_vector (15 DOWNTO 0);
            D2      :  IN  std_logic_vector (15 DOWNTO 0);
            Y       :  OUT std_logic_vector (15 DOWNTO 0);
            S       :  IN  std_logic
        );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT  (
            A16     :  IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16     :  OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16 :  OUT STD_LOGIC
        );
END COMPONENT;
```

```
SIGNAL REAL_A32         : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32         : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32         : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32         : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_0_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_1_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_2_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_3_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_0_OUT     : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_1_OUT     : STD_LOGIC_VECTOR (15 DOWNTO 0);

BEGIN
  REAL_A32           <= A32(31 DOWNTO 16);
  IMAG_A32           <= A32(15 DOWNTO 0);
  REAL_R32           <= MUX_2_OUT;
  IMAG_R32           <= MUX_3_OUT;
  R32(31 DOWNTO 16)  <= REAL_R32;
  R32(15 DOWNTO 0)   <= IMAG_R32;


  -- Port Mapping
  MUX_0 :
       mux_2to1_16b
        PORT MAP
            (
                    D1 =>REAL_A32,
                    D2 =>IMAG_A32,
                    Y  =>MUX_0_OUT,
                    S  =>TYPESEL(2)
            );
  MUX_1 :
       mux_2to1_16b
        PORT MAP
            (
                    D1 =>IMAG_A32,
                    D2 =>REAL_A32,
                    Y  =>MUX_1_OUT,
                    S  =>TYPESEL(2)
            );
  -- Sign Inversion Circuit
  SGNINV_0 :
       sgninv_16b
        PORT MAP
            (
                    A16   =>MUX_0_OUT,
                    R16 =>SGNINV_0_OUT
            );
  SGNINV_1 :
       sgninv_16b
        PORT MAP
            (
                    A16   =>MUX_1_OUT,
                    R16 =>SGNINV_1_OUT
            );
  MUX_2 :
       mux_2to1_16b
        PORT MAP
            (
                    D1 =>MUX_0_OUT,
                    D2 =>SGNINV_0_OUT,
                    Y  =>MUX_2_OUT,
                    S  =>TYPESEL(1)
            );
  MUX_3 :
       mux_2to1_16b
        PORT MAP
```

```
            (
                    D1 =>MUX_1_OUT,
                    D2 =>SGNINV_1_OUT,
                    Y  =>MUX_3_OUT,
                    S  =>TYPESEL(0)
            );
END structural;
```

generic_complex_mult_block.vhd

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

ENTITY generic_complex_mult_block IS
  GENERIC (
            C_PLUS_S:     STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
            C_ONLY  :   STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
            C_MIN_S :     STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000"

            );
  PORT    (
            A32           :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            TYPESEL       :      IN     STD_LOGIC_VECTOR (2 DOWNTO 0);
            R32           :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
           );
END generic_complex_mult_block;

ARCHITECTURE structural OF generic_complex_mult_block IS
COMPONENT generic_complex_mult_16b IS
  GENERIC (
            C_PLUS_S:     STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
            C_ONLY  :     STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
            C_MIN_S :     STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000"

            );
  PORT    (
            REAL_A32:     IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
            IMAG_A32:     IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
            REAL_R32:     OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
            IMAG_R32:     OUT    STD_LOGIC_VECTOR (15 DOWNTO 0)
           );
END COMPONENT;
COMPONENT mux_2to1_16b IS
  PORT    (
            D1     :      IN     std_logic_vector (15 DOWNTO 0);
            D2     :      IN     std_logic_vector (15 DOWNTO 0);
            Y      :      OUT    std_logic_vector (15 DOWNTO 0);
            S      :      IN     std_logic
           );
END COMPONENT;
COMPONENT sgninv_16b IS
  PORT    (
            A16           :      IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
            R16           :      OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
            C_OUT16       :      OUT    STD_LOGIC
           );
END COMPONENT;

SIGNAL REAL_A32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_MULT_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_MULT_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_0_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_1_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_2_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_3_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_4_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_5_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```
SIGNAL SGNINV_0_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_1_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_2_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL SGNINV_3_OUT        : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_4_SELECTOR      : STD_LOGIC;
SIGNAL MUX_5_SELECTOR      : STD_LOGIC;

BEGIN
  REAL_A32          <= A32(31 DOWNTO 16);
  IMAG_A32          <= A32(15 DOWNTO 0);
  REAL_R32          <= MUX_4_OUT;
  IMAG_R32          <= MUX_5_OUT;
  R32(31 DOWNTO 16) <= REAL_R32;
  R32(15 DOWNTO 0)  <= IMAG_R32;
  MUX_4_SELECTOR    <=TYPESEL(2) AND NOT((TYPESEL(1) XOR TYPESEL(0)));
  MUX_5_SELECTOR    <=(NOT TYPESEL(2)) AND (TYPESEL(1) XOR TYPESEL(0));


  -- Port Mapping
  -- PRE PROCESSING
  -- Swapping Mux
  MUX_0   :
          mux_2to1_16b
             PORT MAP
              (
                    D1     =>REAL_A32,
                    D2     =>IMAG_A32,
                    Y      =>MUX_0_OUT,
                    S      =>TYPESEL(2)
              );
  MUX_1   :
          mux_2to1_16b
             PORT MAP
              (
                    D1     =>IMAG_A32,
                    D2     =>REAL_A32,
                    Y      =>MUX_1_OUT,
                    S      =>TYPESEL(2)
              );
  -- Sign Inversion Circuit
  SGNINV_0 :
          sgninv_16b
             PORT MAP
              (
                    A16    =>MUX_0_OUT,
                    R16    =>SGNINV_0_OUT
              );
  SGNINV_1 :
          sgninv_16b
             PORT MAP
              (
                    A16    =>MUX_1_OUT,
                    R16    =>SGNINV_1_OUT
              );
  MUX_2   :
          mux_2to1_16b
             PORT MAP
              (
                    D1     =>MUX_0_OUT,
                    D2     =>SGNINV_0_OUT,
                    Y      =>MUX_2_OUT,
                    S      =>TYPESEL(1)
              );
  MUX_3   :
          mux_2to1_16b
             PORT MAP
              (
                    D1     =>MUX_1_OUT,
```

```
                    D2      =>SGNINV_1_OUT,
                    Y       =>MUX_3_OUT,
                    S       =>TYPESEL(0)
                );
    -- MULTIPLICATION
    COMPLEX_MULTIPLIER :
            generic_complex_mult_16b
               GENERIC MAP
                (
                    C_PLUS_S      =>C_PLUS_S,
                    C_ONLY        =>C_ONLY,
                    C_MIN_S       =>C_MIN_S
                )
               PORT MAP
                (
                    REAL_A32      =>MUX_2_OUT,
                    IMAG_A32      =>MUX_3_OUT,
                    REAL_R32      =>REAL_MULT_OUT,
                    IMAG_R32      =>IMAG_MULT_OUT
                );
    -- POST PROCESSING
    -- Sign Inversion Circuit
    SGNINV_2 :
            sgninv_16b
               PORT MAP
                (
                    A16     =>REAL_MULT_OUT,
                    R16     =>SGNINV_2_OUT
                );
    SGNINV_3 :
            sgninv_16b
               PORT MAP
                (
                    A16     =>IMAG_MULT_OUT,
                    R16     =>SGNINV_3_OUT
                );
    MUX_4    :
            mux_2to1_16b
               PORT MAP
                (
                    D1      =>REAL_MULT_OUT,
                    D2      =>SGNINV_2_OUT,
                    Y       =>MUX_4_OUT,
                    S       =>MUX_4_SELECTOR
                );
    MUX_5    :
            mux_2to1_16b
               PORT MAP
                (
                    D1      =>IMAG_MULT_OUT,
                    D2      =>SGNINV_3_OUT,
                    Y       =>MUX_5_OUT,
                    S       =>MUX_5_SELECTOR
                );
END structural;
```

interdimensional_multiplier.vhd

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

ENTITY interdimensional_multiplier IS
  PORT (
            clk             :       IN  STD_LOGIC;
            rst             :       IN  STD_LOGIC;
            SET_0_IN        :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_1_IN        :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_2_IN        :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_3_IN        :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_4_IN        :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
```

```
            SET_5_IN        :      IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_6_IN        :      IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_7_IN        :      IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            Shuf_Ctrl_1     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_2     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_3     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_4     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_5     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_6     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_7     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Shuf_Ctrl_8     :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_1      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_2      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_3      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_4      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_5      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_6      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_7      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Type_Sel_8      :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_0    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_1    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_2    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_3    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_4    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_5    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_6    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_Sel_7    :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_0   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_1   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_2   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_3   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_4   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_5   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_6   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            DeShuf_Ctrl_7   :      IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
            Bypass_EN_0     :      IN  STD_LOGIC;
            Bypass_EN_1     :      IN  STD_LOGIC;
            Bypass_EN_2     :      IN  STD_LOGIC;
            Bypass_EN_3     :      IN  STD_LOGIC;
            Bypass_EN_4     :      IN  STD_LOGIC;
            Bypass_EN_5     :      IN  STD_LOGIC;
            Bypass_EN_6     :      IN  STD_LOGIC;
            Bypass_EN_7     :      IN  STD_LOGIC;
            Hold_Ctrl_0     :      IN  STD_LOGIC;
            Hold_Ctrl_1     :      IN  STD_LOGIC;
            Hold_Ctrl_2     :      IN  STD_LOGIC;
            Hold_Ctrl_3     :      IN  STD_LOGIC;
            Hold_Ctrl_4     :      IN  STD_LOGIC;
            Hold_Ctrl_5     :      IN  STD_LOGIC;
            Hold_Ctrl_6     :      IN  STD_LOGIC;
            Hold_Ctrl_7     :      IN  STD_LOGIC;
            DFF_Ctrl_0      :      IN  STD_LOGIC;
            DFF_Ctrl_1      :      IN  STD_LOGIC;
            DFF_Ctrl_2      :      IN  STD_LOGIC;
            DFF_Ctrl_3      :      IN  STD_LOGIC;
            DFF_Ctrl_4      :      IN  STD_LOGIC;
            DFF_Ctrl_5      :      IN  STD_LOGIC;
            DFF_Ctrl_6      :      IN  STD_LOGIC;
            DFF_Ctrl_7      :      IN  STD_LOGIC;
            SET_0_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_1_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_2_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_3_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_4_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_5_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_6_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            SET_7_OUT       :      OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
```

```
END interdimensional_multiplier;

ARCHITECTURE structural OF interdimensional_multiplier IS
COMPONENT generic_complex_mult_block IS
  GENERIC (
           C_PLUS_S: STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
           C_ONLY  : STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000";
           C_MIN_S : STD_LOGIC_VECTOR (15 DOWNTO 0) := "0000000000000000"

         );
  PORT (
           A32     :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
           TYPESEL :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
           R32     :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT bypass_32b IS
  PORT (
           A32     :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
           TYPESEL :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
           R32     :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END COMPONENT;
COMPONENT mux_2to1_32b IS
  PORT (
           D1              :      IN     std_logic_vector (31 DOWNTO 0);
           D2              :      IN     std_logic_vector (31 DOWNTO 0);
           Y               :      OUT    std_logic_vector (31 DOWNTO 0);
           S               :      IN     std_logic
         );
END COMPONENT;
COMPONENT mux_8to1_32b IS
  PORT (
           D1              :      IN     std_logic_vector (31 DOWNTO 0);
           D2              :      IN     std_logic_vector (31 DOWNTO 0);
           D3              :      IN     std_logic_vector (31 DOWNTO 0);
           D4              :      IN     std_logic_vector (31 DOWNTO 0);
           D5              :      IN     std_logic_vector (31 DOWNTO 0);
           D6              :      IN     std_logic_vector (31 DOWNTO 0);
           D7              :      IN     std_logic_vector (31 DOWNTO 0);
           D8              :      IN     std_logic_vector (31 DOWNTO 0);
           Y               :      OUT    std_logic_vector (31 DOWNTO 0);
           S               :      IN     std_logic_vector (2 DOWNTO 0)
         );
END COMPONENT;
COMPONENT dff_with_hold_32b IS
  PORT (
           D               :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
           clk             :      IN     STD_LOGIC;
           hold            :      IN     STD_LOGIC;
           rst             :      IN     STD_LOGIC;
           Q               :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END COMPONENT;
SIGNAL MUX_SHUFFLE_1_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_2_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_3_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_4_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_5_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_6_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_7_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_SHUFFLE_8_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_1_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_2_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_3_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_4_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_5_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_6_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
```

```
SIGNAL BLOCK_MULT_7_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_MULT_8_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_0_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_1_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_2_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_3_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_4_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_5_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_6_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL BLOCK_BYPASS_7_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_0_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_1_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_2_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_3_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_4_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_5_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_6_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_DESHUFFLE_7_OUT  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_0_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_1_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_2_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_3_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_4_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_5_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_6_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_BYPASS_7_OUT     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_0_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_1_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_2_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_3_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_4_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_5_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_6_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DFF_7_OUT            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_0_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_1_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_2_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_3_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_4_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_5_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_6_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL MUX_HOLD_7_OUT       : STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN
  -- Assignment
  SET_0_OUT<= MUX_HOLD_0_OUT;
  SET_1_OUT<= MUX_HOLD_1_OUT;
  SET_2_OUT<= MUX_HOLD_2_OUT;
  SET_3_OUT<= MUX_HOLD_3_OUT;
  SET_4_OUT<= MUX_HOLD_4_OUT;
  SET_5_OUT<= MUX_HOLD_5_OUT;
  SET_6_OUT<= MUX_HOLD_6_OUT;
  SET_7_OUT<= MUX_HOLD_7_OUT;
  -- Port Mapping
  MUX_SHUFFLE_1 :
       mux_8to1_32b
          PORT MAP
            (
                           D1     =>SET_0_IN,
                           D2     =>SET_1_IN,
                           D3     =>SET_2_IN,
                           D4     =>SET_3_IN,
                           D5     =>SET_4_IN,
                           D6     =>SET_5_IN,
                           D7     =>SET_6_IN,
                           D8     =>SET_7_IN,
                           Y      =>MUX_SHUFFLE_1_OUT,
                           S      =>Shuf_Ctrl_1
```

```
          );
MUX_SHUFFLE_2 :
      mux_8to1_32b
         PORT MAP
          (
                              D1     =>SET_0_IN,
                              D2     =>SET_1_IN,
                              D3     =>SET_2_IN,
                              D4     =>SET_3_IN,
                              D5     =>SET_4_IN,
                              D6     =>SET_5_IN,
                              D7     =>SET_6_IN,
                              D8     =>SET_7_IN,
                              Y      =>MUX_SHUFFLE_2_OUT,
                              S      =>Shuf_Ctrl_2
          );
MUX_SHUFFLE_3 :
      mux_8to1_32b
         PORT MAP
          (
                              D1     =>SET_0_IN,
                              D2     =>SET_1_IN,
                              D3     =>SET_2_IN,
                              D4     =>SET_3_IN,
                              D5     =>SET_4_IN,
                              D6     =>SET_5_IN,
                              D7     =>SET_6_IN,
                              D8     =>SET_7_IN,
                              Y      =>MUX_SHUFFLE_3_OUT,
                              S      =>Shuf_Ctrl_3
          );
MUX_SHUFFLE_4 :
      mux_8to1_32b
         PORT MAP
          (
                              D1     =>SET_0_IN,
                              D2     =>SET_1_IN,
                              D3     =>SET_2_IN,
                              D4     =>SET_3_IN,
                              D5     =>SET_4_IN,
                              D6     =>SET_5_IN,
                              D7     =>SET_6_IN,
                              D8     =>SET_7_IN,
                              Y      =>MUX_SHUFFLE_4_OUT,
                              S      =>Shuf_Ctrl_4
          );
MUX_SHUFFLE_5 :
      mux_8to1_32b
         PORT MAP
          (
                              D1     =>SET_0_IN,
                              D2     =>SET_1_IN,
                              D3     =>SET_2_IN,
                              D4     =>SET_3_IN,
                              D5     =>SET_4_IN,
                              D6     =>SET_5_IN,
                              D7     =>SET_6_IN,
                              D8     =>SET_7_IN,
                              Y      =>MUX_SHUFFLE_5_OUT,
                              S      =>Shuf_Ctrl_5
          );
MUX_SHUFFLE_6 :
      mux_8to1_32b
         PORT MAP
          (
                              D1     =>SET_0_IN,
                              D2     =>SET_1_IN,
                              D3     =>SET_2_IN,
```

```
                        D4      =>SET_3_IN,
                        D5      =>SET_4_IN,
                        D6      =>SET_5_IN,
                        D7      =>SET_6_IN,
                        D8      =>SET_7_IN,
                        Y       =>MUX_SHUFFLE_6_OUT,
                        S       =>Shuf_Ctrl_6
          );
MUX_SHUFFLE_7 :
     mux_8to1_32b
        PORT MAP
          (
                        D1      =>SET_0_IN,
                        D2      =>SET_1_IN,
                        D3      =>SET_2_IN,
                        D4      =>SET_3_IN,
                        D5      =>SET_4_IN,
                        D6      =>SET_5_IN,
                        D7      =>SET_6_IN,
                        D8      =>SET_7_IN,
                        Y       =>MUX_SHUFFLE_7_OUT,
                        S       =>Shuf_Ctrl_7
          );
MUX_SHUFFLE_8 :
     mux_8to1_32b
        PORT MAP
          (
                        D1      =>SET_0_IN,
                        D2      =>SET_1_IN,
                        D3      =>SET_2_IN,
                        D4      =>SET_3_IN,
                        D5      =>SET_4_IN,
                        D6      =>SET_5_IN,
                        D7      =>SET_6_IN,
                        D8      =>SET_7_IN,
                        Y       =>MUX_SHUFFLE_8_OUT,
                        S       =>Shuf_Ctrl_8
          );
BLOCK_MULT_1 :
     generic_complex_mult_block
        GENERIC MAP
          (
                        C_PLUS_S      =>"0001000101111110",
                        C_ONLY        =>"0000111111101100",
                        C_MIN_S       =>"0000111001011011"
          )
        PORT MAP
          (
                        A32           =>MUX_SHUFFLE_1_OUT,
                        TYPESEL       =>Type_Sel_1,
                        R32           =>BLOCK_MULT_1_OUT
          );
BLOCK_MULT_2 :
     generic_complex_mult_block
        GENERIC MAP
          (
                        C_PLUS_S      =>"0001001011010000",
                        C_ONLY        =>"0000111110110001",
                        C_MIN_S       =>"0000110010010010"
          )
        PORT MAP
          (
                        A32           =>MUX_SHUFFLE_2_OUT,
                        TYPESEL       =>Type_Sel_2,
                        R32           =>BLOCK_MULT_2_OUT
          );
BLOCK_MULT_3 :
     generic_complex_mult_block
```

```
          GENERIC MAP
            (
                              C_PLUS_S      =>"0001001111110101",
                              C_ONLY        =>"0000111101010000",
                              C_MIN_S       =>"0000101010101011"
            )
          PORT MAP
            (
                              A32           =>MUX_SHUFFLE_3_OUT,
                              TYPESEL       =>Type_Sel_3,
                              R32           =>BLOCK_MULT_3_OUT
            );
    BLOCK_MULT_4 :
         generic_complex_mult_block
            GENERIC MAP
            (
                              C_PLUS_S      =>"0001010011101000",
                              C_ONLY        =>"0000111011001000",
                              C_MIN_S       =>"0000100010101001"
            )
          PORT MAP
            (
                              A32           =>MUX_SHUFFLE_4_OUT,
                              TYPESEL       =>Type_Sel_4,
                              R32           =>BLOCK_MULT_4_OUT
            );
    BLOCK_MULT_5 :
         generic_complex_mult_block
            GENERIC MAP
            (
                              C_PLUS_S      =>"0001010110100111",
                              C_ONLY        =>"0000111000011100",
                              C_MIN_S       =>"0000011010010010"
          PORT MAP
            (
                              A32           =>MUX_SHUFFLE_5_OUT,
                              TYPESEL       =>Type_Sel_5,
                              R32           =>BLOCK_MULT_5_OUT
            );
    BLOCK_MULT_6 :
         generic_complex_mult_block
            GENERIC MAP
            (
                              C_PLUS_S      =>"0001011000110001",
                              C_ONLY        =>"0000110101001110",
                              C_MIN_S       =>"0000010001101010"
            )
          PORT MAP
            (
                              A32           =>MUX_SHUFFLE_6_OUT,
                              TYPESEL       =>Type_Sel_6,
                              R32           =>BLOCK_MULT_6_OUT
            );
    BLOCK_MULT_7 :
         generic_complex_mult_block
            GENERIC MAP
            (
                              C_PLUS_S      =>"0001011010000101",
                              C_ONLY        =>"0000110001011110",
                              C_MIN_S       =>"0000001000111000"
            )
          PORT MAP
            (
                              A32           =>MUX_SHUFFLE_7_OUT,
                              TYPESEL       =>Type_Sel_7,
                              R32           =>BLOCK_MULT_7_OUT
            );
    BLOCK_MULT_8 :
```

```
      generic_complex_mult_block
         GENERIC MAP
          (
                          C_PLUS_S      =>"0001011010100001",
                          C_ONLY        =>"0000101101010000",
                          C_MIN_S       =>"0000000000000000"
          )
         PORT MAP
          (
                          A32           =>MUX_SHUFFLE_8_OUT,
                          TYPESEL       =>Type_Sel_8,
                          R32           =>BLOCK_MULT_8_OUT
          );

  BLOCK_BYPASS_0 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_0_IN,
                          TYPESEL       =>Bypass_Sel_0,
                          R32           =>BLOCK_BYPASS_0_OUT
          );
  BLOCK_BYPASS_1 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_1_IN,
                          TYPESEL       =>Bypass_Sel_1,
                          R32           =>BLOCK_BYPASS_1_OUT
          );
  BLOCK_BYPASS_2 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_2_IN,
                          TYPESEL       =>Bypass_Sel_2,
                          R32           =>BLOCK_BYPASS_2_OUT
          );
  BLOCK_BYPASS_3 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_3_IN,
                          TYPESEL       =>Bypass_Sel_3,
                          R32           =>BLOCK_BYPASS_3_OUT
          );
  BLOCK_BYPASS_4 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_4_IN,
                          TYPESEL       =>Bypass_Sel_4,
                          R32           =>BLOCK_BYPASS_4_OUT
          );
  BLOCK_BYPASS_5 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_5_IN,
                          TYPESEL       =>Bypass_Sel_5,
                          R32           =>BLOCK_BYPASS_5_OUT
          );
  BLOCK_BYPASS_6 :
      bypass_32b
         PORT MAP
          (
                          A32           =>SET_6_IN,
                          TYPESEL       =>Bypass_Sel_6,
```

```
                        R32         =>BLOCK_BYPASS_6_OUT
        );
BLOCK_BYPASS_7 :
     bypass_32b
        PORT MAP
        (
                        A32         =>SET_7_IN,
                        TYPESEL     =>Bypass_Sel_7,
                        R32         =>BLOCK_BYPASS_7_OUT
        );

MUX_DESHUFFLE_0 :
     mux_8to1_32b
        PORT MAP
        (
                        D1    =>BLOCK_MULT_1_OUT,
                        D2    =>BLOCK_MULT_2_OUT,
                        D3    =>BLOCK_MULT_3_OUT,
                        D4    =>BLOCK_MULT_4_OUT,
                        D5    =>BLOCK_MULT_5_OUT,
                        D6    =>BLOCK_MULT_6_OUT,
                        D7    =>BLOCK_MULT_7_OUT,
                        D8    =>BLOCK_MULT_8_OUT,
                        Y     =>MUX_DESHUFFLE_0_OUT,
                        S     =>DeShuf_Ctrl_0
        );
MUX_DESHUFFLE_1 :
     mux_8to1_32b
        PORT MAP
        (
                        D1    =>BLOCK_MULT_1_OUT,
                        D2    =>BLOCK_MULT_2_OUT,
                        D3    =>BLOCK_MULT_3_OUT,
                        D4    =>BLOCK_MULT_4_OUT,
                        D5    =>BLOCK_MULT_5_OUT,
                        D6    =>BLOCK_MULT_6_OUT,
                        D7    =>BLOCK_MULT_7_OUT,
                        D8    =>BLOCK_MULT_8_OUT,
                        Y     =>MUX_DESHUFFLE_1_OUT,
                        S     =>DeShuf_Ctrl_1
        );
MUX_DESHUFFLE_2 :
     mux_8to1_32b
        PORT MAP
        (
                        D1    =>BLOCK_MULT_1_OUT,
                        D2    =>BLOCK_MULT_2_OUT,
                        D3    =>BLOCK_MULT_3_OUT,
                        D4    =>BLOCK_MULT_4_OUT,
                        D5    =>BLOCK_MULT_5_OUT,
                        D6    =>BLOCK_MULT_6_OUT,
                        D7    =>BLOCK_MULT_7_OUT,
                        D8    =>BLOCK_MULT_8_OUT,
                        Y     =>MUX_DESHUFFLE_2_OUT,
                        S     =>DeShuf_Ctrl_2
        );
MUX_DESHUFFLE_3 :
     mux_8to1_32b
        PORT MAP
        (
                        D1    =>BLOCK_MULT_1_OUT,
                        D2    =>BLOCK_MULT_2_OUT,
                        D3    =>BLOCK_MULT_3_OUT,
                        D4    =>BLOCK_MULT_4_OUT,
                        D5    =>BLOCK_MULT_5_OUT,
                        D6    =>BLOCK_MULT_6_OUT,
                        D7    =>BLOCK_MULT_7_OUT,
                        D8    =>BLOCK_MULT_8_OUT,
```

```
                              Y      =>MUX_DESHUFFLE_3_OUT,
                              S      =>DeShuf_Ctrl_3
            );
   MUX_DESHUFFLE_4 :
         mux_8to1_32b
            PORT MAP
            (
                              D1     =>BLOCK_MULT_1_OUT,
                              D2     =>BLOCK_MULT_2_OUT,
                              D3     =>BLOCK_MULT_3_OUT,
                              D4     =>BLOCK_MULT_4_OUT,
                              D5     =>BLOCK_MULT_5_OUT,
                              D6     =>BLOCK_MULT_6_OUT,
                              D7     =>BLOCK_MULT_7_OUT,
                              D8     =>BLOCK_MULT_8_OUT,
                              Y      =>MUX_DESHUFFLE_4_OUT,
                              S      =>DeShuf_Ctrl_4
            );
   MUX_DESHUFFLE_5 :
         mux_8to1_32b
            PORT MAP
            (
                              D1     =>BLOCK_MULT_1_OUT,
                              D2     =>BLOCK_MULT_2_OUT,
                              D3     =>BLOCK_MULT_3_OUT,
                              D4     =>BLOCK_MULT_4_OUT,
                              D5     =>BLOCK_MULT_5_OUT,
                              D6     =>BLOCK_MULT_6_OUT,
                              D7     =>BLOCK_MULT_7_OUT,
                              D8     =>BLOCK_MULT_8_OUT,
                              Y      =>MUX_DESHUFFLE_5_OUT,
                              S      =>DeShuf_Ctrl_5
            );
   MUX_DESHUFFLE_6 :
         mux_8to1_32b
            PORT MAP
            (
                              D1     =>BLOCK_MULT_1_OUT,
                              D2     =>BLOCK_MULT_2_OUT,
                              D3     =>BLOCK_MULT_3_OUT,
                              D4     =>BLOCK_MULT_4_OUT,
                              D5     =>BLOCK_MULT_5_OUT,
                              D6     =>BLOCK_MULT_6_OUT,
                              D7     =>BLOCK_MULT_7_OUT,
                              D8     =>BLOCK_MULT_8_OUT,
                              Y      =>MUX_DESHUFFLE_6_OUT,
                              S      =>DeShuf_Ctrl_6
            );
   MUX_DESHUFFLE_7 :
         mux_8to1_32b
            PORT MAP
            (
                              D1     =>BLOCK_MULT_1_OUT,
                              D2     =>BLOCK_MULT_2_OUT,
                              D3     =>BLOCK_MULT_3_OUT,
                              D4     =>BLOCK_MULT_4_OUT,
                              D5     =>BLOCK_MULT_5_OUT,
                              D6     =>BLOCK_MULT_6_OUT,
                              D7     =>BLOCK_MULT_7_OUT,
                              D8     =>BLOCK_MULT_8_OUT,
                              Y      =>MUX_DESHUFFLE_7_OUT,
                              S      =>DeShuf_Ctrl_7
            );

   MUX_BYPASS_0 :
         mux_2to1_32b
            PORT MAP
            (
```

```
                              D1     =>MUX_DESHUFFLE_0_OUT,
                              D2     =>BLOCK_BYPASS_0_OUT,
                              Y      =>MUX_BYPASS_0_OUT,
                              S      =>Bypass_EN_0
            );
   MUX_BYPASS_1 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_1_OUT,
                              D2     =>BLOCK_BYPASS_1_OUT,
                              Y      =>MUX_BYPASS_1_OUT,
                              S      =>Bypass_EN_1
            );
   MUX_BYPASS_2 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_2_OUT,
                              D2     =>BLOCK_BYPASS_2_OUT,
                              Y      =>MUX_BYPASS_2_OUT,
                              S      =>Bypass_EN_2
            );
   MUX_BYPASS_3 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_3_OUT,
                              D2     =>BLOCK_BYPASS_3_OUT,
                              Y      =>MUX_BYPASS_3_OUT,
                              S      =>Bypass_EN_3
            );
   MUX_BYPASS_4 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_4_OUT,
                              D2     =>BLOCK_BYPASS_4_OUT,
                              Y      =>MUX_BYPASS_4_OUT,
                              S      =>Bypass_EN_4
            );
   MUX_BYPASS_5 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_5_OUT,
                              D2     =>BLOCK_BYPASS_5_OUT,
                              Y      =>MUX_BYPASS_5_OUT,
                              S      =>Bypass_EN_5
            );
   MUX_BYPASS_6 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_6_OUT,
                              D2     =>BLOCK_BYPASS_6_OUT,
                              Y      =>MUX_BYPASS_6_OUT,
                              S      =>Bypass_EN_6
            );
   MUX_BYPASS_7 :
         mux_2to1_32b
            PORT MAP
            (
                              D1     =>MUX_DESHUFFLE_7_OUT,
                              D2     =>BLOCK_BYPASS_7_OUT,
                              Y      =>MUX_BYPASS_7_OUT,
                              S      =>Bypass_EN_7
            );
```

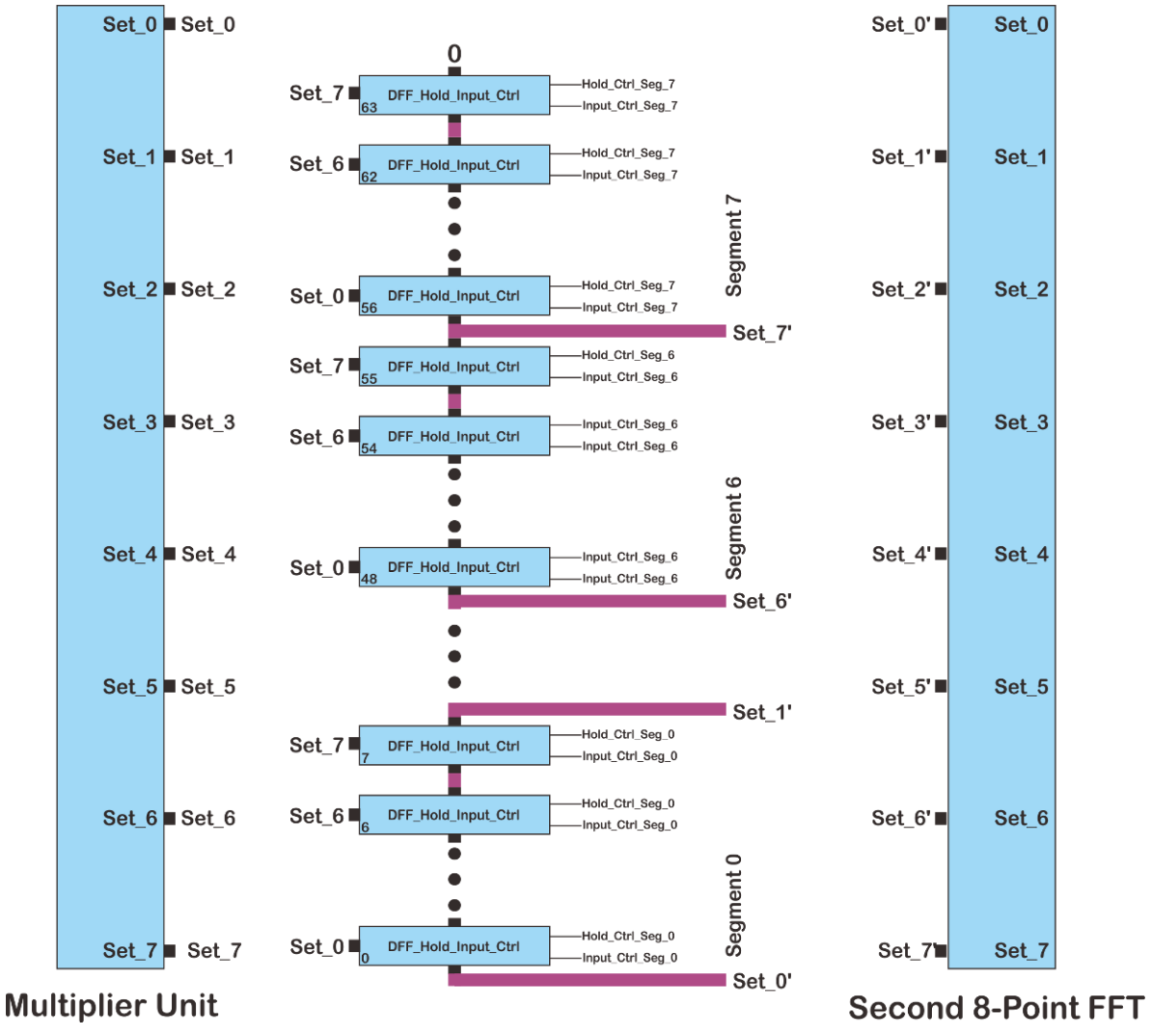
```
    DFF_0 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_0_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_0,
                            rst  => rst,
                            Q    => DFF_0_OUT
             );
    DFF_1 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_1_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_1,
                            rst  => rst,
                            Q    => DFF_1_OUT
             );
    DFF_2 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_2_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_2,
                            rst  => rst,
                            Q    => DFF_2_OUT
             );
    DFF_3 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_3_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_3,
                            rst  => rst,
                            Q    => DFF_3_OUT
             );
    DFF_4 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_4_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_4,
                            rst  => rst,
                            Q    => DFF_4_OUT
             );
    DFF_5 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_5_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_5,
                            rst  => rst,
                            Q    => DFF_5_OUT
             );
    DFF_6 :
        dff_with_hold_32b
           PORT MAP
             (
                            D    => MUX_BYPASS_6_OUT,
                            Clk  => clk,
                            hold=> Hold_Ctrl_6,
                            rst  => rst,
```

```
                        Q    => DFF_6_OUT
        );
  DFF_7 :
      dff_with_hold_32b
         PORT MAP
          (
                        D    => MUX_BYPASS_7_OUT,
                        Clk  => clk,
                        hold=> Hold_Ctrl_7,
                        rst  => rst,
                        Q    => DFF_7_OUT
        );
MUX_HOLD_0 :
      mux_2to1_32b
         PORT MAP
          (
                        D1     =>MUX_BYPASS_0_OUT,
                        D2     =>DFF_0_OUT,
                        Y      =>MUX_HOLD_0_OUT,
                        S      =>DFF_Ctrl_0
        );
MUX_HOLD_1 :
      mux_2to1_32b
         PORT MAP
          (
                        D1     =>MUX_BYPASS_1_OUT,
                        D2     =>DFF_1_OUT,
                        Y      =>MUX_HOLD_1_OUT,
                        S      =>DFF_Ctrl_1
        );
MUX_HOLD_2 :
      mux_2to1_32b
         PORT MAP
          (
                        D1     =>MUX_BYPASS_2_OUT,
                        D2     =>DFF_2_OUT,
                        Y      =>MUX_HOLD_2_OUT,
                        S      =>DFF_Ctrl_2
        );
MUX_HOLD_3 :
      mux_2to1_32b
         PORT MAP
          (
                        D1     =>MUX_BYPASS_3_OUT,
                        D2     =>DFF_3_OUT,
                        Y      =>MUX_HOLD_3_OUT,
                        S      =>DFF_Ctrl_3
        );
MUX_HOLD_4 :
      mux_2to1_32b
         PORT MAP
          (
                        D1     =>MUX_BYPASS_4_OUT,
                        D2     =>DFF_4_OUT,
                        Y      =>MUX_HOLD_4_OUT,
                        S      =>DFF_Ctrl_4
        );
MUX_HOLD_5 :
      mux_2to1_32b
         PORT MAP
          (
                        D1     =>MUX_BYPASS_5_OUT,
                        D2     =>DFF_5_OUT,
                        Y      =>MUX_HOLD_5_OUT,
                        S      =>DFF_Ctrl_5
        );
MUX_HOLD_6 :
      mux_2to1_32b
```

```
            PORT MAP
              (
                                D1      =>MUX_BYPASS_6_OUT,
                                D2      =>DFF_6_OUT,
                                Y       =>MUX_HOLD_6_OUT,
                                S       =>DFF_Ctrl_6
              );
    MUX_HOLD_7 :
          mux_2to1_32b
            PORT MAP
              (
                                D1      =>MUX_BYPASS_7_OUT,
                                D2      =>DFF_7_OUT,
                                Y       =>MUX_HOLD_7_OUT,
                                S       =>DFF_Ctrl_7
              );
END structural;
```

F. Blok Internal Register (CB Unit)

Blok Internal Register (CB Unit) bertugas untuk menerima data dari Interdimensional Constant Multiplier, menyusun dan menyimpan data tersebut, serta meneruskan data tersebut ke FFT 8-titik tahap kedua. Dengan memperhatikan operasi yang dilakukan pada analisis matematika FFT 64-titik, blok internal register (CB-Unit) ini dapat digambarkan sebagai berikut.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Terdapat 64 buah DFF_Hold_Input_Ctrl yang digunakan sebagai register yang terbagi ke dalam 8 buah segmen. Pengaturan register akan dilakukan oleh master control. Berikut ini adalah implementasi VHDL dari internal register.

dff_segment_for_cb.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dff_segment_for_cb IS
  PORT  (
             D1        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D2        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D3        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D4        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D5        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D6        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D7        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D8        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D9        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             clk       :   IN      STD_LOGIC;
             hold      :   IN      STD_LOGIC;
             in_sel    :   IN      STD_LOGIC;
             rst       :   IN      STD_LOGIC;
             Q         :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END dff_segment_for_cb;

ARCHITECTURE structural of dff_segment_for_cb IS
COMPONENT dff_with_hold_input_ctrl IS
  PORT  (
             D1        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             D2        :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
             clk       :   IN      STD_LOGIC;
             hold      :   IN      STD_LOGIC;
             in_sel    :   IN      STD_LOGIC;
             rst       :   IN      STD_LOGIC;
             Q         :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END COMPONENT;
SIGNAL DFF_0_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_1_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_2_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_3_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_4_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_5_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_6_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_7_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
  BEGIN
  Q     <=    DFF_0_OUT;
  -- Port Map
  DFF_7 :
        dff_with_hold_input_ctrl
         PORT MAP (
                        D1  =>D1,
                        D2  =>D9,
                        clk =>clk,
                        hold=>hold,
                        in_sel=>in_sel,
                        rst =>rst,
                        Q   =>DFF_7_OUT
                       );
  DFF_6 :
        dff_with_hold_input_ctrl
         PORT MAP (
                        D1  =>DFF_7_OUT,
                        D2  =>D8,
                        clk =>clk,
```

```
                    hold=>hold,
                    in_sel=>in_sel,
                    rst =>rst,
                    Q   =>DFF_6_OUT
                   );
DFF_5 :
     dff_with_hold_input_ctrl
      PORT MAP (
                    D1  =>DFF_6_OUT,
                    D2  =>D7,
                    clk =>clk,
                    hold=>hold,
                    in_sel=>in_sel,
                    rst =>rst,
                    Q   =>DFF_5_OUT
                   );
DFF_4 :
     dff_with_hold_input_ctrl
      PORT MAP (
                    D1  =>DFF_5_OUT,
                    D2  =>D6,
                    clk =>clk,
                    hold=>hold,
                    in_sel=>in_sel,
                    rst =>rst,
                    Q   =>DFF_4_OUT
                   );
DFF_3 :
     dff_with_hold_input_ctrl
      PORT MAP (
                    D1  =>DFF_4_OUT,
                    D2  =>D5,
                    clk =>clk,
                    hold=>hold,
                    in_sel=>in_sel,
                    rst =>rst,
                    Q   =>DFF_3_OUT
                   );
DFF_2 :
     dff_with_hold_input_ctrl
      PORT MAP (
                    D1  =>DFF_3_OUT,
                    D2  =>D4,
                    clk =>clk,
                    hold=>hold,
                    in_sel=>in_sel,
                    rst =>rst,
                    Q   =>DFF_2_OUT
                   );
DFF_1 :
     dff_with_hold_input_ctrl
      PORT MAP (
                    D1  =>DFF_2_OUT,
                    D2  =>D3,
                    clk =>clk,
                    hold=>hold,
                    in_sel=>in_sel,
                    rst =>rst,
                    Q   =>DFF_1_OUT
                   );
DFF_0 :
     dff_with_hold_input_ctrl
      PORT MAP (
                    D1    =>DFF_1_OUT,
                    D2    =>D2,
                    clk   =>clk,
                    hold  =>hold,
                    in_sel=>in_sel,
```

```
                    rst    =>rst,
                    Q      =>DFF_0_OUT
                  );
  END structural;
```

cb_circuit.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY cb_circuit IS
  PORT  (
          D1                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D2                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D3                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D4                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D5                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D6                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D7                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D8                :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          clk               :      IN     STD_LOGIC;
          hold_seg_0        :      IN     STD_LOGIC;
          hold_seg_1        :      IN     STD_LOGIC;
          hold_seg_2        :      IN     STD_LOGIC;
          hold_seg_3        :      IN     STD_LOGIC;
          hold_seg_4        :      IN     STD_LOGIC;
          hold_seg_5        :      IN     STD_LOGIC;
          hold_seg_6        :      IN     STD_LOGIC;
          hold_seg_7        :      IN     STD_LOGIC;
          in_ctrl_all_seg   :      IN     STD_LOGIC;
          rst               :      IN     STD_LOGIC;
          Q1                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q2                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q3                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q4                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q5                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q6                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q7                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
          Q8                :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END cb_circuit;

ARCHITECTURE structural of cb_circuit IS
COMPONENT dff_segment_for_cb IS
  PORT  (
          D1        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D2        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D3        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D4        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D5        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D6        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D7        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D8        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          D9        :      IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
          clk       :      IN     STD_LOGIC;
          hold      :      IN     STD_LOGIC;
          in_sel    :      IN     STD_LOGIC;
          rst       :      IN     STD_LOGIC;
          Q         :      OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END COMPONENT;

SIGNAL SEGMENT_0_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_1_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_2_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_3_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_4_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_5_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_6_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
```

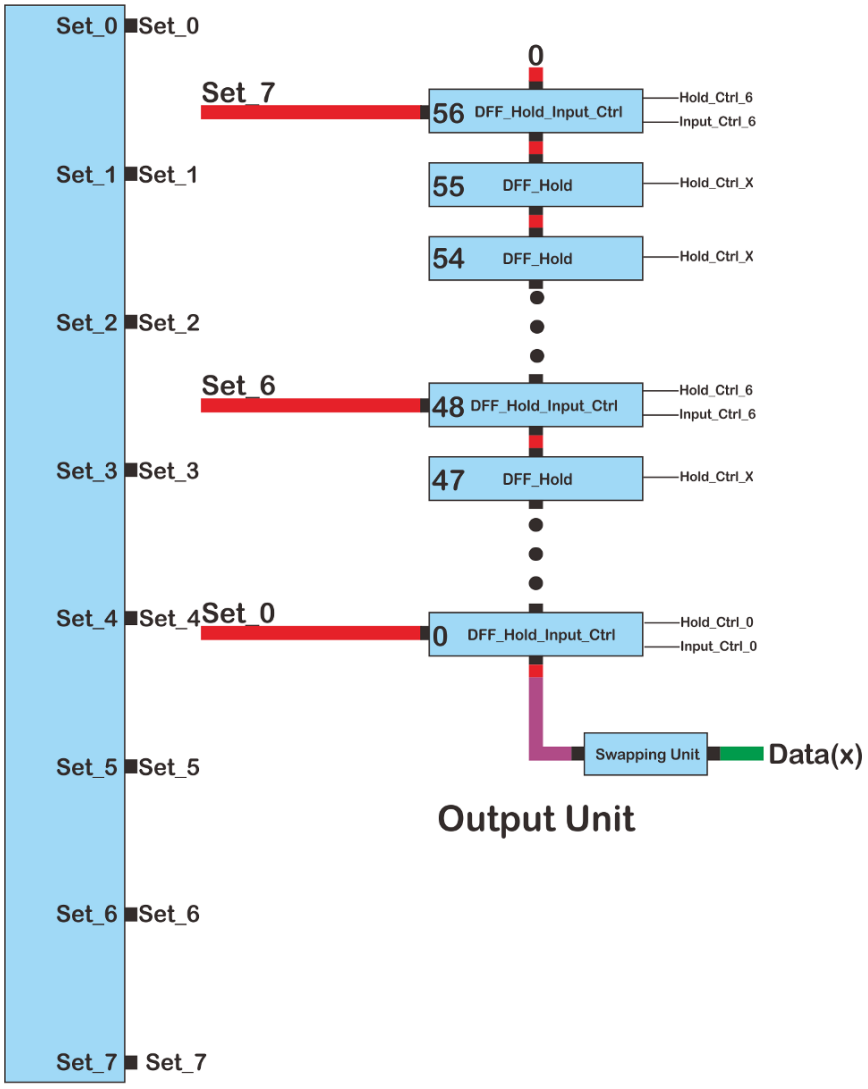
```
SIGNAL SEGMENT_7_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
  BEGIN
  Q1                <=        SEGMENT_0_OUT;
  Q2                <=        SEGMENT_1_OUT;
  Q3                <=        SEGMENT_2_OUT;
  Q4                <=        SEGMENT_3_OUT;
  Q5                <=        SEGMENT_4_OUT;
  Q6                <=        SEGMENT_5_OUT;
  Q7                <=        SEGMENT_6_OUT;
  Q8                <=        SEGMENT_7_OUT;
  SEGMEN_7 :
        dff_segment_for_cb
         PORT MAP (
                        D1              =>std_logic_vector(to_unsigned(0,32)),
                        D2              =>D1,
                        D3              =>D2,
                        D4              =>D3,
                        D5              =>D4,
                        D6              =>D5,
                        D7              =>D6,
                        D8              =>D7,
                        D9              =>D8,
                        clk             =>clk,
                        hold            =>hold_seg_7,
                        in_sel          =>in_ctrl_all_seg,
                        rst             =>rst,
                        Q               =>SEGMENT_7_OUT
                );
  SEGMEN_6 :
        dff_segment_for_cb
         PORT MAP (
                        D1              =>SEGMENT_7_OUT,
                        D2              =>D1,
                        D3              =>D2,
                        D4              =>D3,
                        D5              =>D4,
                        D6              =>D5,
                        D7              =>D6,
                        D8              =>D7,
                        D9              =>D8,
                        clk             =>clk,
                        hold            =>hold_seg_6,
                        in_sel          =>in_ctrl_all_seg,
                        rst             =>rst,
                        Q               =>SEGMENT_6_OUT
                );
  SEGMEN_5 :
        dff_segment_for_cb
         PORT MAP (
                        D1              =>SEGMENT_6_OUT,
                        D2              =>D1,
                        D3              =>D2,
                        D4              =>D3,
                        D5              =>D4,
                        D6              =>D5,
                        D7              =>D6,
                        D8              =>D7,
                        D9              =>D8,
                        clk             =>clk,
                        hold            =>hold_seg_5,
                        in_sel          =>in_ctrl_all_seg,
                        rst             =>rst,
                        Q               =>SEGMENT_5_OUT
                );
  SEGMEN_4 :
        dff_segment_for_cb
         PORT MAP (
                        D1              =>SEGMENT_5_OUT,
```

```
                    D2              =>D1,
                    D3              =>D2,
                    D4              =>D3,
                    D5              =>D4,
                    D6              =>D5,
                    D7              =>D6,
                    D8              =>D7,
                    D9              =>D8,
                    clk             =>clk,
                    hold            =>hold_seg_4,
                    in_sel          =>in_ctrl_all_seg,
                    rst             =>rst,
                    Q               =>SEGMENT_4_OUT
                );
    SEGMEN_3 :
          dff_segment_for_cb
           PORT MAP (
                    D1              =>SEGMENT_4_OUT,
                    D2              =>D1,
                    D3              =>D2,
                    D4              =>D3,
                    D5              =>D4,
                    D6              =>D5,
                    D7              =>D6,
                    D8              =>D7,
                    D9              =>D8,
                    clk             =>clk,
                    hold            =>hold_seg_3,
                    in_sel          =>in_ctrl_all_seg,
                    rst             =>rst,
                    Q               =>SEGMENT_3_OUT
                );
    SEGMEN_2 :
          dff_segment_for_cb
           PORT MAP (
                    D1              =>SEGMENT_3_OUT,
                    D2              =>D1,
                    D3              =>D2,
                    D4              =>D3,
                    D5              =>D4,
                    D6              =>D5,
                    D7              =>D6,
                    D8              =>D7,
                    D9              =>D8,
                    clk             =>clk,
                    hold            =>hold_seg_2,
                    in_sel          =>in_ctrl_all_seg,
                    rst             =>rst,
                    Q               =>SEGMENT_2_OUT
                );
    SEGMEN_1 :
          dff_segment_for_cb
           PORT MAP (
                    D1              =>SEGMENT_2_OUT,
                    D2              =>D1,
                    D3              =>D2,
                    D4              =>D3,
                    D5              =>D4,
                    D6              =>D5,
                    D7              =>D6,
                    D8              =>D7,
                    D9              =>D8,
                    clk             =>clk,
                    hold            =>hold_seg_1,
                    in_sel          =>in_ctrl_all_seg,
                    rst             =>rst,
                    Q               =>SEGMENT_1_OUT
                );
```

```
SEGMEN_0 :
      dff_segment_for_cb
       PORT MAP (
                  D1            =>SEGMENT_1_OUT,
                  D2            =>D1,
                  D3            =>D2,
                  D4            =>D3,
                  D5            =>D4,
                  D6            =>D5,
                  D7            =>D6,
                  D8            =>D7,
                  D9            =>D8,
                  clk           =>clk,
                  hold          =>hold_seg_0,
                  in_sel        =>in_ctrl_all_seg,
                  rst           =>rst,
                  Q             =>SEGMENT_0_OUT
              );
END structural;
```

G. Blok Output Buffer (O/P Unit)

Output buffer digunakan untuk menyimpan data yang dikirim oleh FFT 8-point tahap kedua. Terdapat tiga fungsi output buffer yaitu menyimpan data, menyusun data, dan mengubah data paralel menjadi data serial. Berdasarkan fungsinya tersebut, output buffer diimplementasikan sebagai berikut.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Output buffer ini memiliki struktur yang merupakan komplemen dari input buffer. Terdapat tujuh buah segmen yang masing-masing terdiri atas satu buah DFF_Hold_Input_Ctrl register di bagian bawah segmen dan tujuh buah DFF_Hold register. Terdapat pula satu buah DFF_Hold_Input_Ctrl di depan segmen ketujuh. Semua aktivitas Output Unit akan diatur oleh Master Control.

dff_segment_for_output.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dff_segment_for_output IS
  PORT   (
              D7          :   IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D1          :   IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk         :   IN  STD_LOGIC;
              hold        :   IN  STD_LOGIC;
              in_sel      :   IN  STD_LOGIC;
              rst         :   IN  STD_LOGIC;
              Q           :   OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END dff_segment_for_output;

ARCHITECTURE structural of dff_segment_for_output IS
COMPONENT dff_with_hold_32b IS
  PORT   (
              D           :   IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk         :   IN    STD_LOGIC;
              hold        :   IN    STD_LOGIC;
              rst         :   IN    STD_LOGIC;
              Q           :   OUT   STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
COMPONENT dff_with_hold_input_ctrl IS
  PORT   (
              D1          :   IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
              D2          :   IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk         :   IN    STD_LOGIC;
              hold        :   IN    STD_LOGIC;
              in_sel      :   IN    STD_LOGIC;
              rst         :   IN    STD_LOGIC;
              Q           :   OUT   STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
SIGNAL DFF_0_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_1_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_2_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_3_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_4_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_5_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_6_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DFF_7_OUT:    STD_LOGIC_VECTOR (31 DOWNTO 0);
  BEGIN
  Q      <=  DFF_0_OUT;
  -- Port Map
  DFF_7 :
        dff_with_hold_32b
          PORT MAP (
                      D   =>D7,
                      clk =>clk,
                      hold=>hold,
                      rst =>rst,
                      Q   =>DFF_7_OUT
                    );
  DFF_6 :
        dff_with_hold_32b
          PORT MAP (
                      D   =>DFF_7_OUT,
```

```
                        clk =>clk,
                        hold=>hold,
                        rst =>rst,
                        Q   =>DFF_6_OUT
                    );
    DFF_5 :
           dff_with_hold_32b
            PORT MAP (
                        D   =>DFF_6_OUT,
                        clk =>clk,
                        hold=>hold,
                        rst =>rst,
                        Q   =>DFF_5_OUT
                    );
    DFF_4 :
           dff_with_hold_32b
            PORT MAP (
                        D   =>DFF_5_OUT,
                        clk =>clk,
                        hold=>hold,
                        rst =>rst,
                        Q   =>DFF_4_OUT
                    );
    DFF_3 :
           dff_with_hold_32b
            PORT MAP (
                        D   =>DFF_4_OUT,
                        clk =>clk,
                        hold=>hold,
                        rst =>rst,
                        Q   =>DFF_3_OUT
                    );
    DFF_2 :
           dff_with_hold_32b
            PORT MAP (
                        D   =>DFF_3_OUT,
                        clk =>clk,
                        hold=>hold,
                        rst =>rst,
                        Q   =>DFF_2_OUT
                    );
    DFF_1 :
           dff_with_hold_32b
            PORT MAP (
                        D   =>DFF_2_OUT,
                        clk =>clk,
                        hold=>hold,
                        rst =>rst,
                        Q   =>DFF_1_OUT
                    );
    DFF_0 :
           dff_with_hold_input_ctrl
            PORT MAP (
                        D1    =>DFF_1_OUT,
                        D2    =>D1,
                        clk   =>clk,
                        hold  =>hold,
                        in_sel=>in_sel,
                        rst   =>rst,
                        Q     =>DFF_0_OUT
                    );

    END structural;
```

output_circuit.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
ENTITY output_circuit IS
  PORT   (
            D1                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D2                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D3                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D4                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D5                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D6                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D7                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D8                  :  IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk                 :  IN     STD_LOGIC;
            hold_all_seg        :  IN     STD_LOGIC;
            in_ctrl_all_seg     : IN      STD_LOGIC;
            mode                : IN      STD_LOGIC;
            rst                 :  IN     STD_LOGIC;
            Q                   :  OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END output_circuit;

ARCHITECTURE structural of output_circuit IS
COMPONENT dff_segment_for_output IS
  PORT   (
            D7          : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D1          : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk         : IN     STD_LOGIC;
            hold        : IN     STD_LOGIC;
            in_sel      : IN     STD_LOGIC;
            rst         : IN     STD_LOGIC;
            Q           : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;

COMPONENT dff_with_hold_input_ctrl IS
  PORT   (
            D1          : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            D2          : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
            clk         : IN     STD_LOGIC;
            hold        : IN     STD_LOGIC;
            in_sel      : IN     STD_LOGIC;
            rst         : IN     STD_LOGIC;
            Q           : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
COMPONENT real_imaginary_interchange IS
  PORT   (
            A32         : IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
            Swap        : IN  STD_LOGIC;
            R32         : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;

SIGNAL SWAP_OUT         :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL LEAD_BUF_OUT     :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_0_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_1_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_2_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_3_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_4_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_5_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_6_OUT    :   STD_LOGIC_VECTOR (31 DOWNTO 0);
  BEGIN
  Q                 <=SWAP_OUT;
  -- Port Map
  LEAD_BUF :
         dff_with_hold_input_ctrl
           PORT MAP (
                        D1     =>std_logic_vector(to_unsigned(0,32)),
                        D2     =>D8,
```

```
                        clk   =>clk,
                        hold  =>hold_all_seg,
                        in_sel=>in_ctrl_all_seg,
                        rst   =>rst,
                        Q     =>LEAD_BUF_OUT
                     );
SEGMEN_6 :
       dff_segment_for_output
          PORT MAP (
                        D7       =>LEAD_BUF_OUT,
                        D1       =>D7,
                        clk      =>clk,
                        hold     =>hold_all_seg,
                        in_sel   =>in_ctrl_all_seg,
                        rst      =>rst,
                        Q        =>SEGMENT_6_OUT
                     );
SEGMEN_5 :
       dff_segment_for_output
          PORT MAP (
                        D7       =>SEGMENT_6_OUT,
                        D1       =>D6,
                        clk      =>clk,
                        hold     =>hold_all_seg,
                        in_sel   =>in_ctrl_all_seg,
                        rst      =>rst,
                        Q        =>SEGMENT_5_OUT
                     );
SEGMEN_4 :
       dff_segment_for_output
          PORT MAP (
                        D7       =>SEGMENT_5_OUT,
                        D1       =>D5,
                        clk      =>clk,
                        hold     =>hold_all_seg,
                        in_sel   =>in_ctrl_all_seg,
                        rst      =>rst,
                        Q        =>SEGMENT_4_OUT
                     );
SEGMEN_3 :
       dff_segment_for_output
          PORT MAP (
                        D7       =>SEGMENT_4_OUT,
                        D1       =>D4,
                        clk      =>clk,
                        hold     =>hold_all_seg,
                        in_sel   =>in_ctrl_all_seg,
                        rst      =>rst,
                        Q        =>SEGMENT_3_OUT
                     );
SEGMEN_2 :
       dff_segment_for_output
          PORT MAP (
                        D7       =>SEGMENT_3_OUT,
                        D1       =>D3,
                        clk      =>clk,
                        hold     =>hold_all_seg,
                        in_sel   =>in_ctrl_all_seg,
                        rst      =>rst,
                        Q        =>SEGMENT_2_OUT
                     );
SEGMEN_1 :
       dff_segment_for_output
          PORT MAP (
                        D7       =>SEGMENT_2_OUT,
                        D1       =>D2,
                        clk      =>clk,
                        hold     =>hold_all_seg,
```

```
                        in_sel    =>in_ctrl_all_seg,
                        rst       =>rst,
                        Q         =>SEGMENT_1_OUT
                      );
  SEGMEN_0 :
        dff_segment_for_output
           PORT MAP (
                        D7        =>SEGMENT_1_OUT,
                        D1        =>D1,
                        clk       =>clk,
                        hold      =>hold_all_seg,
                        in_sel    =>in_ctrl_all_seg,
                        rst       =>rst,
                        Q         =>SEGMENT_0_OUT
                      );
    SWAP :
        real_imaginary_interchange
           PORT MAP (
                        A32       =>SEGMENT_0_OUT,
                        Swap      =>mode,
                        R32       =>SWAP_OUT
                      );
  END structural;
```

H. Blok Master Control

Blok Master Control bertanggung jawab mengatur jalannya rangkaian mulai dari mengatur Input Buffer, mengatur perkalian interdimensional twiddle factor, mengatur internal register, hingga mengatur Output Buffer. Oleh karena itu, blok master control merupakan rangkaian yang sangat kompleks baik dari sisi struktur maupun sisi timing. Blok master control harus dapat melakukan sinkronisasi pada seluruh bagian rangkaian sehingga pemindahan data antarblok rangkaian dapat dilakukan dengan baik.

Dalam melakukan tugasnya, blok master control dibantu oleh dua buah counter 6-bit yaitu input counter dan output counter.

- Input counter bertugas untuk menghitung jumlah data serial yang masuk pada Input Buffer. Input Counter diaktifkan dengan menggunakan sinyal datastart dari luar rangkaian yang menandakan bahwa terdapat stream data yang valid untuk diproses. Sinyal datastart akan aktif hingga 64 clock cycle selanjutnya. Counter akan menghitung hingga 64 kali sesuai jumlah data yang dimasukkan dan akan otomatis kembali ke nol dan berhenti berhitung. Pada saat register ke-57 (register 56) dari input buffer akan terisi data, Input counter akan mengaktifkan sinyal trigger untuk membangunkan master control. Selanjutnya master control mengambil alih kontrol rangkaian.
- Output counter bertugas untuk menghitung jumlah data serial yang keluar dari Output Buffer. Output counter diaktifkan oleh Master Control ketika output data pertama tersedia. Selanjutnya Output counter akan mengaktifkan sinyal datavalid yang akan aktif hingga 64 clock cycle selanjutnya yang menandakan bahwa data output valid sedang dikeluarkan dari output rangkaian. Sama seperti input counter, output counter akan otomatis kembali ke nol dan berhenti berhitung ketika data output terakhir telah dikeluarkan.

Berikut ini adalah implementasi input counter dan output counter dalam kode VHDL.

input_counter.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_unsigned.all;

ENTITY input_counter IS
```

```
   PORT   (
               clk          :      IN  STD_LOGIC;
               rst          :      IN  STD_LOGIC;
               datastart    :      IN  STD_LOGIC;
               counter_o    :      OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
               mastertrig   :      OUT STD_LOGIC
          );
END input_counter;

ARCHITECTURE behavioral OF input_counter IS
TYPE FSMSTATE IS (
         idle,
         counting
         );

SIGNAL currentstate        : FSMSTATE;
SIGNAL counter             : STD_LOGIC_VECTOR (5 DOWNTO 0);
   BEGIN
    counter_o <= counter;
   PROCESS(clk,rst,currentstate,datastart)
   BEGIN
         IF rst='1' THEN
          currentstate <= idle;
         ELSE
          IF ((clk'EVENT) AND (clk='1'))  THEN
              CASE currentstate IS
                        WHEN idle                    =>
                        IF (datastart='1') THEN
                              currentstate <= counting;
                              counter      <= "000000";
                              mastertrig   <= '0';
                        ELSE
                              currentstate <= currentstate;
                              counter      <= "000000";
                              mastertrig   <= '0';
                        END IF;
                        WHEN counting         =>
                        IF (counter="110101") THEN
                              currentstate <= counting;
                              counter       <= counter + '1';
                              mastertrig   <= '1';
                        ELSIF (counter="111110") THEN
                              currentstate <= idle;
                              counter      <= counter + '1';
                              mastertrig   <= '0';
                        ELSE
                              currentstate <= counting;
                              counter      <= counter + '1';
                              mastertrig   <= '0';
                        END IF;
              END CASE;
          ELSE
              currentstate <= currentstate;
          END IF;
         END IF;
   END PROCESS;
END behavioral;
```

output_counter.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_unsigned.all;

ENTITY output_counter IS
  PORT  (
            Clk          : IN      STD_LOGIC;
            rst          : IN      STD_LOGIC;
            dataind      : IN      STD_LOGIC;
```

```
            counter_o    : OUT      STD_LOGIC_VECTOR (5 DOWNTO 0);
            datavalid    : OUT      STD_LOGIC
        );
END output_counter;

ARCHITECTURE behavioral OF output_counter IS
TYPE FSMSTATE IS (
  idle,
  counting
  );

SIGNAL currentstate     : FSMSTATE;
SIGNAL counter           : STD_LOGIC_VECTOR (5 DOWNTO 0);
  BEGIN
    counter_o <= counter;
  PROCESS(clk,rst,currentstate,dataind)
  BEGIN
       IF rst='1' THEN
         currentstate <= idle;
       ELSE
         IF ((clk'EVENT) AND (clk='1'))  THEN
             CASE currentstate IS
                        WHEN idle                       =>
                          IF (dataind='1') THEN
                            currentstate <= counting;
                            counter              <= "000000";
                            datavalid    <= '1';
                          ELSE
                            currentstate <= currentstate;
                            counter              <= "000000";
                            datavalid    <= '0';
                          END IF;
                        WHEN counting           =>
                          IF (counter="111110") THEN
                            currentstate <= idle;
                            counter              <= counter + '1';
                            datavalid    <= '1';
                          ELSE
                            currentstate <= counting;
                            counter              <= counter + '1';
                            datavalid    <= '1';
                          END IF;
             END CASE;
         ELSE
             currentstate <= currentstate;
         END IF;
       END IF;
  END PROCESS;
END behavioral;
```

Master Control memiliki 23 state yang merepresentasikan 23 cycle pemrosesan data yang akan dilakukan oleh FFT/IFFT 64-titik ini. Dengan asumsi input buffer dan output buffer yang telah penuh, operasi FFT/IFFT 64-titik dapat dilakukan selama 32 clock cycle. Berikut ini adalah tabel state beserta sinyal kontrol yang berasosiasi dengan state tersebut.

State	Information	Input Block						Interdimensional Complex Multiplier Block												CB Block		Output Block	
State 0	Wait for Input Counter to reach 56 and activate the master control	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	1		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 1	Input Counter send enable signal after 56 input data reached. Performing block data set calculation for first stage FFT (T[0]) B(0), B(8), B(16), B(24), B(32), B(40), B(48), B(56)	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	0	DFF_Ctrl_0	0	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	0	DeShuf_Ctrl_1	x	Bypass_EN_1	1	Hold_Ctrl_1	0	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	0	DeShuf_Ctrl_2	x	Bypass_EN_2	1	Hold_Ctrl_2	0	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	0	DeShuf_Ctrl_3	x	Bypass_EN_3	1	Hold_Ctrl_3	0	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	0	DeShuf_Ctrl_4	x	Bypass_EN_4	1	Hold_Ctrl_4	0	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	0	DeShuf_Ctrl_5	x	Bypass_EN_5	1	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	0	DeShuf_Ctrl_6	x	Bypass_EN_6	1	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	0	DeShuf_Ctrl_7	x	Bypass_EN_7	1	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 2	Performing block data set calculation for first stage FFT (T[1]) B(1), B(9), B(17), B(25), B(33), B(41), B(49), B(57)	hold_all_seg	0	Shuf_Ctrl_1	1	Type_Sel_1	1	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	x	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	2	Type_Sel_2	1	Bypass_Sel_1	x	DeShuf_Ctrl_1	0	Bypass_EN_1	0	Hold_Ctrl_1	x	DFF_Ctrl_1	0	hold_seg_1	0	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	3	Type_Sel_3	1	Bypass_Sel_2	x	DeShuf_Ctrl_2	1	Bypass_EN_2	0	Hold_Ctrl_2	x	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	4	Type_Sel_4	1	Bypass_Sel_3	x	DeShuf_Ctrl_3	2	Bypass_EN_3	0	Hold_Ctrl_3	x	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	5	Type_Sel_5	1	Bypass_Sel_4	x	DeShuf_Ctrl_4	3	Bypass_EN_4	0	Hold_Ctrl_4	x	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	6	Type_Sel_6	1	Bypass_Sel_5	x	DeShuf_Ctrl_5	4	Bypass_EN_5	0	Hold_Ctrl_5	x	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	7	Type_Sel_7	1	Bypass_Sel_6	x	DeShuf_Ctrl_6	5	Bypass_EN_6	0	Hold_Ctrl_6	x	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	1	Bypass_Sel_7	x	DeShuf_Ctrl_7	6	Bypass_EN_7	0	Hold_Ctrl_7	x	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 3	Performing block data set calculation for first stage FFT (T[2]) B(2), B(10), B(18), B(26), B(34), B(42), B(50), B(58) Cycle 1 of 2	hold_all_seg	1	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	0	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	1	Type_Sel_2	1	Bypass_Sel_1	x	DeShuf_Ctrl_1	1	Bypass_EN_1	0	Hold_Ctrl_1	0	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	3	Bypass_EN_2	0	Hold_Ctrl_2	0	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	0	Shuf_Ctrl_4	2	Type_Sel_4	1	Bypass_Sel_3	x	DeShuf_Ctrl_3	5	Bypass_EN_3	0	Hold_Ctrl_3	0	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	7	Bypass_EN_4	0	Hold_Ctrl_4	0	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	3	Type_Sel_6	1	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	1	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	4	Type_Sel_8	1	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 4		hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	1	DFF_Ctrl_0	1	hold_seg_0	1	hold_all_seg	0

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

	Performing block data set calculation for first stage FFT (T[2]) B(2), B(10), B(18), B(26), B(34), B(42), B(50), B(58) Cycle 2 of 2	hold_buf_3	1	Shuf_Ctrl_2	7	Type_Sel_2	5	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	0	Hold_Ctrl_1	1	DFF_Ctrl_1	1	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	0	Hold_Ctrl_2	1	DFF_Ctrl_2	1	hold_seg_2	0	counter_en	0
		hold_buf_1	0	Shuf_Ctrl_4	6	Type_Sel_4	5	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	0	Hold_Ctrl_3	1	DFF_Ctrl_3	1	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	0	Hold_Ctrl_4	1	DFF_Ctrl_4	1	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	5	Type_Sel_6	5	Bypass_Sel_5	x	DeShuf_Ctrl_5	5	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	1	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	3	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	1	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 5	Performing block data set calculation for first stage FFT (T[3]) B(3), B(11), B(19), B(27), B(35), B(43), B(51), B(59)	hold_all_seg	0	Shuf_Ctrl_1	5	Type_Sel_1	5	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	x	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	6	Type_Sel_2	7	Bypass_Sel_1	x	DeShuf_Ctrl_1	2	Bypass_EN_1	0	Hold_Ctrl_1	x	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	1	Type_Sel_3	1	Bypass_Sel_2	x	DeShuf_Ctrl_2	5	Bypass_EN_2	0	Hold_Ctrl_2	x	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	0	Shuf_Ctrl_4	4	Type_Sel_4	5	Bypass_Sel_3	x	DeShuf_Ctrl_3	6	Bypass_EN_3	0	Hold_Ctrl_3	x	DFF_Ctrl_3	0	hold_seg_3	0	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	7	Type_Sel_5	7	Bypass_Sel_4	x	DeShuf_Ctrl_4	3	Bypass_EN_4	0	Hold_Ctrl_4	x	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	2	Type_Sel_6	1	Bypass_Sel_5	x	DeShuf_Ctrl_5	0	Bypass_EN_5	0	Hold_Ctrl_5	x	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	1	Shuf_Ctrl_7	3	Type_Sel_7	5	Bypass_Sel_6	x	DeShuf_Ctrl_6	1	Bypass_EN_6	0	Hold_Ctrl_6	x	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	4	Bypass_EN_7	0	Hold_Ctrl_7	x	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 6	Performing block data set calculation for first stage FFT (T[4]) B(4), B(12), B(20), B(28), B(36), B(44), B(52), B(60) Cycle 1 of 4	hold_all_seg	1	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	0	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	3	Bypass_EN_1	0	Hold_Ctrl_1	0	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	0	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	7	Bypass_EN_2	0	Hold_Ctrl_2	0	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	1	Type_Sel_4	1	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	0	Hold_Ctrl_3	0	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	5	DeShuf_Ctrl_4	x	Bypass_EN_4	1	Hold_Ctrl_4	0	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	1	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	2	Type_Sel_8	1	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 7	Performing block data set calculation for first stage FFT (T[4]) B(4), B(12), B(20), B(28), B(36), B(44), B(52), B(60) Cycle 2 of 4	hold_all_seg	1	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	1	DFF_Ctrl_0	1	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	0	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	0	Hold_Ctrl_1	1	DFF_Ctrl_1	1	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	0	Hold_Ctrl_2	1	DFF_Ctrl_2	1	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	3	Type_Sel_4	5	Bypass_Sel_3	x	DeShuf_Ctrl_3	3	Bypass_EN_3	0	Hold_Ctrl_3	0	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	1	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	0	Hold_Ctrl_4	1	DFF_Ctrl_4	1	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 8	Performing block data set	hold_all_seg	1	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	1	DFF_Ctrl_0	1	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	0	Hold_Ctrl_1	1	DFF_Ctrl_1	1	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	0	Hold_Ctrl_2	1	DFF_Ctrl_2	1	hold_seg_2	1	counter_en	0

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

	calculation for first stage FFT (T[4]) B(4), B(12), B(20), B(28), B(36), B(44), B(52), B(60) Cycle 3 of 4	hold_buf_1	1	Shuf_Ctrl_4	5	Type_Sel_4	7	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	0	Hold_Ctrl_3	1	DFF_Ctrl_3	1	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	0	Hold_Ctrl_4	1	DFF_Ctrl_4	1	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	3	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	7	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	6	Type_Sel_8	7	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 9	Performing block data set calculation for first stage FFT (T[4]) B(4), B(12), B(20), B(28), B(36), B(44), B(52), B(60) Cycle 4 of 4	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	1	DFF_Ctrl_0	1	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	0	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	0	Hold_Ctrl_1	1	DFF_Ctrl_1	1	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	0	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	0	Hold_Ctrl_2	1	DFF_Ctrl_2	1	hold_seg_2	1	counter_en	0
		hold_buf_1	0	Shuf_Ctrl_4	7	Type_Sel_4	3	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	0	Hold_Ctrl_3	1	DFF_Ctrl_3	1	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	0	Hold_Ctrl_4	1	DFF_Ctrl_4	1	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	0	Hold_Ctrl_5	1	DFF_Ctrl_5	1	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	0	Hold_Ctrl_6	1	DFF_Ctrl_6	1	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	3	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 10	Performing block data set calculation for first stage FFT (T[5]) B(5), B(13), B(21), B(29), B(37), B(45), B(53), B(61)	hold_all_seg	0	Shuf_Ctrl_1	3	Type_Sel_1	5	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	x	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	0	Shuf_Ctrl_2	6	Type_Sel_2	3	Bypass_Sel_1	x	DeShuf_Ctrl_1	4	Bypass_EN_1	0	Hold_Ctrl_1	x	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	0	Shuf_Ctrl_3	7	Type_Sel_3	2	Bypass_Sel_2	x	DeShuf_Ctrl_2	5	Bypass_EN_2	0	Hold_Ctrl_2	x	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	0	Shuf_Ctrl_4	4	Type_Sel_4	7	Bypass_Sel_3	x	DeShuf_Ctrl_3	0	Bypass_EN_3	0	Hold_Ctrl_3	x	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	1	Type_Sel_5	1	Bypass_Sel_4	x	DeShuf_Ctrl_4	3	Bypass_EN_4	0	Hold_Ctrl_4	x	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	2	Type_Sel_6	5	Bypass_Sel_5	x	DeShuf_Ctrl_5	6	Bypass_EN_5	0	Hold_Ctrl_5	x	DFF_Ctrl_5	0	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	5	Type_Sel_7	3	Bypass_Sel_6	x	DeShuf_Ctrl_6	1	Bypass_EN_6	0	Hold_Ctrl_6	x	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	2	Bypass_EN_7	0	Hold_Ctrl_7	x	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 11	Performing block data set calculation for first stage FFT (T[6]) B(6), B(14), B(22), B(30), B(38), B(46), B(54), B(62) Cycle 1 of 2	hold_all_seg	1	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	0	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	3	Type_Sel_2	7	Bypass_Sel_1	x	DeShuf_Ctrl_1	5	Bypass_EN_1	0	Hold_Ctrl_1	0	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	3	Bypass_EN_2	0	Hold_Ctrl_2	0	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	2	Type_Sel_4	5	Bypass_Sel_3	x	DeShuf_Ctrl_3	1	Bypass_EN_3	0	Hold_Ctrl_3	0	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	7	Bypass_EN_4	0	Hold_Ctrl_4	0	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	1	Type_Sel_6	1	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	1	Shuf_Ctrl_8	4	Type_Sel_8	7	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 12	Performing block data set calculation for first stage	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	1	DFF_Ctrl_0	1	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	0	Shuf_Ctrl_2	5	Type_Sel_2	3	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	0	Hold_Ctrl_1	1	DFF_Ctrl_1	1	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	0	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	0	Hold_Ctrl_2	1	DFF_Ctrl_2	1	hold_seg_2	1	counter_en	0
		hold_buf_1	0	Shuf_Ctrl_4	6	Type_Sel_4	2	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	0	Hold_Ctrl_3	1	DFF_Ctrl_3	1	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	0	Hold_Ctrl_4	1	DFF_Ctrl_4	1	hold_seg_4	1		

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

	FFT (T[6]) B(6), B(14), B(22), B(30), B(38), B(46), B(54), B(62) Cycle 2 of 2	in_ctrl_buf_2	0	Shuf_Ctrl_6	7	Type_Sel_6	6	Bypass_Sel_5	x	DeShuf_Ctrl_5	1	Bypass_EN_5	0	Hold_Ctrl_5	0	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	3	Bypass_EN_6	0	Hold_Ctrl_6	0	DFF_Ctrl_6	0	hold_seg_6	0		
		pos_hold_ctrl	1	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	5	Bypass_EN_7	0	Hold_Ctrl_7	0	DFF_Ctrl_7	0	hold_seg_7	1		
																		in_ctrl_all_seg	1		
State 13	Performing block data set calculation for first stage FFT (T[7]) B(7), B(15), B(23), B(31), B(39), B(47), B(55), B(63)	hold_all_seg	0	Shuf_Ctrl_1	7	Type_Sel_1	4	Bypass_Sel_0	0	DeShuf_Ctrl_0	x	Bypass_EN_0	1	Hold_Ctrl_0	x	DFF_Ctrl_0	0	hold_seg_0	1	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	2	Type_Sel_2	5	Bypass_Sel_1	x	DeShuf_Ctrl_1	6	Bypass_EN_1	0	Hold_Ctrl_1	x	DFF_Ctrl_1	0	hold_seg_1	1	in_ctrl_all_seg	0
		hold_buf_2	1	Shuf_Ctrl_3	5	Type_Sel_3	2	Bypass_Sel_2	x	DeShuf_Ctrl_2	1	Bypass_EN_2	0	Hold_Ctrl_2	x	DFF_Ctrl_2	0	hold_seg_2	1	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	4	Type_Sel_4	3	Bypass_Sel_3	x	DeShuf_Ctrl_3	4	Bypass_EN_3	0	Hold_Ctrl_3	x	DFF_Ctrl_3	0	hold_seg_3	1	data_valid	0
		in_ctrl_buf_3	0	Shuf_Ctrl_5	3	Type_Sel_5	7	Bypass_Sel_4	x	DeShuf_Ctrl_4	3	Bypass_EN_4	0	Hold_Ctrl_4	x	DFF_Ctrl_4	0	hold_seg_4	1		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	6	Type_Sel_6	6	Bypass_Sel_5	x	DeShuf_Ctrl_5	2	Bypass_EN_5	0	Hold_Ctrl_5	x	DFF_Ctrl_5	0	hold_seg_5	1		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	1	Type_Sel_7	1	Bypass_Sel_6	x	DeShuf_Ctrl_6	5	Bypass_EN_6	0	Hold_Ctrl_6	x	DFF_Ctrl_6	0	hold_seg_6	1		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	0	Bypass_EN_7	0	Hold_Ctrl_7	x	DFF_Ctrl_7	0	hold_seg_7	0		
																		in_ctrl_all_seg	1		
State 14	Performing block data set calculation for second stage FFT (T[0])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	1
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 15	Performing block data set calculation for second stage FFT (T[1])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 16	Performing block data set calculation for second stage FFT (T[2])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 17	Performing block data	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

	set calculation for second stage FFT (T[3])	hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 18	Performing block data set calculation for second stage FFT (T[4])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 19	Performing block data set calculation for second stage FFT (T[5])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 20	Performing block data set calculation for second stage FFT (T[6])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 21	Performing block data set calculation for second stage FFT (T[7])	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	1
		hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	0
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		
State 22	Finalizing Output Stage	hold_all_seg	0	Shuf_Ctrl_1	x	Type_Sel_1	x	Bypass_Sel_0	x	DeShuf_Ctrl_0	x	Bypass_EN_0	x	Hold_Ctrl_0	x	DFF_Ctrl_0	x	hold_seg_0	0	hold_all_seg	0
		hold_buf_3	1	Shuf_Ctrl_2	x	Type_Sel_2	x	Bypass_Sel_1	x	DeShuf_Ctrl_1	x	Bypass_EN_1	x	Hold_Ctrl_1	x	DFF_Ctrl_1	x	hold_seg_1	0	in_ctrl_all_seg	0

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

	Handover control to output counter until it is count 56	hold_buf_2	1	Shuf_Ctrl_3	x	Type_Sel_3	x	Bypass_Sel_2	x	DeShuf_Ctrl_2	x	Bypass_EN_2	x	Hold_Ctrl_2	x	DFF_Ctrl_2	x	hold_seg_2	0	counter_en	1
		hold_buf_1	1	Shuf_Ctrl_4	x	Type_Sel_4	x	Bypass_Sel_3	x	DeShuf_Ctrl_3	x	Bypass_EN_3	x	Hold_Ctrl_3	x	DFF_Ctrl_3	x	hold_seg_3	0	data_valid	1
		in_ctrl_buf_3	0	Shuf_Ctrl_5	x	Type_Sel_5	x	Bypass_Sel_4	x	DeShuf_Ctrl_4	x	Bypass_EN_4	x	Hold_Ctrl_4	x	DFF_Ctrl_4	x	hold_seg_4	0		
		in_ctrl_buf_2	0	Shuf_Ctrl_6	x	Type_Sel_6	x	Bypass_Sel_5	x	DeShuf_Ctrl_5	x	Bypass_EN_5	x	Hold_Ctrl_5	x	DFF_Ctrl_5	x	hold_seg_5	0		
		in_ctrl_buf_1	0	Shuf_Ctrl_7	x	Type_Sel_7	x	Bypass_Sel_6	x	DeShuf_Ctrl_6	x	Bypass_EN_6	x	Hold_Ctrl_6	x	DFF_Ctrl_6	x	hold_seg_6	0		
		pos_hold_ctrl	0	Shuf_Ctrl_8	x	Type_Sel_8	x	Bypass_Sel_7	x	DeShuf_Ctrl_7	x	Bypass_EN_7	x	Hold_Ctrl_7	x	DFF_Ctrl_7	x	hold_seg_7	0		
																		in_ctrl_all_seg	0		

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

Berikut ini adalah kode VHDL untuk mengimplementasikan Master Control.

master_control.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_unsigned.all;

ENTITY master_control IS
	PORT	(
			clk			: IN	STD_LOGIC;
			rst			: IN	STD_LOGIC;
			mastertrig		: IN	STD_LOGIC;
			hold_all_in		: OUT	STD_LOGIC;
			hold_buf_3		: OUT	STD_LOGIC;
			hold_buf_2		: OUT	STD_LOGIC;
			hold_buf_1		: OUT	STD_LOGIC;
			in_ctrl_buf_3		: OUT	STD_LOGIC;
			in_ctrl_buf_2		: OUT	STD_LOGIC;
			in_ctrl_buf_1		: OUT	STD_LOGIC;
			pos_hold_ctrl		: OUT	STD_LOGIC;
			Shuf_Ctrl_1		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_2		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_3		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_4		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_5		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_6		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_7		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Shuf_Ctrl_8		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_1		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_2		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_3		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_4		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_5		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_6		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_7		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Type_Sel_8		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_0		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_1		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_2		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_3		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_4		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_5		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_6		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_Sel_7		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_0		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_1		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_2		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_3		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_4		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_5		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_6		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			DeShuf_Ctrl_7		: OUT	STD_LOGIC_VECTOR(2 DOWNTO 0);
			Bypass_EN_0		: OUT	STD_LOGIC;
			Bypass_EN_1		: OUT	STD_LOGIC;
			Bypass_EN_2		: OUT	STD_LOGIC;
			Bypass_EN_3		: OUT	STD_LOGIC;
			Bypass_EN_4		: OUT	STD_LOGIC;
			Bypass_EN_5		: OUT	STD_LOGIC;
			Bypass_EN_6		: OUT	STD_LOGIC;
			Bypass_EN_7		: OUT	STD_LOGIC;
			Hold_Ctrl_0		: OUT	STD_LOGIC;
			Hold_Ctrl_1		: OUT	STD_LOGIC;
			Hold_Ctrl_2		: OUT	STD_LOGIC;
			Hold_Ctrl_3		: OUT	STD_LOGIC;
			Hold_Ctrl_4		: OUT	STD_LOGIC;
			Hold_Ctrl_5		: OUT	STD_LOGIC;
			Hold_Ctrl_6		: OUT	STD_LOGIC;
```

```
                Hold_Ctrl_7       : OUT STD_LOGIC;
                DFF_Ctrl_0        : OUT STD_LOGIC;
                DFF_Ctrl_1        : OUT STD_LOGIC;
                DFF_Ctrl_2        : OUT STD_LOGIC;
                DFF_Ctrl_3        : OUT STD_LOGIC;
                DFF_Ctrl_4        : OUT STD_LOGIC;
                DFF_Ctrl_5        : OUT STD_LOGIC;
                DFF_Ctrl_6        : OUT STD_LOGIC;
                DFF_Ctrl_7        : OUT STD_LOGIC;
                hold_seg_0        : OUT STD_LOGIC;
                hold_seg_1        : OUT STD_LOGIC;
                hold_seg_2        : OUT STD_LOGIC;
                hold_seg_3        : OUT STD_LOGIC;
                hold_seg_4        : OUT STD_LOGIC;
                hold_seg_5        : OUT STD_LOGIC;
                hold_seg_6        : OUT STD_LOGIC;
                hold_seg_7        : OUT STD_LOGIC;
                in_ctrl_all_cb    : OUT STD_LOGIC;
                hold_all_out      : OUT STD_LOGIC;
                in_ctrl_all_out   : OUT STD_LOGIC;
                counter_en        : OUT STD_LOGIC
                    );
END master_control;

ARCHITECTURE behavioral OF master_control IS
TYPE FSMSTATE IS (
       state_0, state_1, state_2, state_3, state_4, state_5, state_6,
       state_7, state_8, state_9, state_10, state_11, state_12, state_13,
       state_14, state_15, state_16, state_17, state_18, state_19, state_20,
       state_21, state_22
       );

SIGNAL currentstate : FSMSTATE;
       BEGIN
       PROCESS(clk,rst,currentstate,mastertrig)
       BEGIN
              IF rst='1' THEN
                     currentstate <= state_0;
              ELSE
                     IF ((clk'EVENT) AND (clk='1'))  THEN
                           CASE currentstate IS
                     WHEN state_0          =>
                         hold_all_in     <=  '0';
                         hold_buf_3      <=  '1';
                         hold_buf_2      <=  '1';
                         hold_buf_1      <=  '1';
                         in_ctrl_buf_3   <=  '0';
                         in_ctrl_buf_2   <=  '0';
                         in_ctrl_buf_1   <=  '0';
                         pos_hold_ctrl   <=  '0';
                         Shuf_Ctrl_1     <=  "000";
                         Shuf_Ctrl_2     <=  "000";
                         Shuf_Ctrl_3     <=  "000";
                         Shuf_Ctrl_4     <=  "000";
                         Shuf_Ctrl_5     <=  "000";
                         Shuf_Ctrl_6     <=  "000";
                         Shuf_Ctrl_7     <=  "000";
                         Shuf_Ctrl_8     <=  "000";
                         Type_Sel_1      <=  "000";
                         Type_Sel_2      <=  "000";
                         Type_Sel_3      <=  "000";
                         Type_Sel_4      <=  "000";
                         Type_Sel_5      <=  "000";
                         Type_Sel_6      <=  "000";
                         Type_Sel_7      <=  "000";
                         Type_Sel_8      <=  "000";
                         Bypass_Sel_0    <=  "000";
                         Bypass_Sel_1    <=  "000";
```

```
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '1';
                    Bypass_EN_2     <=  '1';
                    Bypass_EN_3     <=  '1';
                    Bypass_EN_4     <=  '1';
                    Bypass_EN_5     <=  '1';
                    Bypass_EN_6     <=  '1';
                    Bypass_EN_7     <=  '1';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=   '0';
                    counter_en      <=  '0';
                       IF mastertrig='1' THEN
                              currentstate   <= state_1;
                       ELSE
                              currentstate   <= state_0;
                       END IF;
                WHEN state_1          =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '0';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "000";
```

```
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "000";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '1';
                    Bypass_EN_2     <=  '1';
                    Bypass_EN_3     <=  '1';
                    Bypass_EN_4     <=  '1';
                    Bypass_EN_5     <=  '1';
                    Bypass_EN_6     <=  '1';
                    Bypass_EN_7     <=  '1';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '0';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_2;
                WHEN state_2          =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
```

```
                        hold_buf_2      <=  '1';
                        hold_buf_1      <=  '1';
                        in_ctrl_buf_3   <=  '0';
                        in_ctrl_buf_2   <=  '0';
                        in_ctrl_buf_1   <=  '0';
                        pos_hold_ctrl   <=  '0';
                        Shuf_Ctrl_1     <=  "001";
                        Shuf_Ctrl_2     <=  "010";
                        Shuf_Ctrl_3     <=  "011";
                        Shuf_Ctrl_4     <=  "100";
                        Shuf_Ctrl_5     <=  "101";
                        Shuf_Ctrl_6     <=  "110";
                        Shuf_Ctrl_7     <=  "111";
                        Shuf_Ctrl_8     <=  "000";
                        Type_Sel_1      <=  "001";
                        Type_Sel_2      <=  "001";
                        Type_Sel_3      <=  "001";
                        Type_Sel_4      <=  "001";
                        Type_Sel_5      <=  "001";
                        Type_Sel_6      <=  "001";
                        Type_Sel_7      <=  "001";
                        Type_Sel_8      <=  "001";
                        Bypass_Sel_0    <=  "000";
                        Bypass_Sel_1    <=  "000";
                        Bypass_Sel_2    <=  "000";
                        Bypass_Sel_3    <=  "000";
                        Bypass_Sel_4    <=  "000";
                        Bypass_Sel_5    <=  "000";
                        Bypass_Sel_6    <=  "000";
                        Bypass_Sel_7    <=  "000";
                        DeShuf_Ctrl_0   <=  "000";
                        DeShuf_Ctrl_1   <=  "000";
                        DeShuf_Ctrl_2   <=  "001";
                        DeShuf_Ctrl_3   <=  "010";
                        DeShuf_Ctrl_4   <=  "011";
                        DeShuf_Ctrl_5   <=  "100";
                        DeShuf_Ctrl_6   <=  "101";
                        DeShuf_Ctrl_7   <=  "110";
                        Bypass_EN_0     <=  '1';
                        Bypass_EN_1     <=  '0';
                        Bypass_EN_2     <=  '0';
                        Bypass_EN_3     <=  '0';
                        Bypass_EN_4     <=  '0';
                        Bypass_EN_5     <=  '0';
                        Bypass_EN_6     <=  '0';
                        Bypass_EN_7     <=  '0';
                        Hold_Ctrl_0     <=  '0';
                        Hold_Ctrl_1     <=  '0';
                        Hold_Ctrl_2     <=  '0';
                        Hold_Ctrl_3     <=  '0';
                        Hold_Ctrl_4     <=  '0';
                        Hold_Ctrl_5     <=  '0';
                        Hold_Ctrl_6     <=  '0';
                        Hold_Ctrl_7     <=  '0';
                        DFF_Ctrl_0      <=  '0';
                        DFF_Ctrl_1      <=  '0';
                        DFF_Ctrl_2      <=  '0';
                        DFF_Ctrl_3      <=  '0';
                        DFF_Ctrl_4      <=  '0';
                        DFF_Ctrl_5      <=  '0';
                        DFF_Ctrl_6      <=  '0';
                        DFF_Ctrl_7      <=  '0';
                        hold_seg_0      <=  '1';
                        hold_seg_1      <=  '0';
                        hold_seg_2      <=  '1';
                        hold_seg_3      <=  '1';
                        hold_seg_4      <=  '1';
                        hold_seg_5      <=  '1';
```

```
                hold_seg_6      <=  '1';
                hold_seg_7      <=  '1';
                in_ctrl_all_cb  <=  '1';
                hold_all_out    <=  '0';
                in_ctrl_all_out<=  '0';
                counter_en      <=  '0';
                currentstate    <= state_3;
            WHEN state_3          =>
                hold_all_in     <=  '1';
                hold_buf_3      <=  '1';
                hold_buf_2      <=  '1';
                hold_buf_1      <=  '0';
                in_ctrl_buf_3   <=  '0';
                in_ctrl_buf_2   <=  '0';
                in_ctrl_buf_1   <=  '1';
                pos_hold_ctrl   <=  '1';
                Shuf_Ctrl_1     <=  "000";
                Shuf_Ctrl_2     <=  "001";
                Shuf_Ctrl_3     <=  "000";
                Shuf_Ctrl_4     <=  "010";
                Shuf_Ctrl_5     <=  "000";
                Shuf_Ctrl_6     <=  "011";
                Shuf_Ctrl_7     <=  "000";
                Shuf_Ctrl_8     <=  "100";
                Type_Sel_1      <=  "000";
                Type_Sel_2      <=  "001";
                Type_Sel_3      <=  "000";
                Type_Sel_4      <=  "001";
                Type_Sel_5      <=  "000";
                Type_Sel_6      <=  "001";
                Type_Sel_7      <=  "000";
                Type_Sel_8      <=  "001";
                Bypass_Sel_0    <=  "000";
                Bypass_Sel_1    <=  "000";
                Bypass_Sel_2    <=  "000";
                Bypass_Sel_3    <=  "000";
                Bypass_Sel_4    <=  "000";
                Bypass_Sel_5    <=  "000";
                Bypass_Sel_6    <=  "000";
                Bypass_Sel_7    <=  "000";
                DeShuf_Ctrl_0   <=  "000";
                DeShuf_Ctrl_1   <=  "001";
                DeShuf_Ctrl_2   <=  "011";
                DeShuf_Ctrl_3   <=  "101";
                DeShuf_Ctrl_4   <=  "111";
                DeShuf_Ctrl_5   <=  "000";
                DeShuf_Ctrl_6   <=  "000";
                DeShuf_Ctrl_7   <=  "000";
                Bypass_EN_0     <=  '1';
                Bypass_EN_1     <=  '0';
                Bypass_EN_2     <=  '0';
                Bypass_EN_3     <=  '0';
                Bypass_EN_4     <=  '0';
                Bypass_EN_5     <=  '0';
                Bypass_EN_6     <=  '0';
                Bypass_EN_7     <=  '0';
                Hold_Ctrl_0     <=  '0';
                Hold_Ctrl_1     <=  '0';
                Hold_Ctrl_2     <=  '0';
                Hold_Ctrl_3     <=  '0';
                Hold_Ctrl_4     <=  '0';
                Hold_Ctrl_5     <=  '0';
                Hold_Ctrl_6     <=  '0';
                Hold_Ctrl_7     <=  '0';
                DFF_Ctrl_0      <=  '0';
                DFF_Ctrl_1      <=  '0';
                DFF_Ctrl_2      <=  '0';
                DFF_Ctrl_3      <=  '0';
```

```
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_4;
                WHEN state_4          =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '0';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '1';
                    pos_hold_ctrl   <=  '1';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "111";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "110";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "101";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "101";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "101";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "101";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "101";
                    DeShuf_Ctrl_6   <=  "011";
                    DeShuf_Ctrl_7   <=  "001";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '1';
                    Hold_Ctrl_1     <=  '1';
```

```
                    Hold_Ctrl_2     <=  '1';
                    Hold_Ctrl_3     <=  '1';
                    Hold_Ctrl_4     <=  '1';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '1';
                    DFF_Ctrl_1      <=  '1';
                    DFF_Ctrl_2      <=  '1';
                    DFF_Ctrl_3      <=  '1';
                    DFF_Ctrl_4      <=  '1';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '0';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_5;
                WHEN state_5          =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '0';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '1';
                    pos_hold_ctrl   <=  '1';
                    Shuf_Ctrl_1     <=  "101";
                    Shuf_Ctrl_2     <=  "110";
                    Shuf_Ctrl_3     <=  "001";
                    Shuf_Ctrl_4     <=  "100";
                    Shuf_Ctrl_5     <=  "111";
                    Shuf_Ctrl_6     <=  "010";
                    Shuf_Ctrl_7     <=  "011";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "101";
                    Type_Sel_2      <=  "111";
                    Type_Sel_3      <=  "001";
                    Type_Sel_4      <=  "101";
                    Type_Sel_5      <=  "111";
                    Type_Sel_6      <=  "001";
                    Type_Sel_7      <=  "101";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "010";
                    DeShuf_Ctrl_2   <=  "101";
                    DeShuf_Ctrl_3   <=  "110";
                    DeShuf_Ctrl_4   <=  "011";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "001";
                    DeShuf_Ctrl_7   <=  "100";
```

```
                    Bypass_EN_0    <=  '1';
                    Bypass_EN_1    <=  '0';
                    Bypass_EN_2    <=  '0';
                    Bypass_EN_3    <=  '0';
                    Bypass_EN_4    <=  '0';
                    Bypass_EN_5    <=  '0';
                    Bypass_EN_6    <=  '0';
                    Bypass_EN_7    <=  '0';
                    Hold_Ctrl_0    <=  '0';
                    Hold_Ctrl_1    <=  '0';
                    Hold_Ctrl_2    <=  '0';
                    Hold_Ctrl_3    <=  '0';
                    Hold_Ctrl_4    <=  '0';
                    Hold_Ctrl_5    <=  '0';
                    Hold_Ctrl_6    <=  '0';
                    Hold_Ctrl_7    <=  '0';
                    DFF_Ctrl_0     <=  '0';
                    DFF_Ctrl_1     <=  '0';
                    DFF_Ctrl_2     <=  '0';
                    DFF_Ctrl_3     <=  '0';
                    DFF_Ctrl_4     <=  '0';
                    DFF_Ctrl_5     <=  '0';
                    DFF_Ctrl_6     <=  '0';
                    DFF_Ctrl_7     <=  '0';
                    hold_seg_0     <=  '1';
                    hold_seg_1     <=  '1';
                    hold_seg_2     <=  '1';
                    hold_seg_3     <=  '0';
                    hold_seg_4     <=  '1';
                    hold_seg_5     <=  '1';
                    hold_seg_6     <=  '1';
                    hold_seg_7     <=  '1';
                    in_ctrl_all_cb <=  '1';
                    hold_all_out   <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en     <=  '0';
                    currentstate   <= state_6;
                WHEN state_6          =>
                    hold_all_in    <=  '1';
                    hold_buf_3     <=  '1';
                    hold_buf_2     <=  '0';
                    hold_buf_1     <=  '1';
                    in_ctrl_buf_3  <=  '0';
                    in_ctrl_buf_2  <=  '1';
                    in_ctrl_buf_1  <=  '0';
                    pos_hold_ctrl  <=  '1';
                    Shuf_Ctrl_1    <=  "000";
                    Shuf_Ctrl_2    <=  "000";
                    Shuf_Ctrl_3    <=  "000";
                    Shuf_Ctrl_4    <=  "001";
                    Shuf_Ctrl_5    <=  "000";
                    Shuf_Ctrl_6    <=  "000";
                    Shuf_Ctrl_7    <=  "000";
                    Shuf_Ctrl_8    <=  "010";
                    Type_Sel_1     <=  "000";
                    Type_Sel_2     <=  "000";
                    Type_Sel_3     <=  "000";
                    Type_Sel_4     <=  "001";
                    Type_Sel_5     <=  "000";
                    Type_Sel_6     <=  "000";
                    Type_Sel_7     <=  "000";
                    Type_Sel_8     <=  "001";
                    Bypass_Sel_0   <=  "000";
                    Bypass_Sel_1   <=  "000";
                    Bypass_Sel_2   <=  "000";
                    Bypass_Sel_3   <=  "000";
                    Bypass_Sel_4   <=  "101";
                    Bypass_Sel_5   <=  "000";
```

```
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "011";
                    DeShuf_Ctrl_2   <=  "111";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '1';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_7;
                WHEN state_7          =>
                    hold_all_in     <=  '1';
                    hold_buf_3      <=  '0';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '1';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '1';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "011";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "101";
```

```
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "011";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '1';
                    Hold_Ctrl_1     <=  '1';
                    Hold_Ctrl_2     <=  '1';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '1';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '1';
                    DFF_Ctrl_1      <=  '1';
                    DFF_Ctrl_2      <=  '1';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '1';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_8;
                WHEN state_8          =>
                    hold_all_in     <=  '1';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '1';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
```

```
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "101";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "110";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "111";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "111";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "011";
                    DeShuf_Ctrl_6   <=  "111";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '1';
                    Hold_Ctrl_1     <=  '1';
                    Hold_Ctrl_2     <=  '1';
                    Hold_Ctrl_3     <=  '1';
                    Hold_Ctrl_4     <=  '1';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '1';
                    DFF_Ctrl_1      <=  '1';
                    DFF_Ctrl_2      <=  '1';
                    DFF_Ctrl_3      <=  '1';
                    DFF_Ctrl_4      <=  '1';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_9;
                WHEN state_9          =>
```

```
                        hold_all_in    <=  '0';
                        hold_buf_3     <=  '0';
                        hold_buf_2     <=  '0';
                        hold_buf_1     <=  '0';
                        in_ctrl_buf_3  <=  '0';
                        in_ctrl_buf_2  <=  '0';
                        in_ctrl_buf_1  <=  '0';
                        pos_hold_ctrl  <=  '1';
                        Shuf_Ctrl_1    <=  "000";
                        Shuf_Ctrl_2    <=  "000";
                        Shuf_Ctrl_3    <=  "000";
                        Shuf_Ctrl_4    <=  "111";
                        Shuf_Ctrl_5    <=  "000";
                        Shuf_Ctrl_6    <=  "000";
                        Shuf_Ctrl_7    <=  "000";
                        Shuf_Ctrl_8    <=  "000";
                        Type_Sel_1     <=  "000";
                        Type_Sel_2     <=  "000";
                        Type_Sel_3     <=  "000";
                        Type_Sel_4     <=  "011";
                        Type_Sel_5     <=  "000";
                        Type_Sel_6     <=  "000";
                        Type_Sel_7     <=  "000";
                        Type_Sel_8     <=  "000";
                        Bypass_Sel_0   <=  "000";
                        Bypass_Sel_1   <=  "000";
                        Bypass_Sel_2   <=  "000";
                        Bypass_Sel_3   <=  "000";
                        Bypass_Sel_4   <=  "000";
                        Bypass_Sel_5   <=  "000";
                        Bypass_Sel_6   <=  "000";
                        Bypass_Sel_7   <=  "000";
                        DeShuf_Ctrl_0  <=  "000";
                        DeShuf_Ctrl_1  <=  "000";
                        DeShuf_Ctrl_2  <=  "000";
                        DeShuf_Ctrl_3  <=  "000";
                        DeShuf_Ctrl_4  <=  "000";
                        DeShuf_Ctrl_5  <=  "000";
                        DeShuf_Ctrl_6  <=  "000";
                        DeShuf_Ctrl_7  <=  "011";
                        Bypass_EN_0    <=  '1';
                        Bypass_EN_1    <=  '0';
                        Bypass_EN_2    <=  '0';
                        Bypass_EN_3    <=  '0';
                        Bypass_EN_4    <=  '0';
                        Bypass_EN_5    <=  '0';
                        Bypass_EN_6    <=  '0';
                        Bypass_EN_7    <=  '0';
                        Hold_Ctrl_0    <=  '1';
                        Hold_Ctrl_1    <=  '1';
                        Hold_Ctrl_2    <=  '1';
                        Hold_Ctrl_3    <=  '1';
                        Hold_Ctrl_4    <=  '1';
                        Hold_Ctrl_5    <=  '1';
                        Hold_Ctrl_6    <=  '1';
                        Hold_Ctrl_7    <=  '0';
                        DFF_Ctrl_0     <=  '1';
                        DFF_Ctrl_1     <=  '1';
                        DFF_Ctrl_2     <=  '1';
                        DFF_Ctrl_3     <=  '1';
                        DFF_Ctrl_4     <=  '1';
                        DFF_Ctrl_5     <=  '1';
                        DFF_Ctrl_6     <=  '1';
                        DFF_Ctrl_7     <=  '0';
                        hold_seg_0     <=  '1';
                        hold_seg_1     <=  '1';
                        hold_seg_2     <=  '1';
                        hold_seg_3     <=  '1';
```

```
                hold_seg_4      <=  '0';
                hold_seg_5      <=  '1';
                hold_seg_6      <=  '1';
                hold_seg_7      <=  '1';
                in_ctrl_all_cb  <=  '1';
                hold_all_out    <=  '0';
                in_ctrl_all_out<=  '0';
                counter_en      <=  '0';
                currentstate    <= state_10;
            WHEN state_10         =>
                hold_all_in     <=  '0';
                hold_buf_3      <=  '0';
                hold_buf_2      <=  '0';
                hold_buf_1      <=  '0';
                in_ctrl_buf_3   <=  '0';
                in_ctrl_buf_2   <=  '0';
                in_ctrl_buf_1   <=  '0';
                pos_hold_ctrl   <=  '1';
                Shuf_Ctrl_1     <=  "011";
                Shuf_Ctrl_2     <=  "110";
                Shuf_Ctrl_3     <=  "111";
                Shuf_Ctrl_4     <=  "100";
                Shuf_Ctrl_5     <=  "001";
                Shuf_Ctrl_6     <=  "010";
                Shuf_Ctrl_7     <=  "101";
                Shuf_Ctrl_8     <=  "000";
                Type_Sel_1      <=  "101";
                Type_Sel_2      <=  "011";
                Type_Sel_3      <=  "010";
                Type_Sel_4      <=  "111";
                Type_Sel_5      <=  "001";
                Type_Sel_6      <=  "101";
                Type_Sel_7      <=  "011";
                Type_Sel_8      <=  "000";
                Bypass_Sel_0    <=  "000";
                Bypass_Sel_1    <=  "000";
                Bypass_Sel_2    <=  "000";
                Bypass_Sel_3    <=  "000";
                Bypass_Sel_4    <=  "000";
                Bypass_Sel_5    <=  "000";
                Bypass_Sel_6    <=  "000";
                Bypass_Sel_7    <=  "000";
                DeShuf_Ctrl_0   <=  "000";
                DeShuf_Ctrl_1   <=  "100";
                DeShuf_Ctrl_2   <=  "101";
                DeShuf_Ctrl_3   <=  "000";
                DeShuf_Ctrl_4   <=  "011";
                DeShuf_Ctrl_5   <=  "110";
                DeShuf_Ctrl_6   <=  "001";
                DeShuf_Ctrl_7   <=  "010";
                Bypass_EN_0     <=  '1';
                Bypass_EN_1     <=  '0';
                Bypass_EN_2     <=  '0';
                Bypass_EN_3     <=  '0';
                Bypass_EN_4     <=  '0';
                Bypass_EN_5     <=  '0';
                Bypass_EN_6     <=  '0';
                Bypass_EN_7     <=  '0';
                Hold_Ctrl_0     <=  '0';
                Hold_Ctrl_1     <=  '0';
                Hold_Ctrl_2     <=  '0';
                Hold_Ctrl_3     <=  '0';
                Hold_Ctrl_4     <=  '0';
                Hold_Ctrl_5     <=  '0';
                Hold_Ctrl_6     <=  '0';
                Hold_Ctrl_7     <=  '0';
                DFF_Ctrl_0      <=  '0';
                DFF_Ctrl_1      <=  '0';
```

```
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '0';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_11;
                WHEN state_11         =>
                    hold_all_in     <=  '1';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '1';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "011";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "010";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "001";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "100";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "111";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "101";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "001";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "111";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "101";
                    DeShuf_Ctrl_2   <=  "011";
                    DeShuf_Ctrl_3   <=  "001";
                    DeShuf_Ctrl_4   <=  "111";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
```

```
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '1';
                    in_ctrl_all_cb <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_12;
                WHEN state_12         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '0';
                    hold_buf_2      <=  '0';
                    hold_buf_1      <=  '0';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '1';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "101";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "110";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "111";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "011";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "010";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "110";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "001";
```

```
                    DeShuf_Ctrl_6  <=  "011";
                    DeShuf_Ctrl_7  <=  "101";
                    Bypass_EN_0    <=  '1';
                    Bypass_EN_1    <=  '0';
                    Bypass_EN_2    <=  '0';
                    Bypass_EN_3    <=  '0';
                    Bypass_EN_4    <=  '0';
                    Bypass_EN_5    <=  '0';
                    Bypass_EN_6    <=  '0';
                    Bypass_EN_7    <=  '0';
                    Hold_Ctrl_0    <=  '1';
                    Hold_Ctrl_1    <=  '1';
                    Hold_Ctrl_2    <=  '1';
                    Hold_Ctrl_3    <=  '1';
                    Hold_Ctrl_4    <=  '1';
                    Hold_Ctrl_5    <=  '0';
                    Hold_Ctrl_6    <=  '0';
                    Hold_Ctrl_7    <=  '0';
                    DFF_Ctrl_0     <=  '1';
                    DFF_Ctrl_1     <=  '1';
                    DFF_Ctrl_2     <=  '1';
                    DFF_Ctrl_3     <=  '1';
                    DFF_Ctrl_4     <=  '1';
                    DFF_Ctrl_5     <=  '0';
                    DFF_Ctrl_6     <=  '0';
                    DFF_Ctrl_7     <=  '0';
                    hold_seg_0     <=  '1';
                    hold_seg_1     <=  '1';
                    hold_seg_2     <=  '1';
                    hold_seg_3     <=  '1';
                    hold_seg_4     <=  '1';
                    hold_seg_5     <=  '1';
                    hold_seg_6     <=  '0';
                    hold_seg_7     <=  '1';
                    in_ctrl_all_cb <=  '1';
                    hold_all_out   <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en     <=  '0';
                    currentstate   <= state_13;
                WHEN state_13        =>
                    hold_all_in    <=  '0';
                    hold_buf_3     <=  '1';
                    hold_buf_2     <=  '1';
                    hold_buf_1     <=  '1';
                    in_ctrl_buf_3  <=  '0';
                    in_ctrl_buf_2  <=  '0';
                    in_ctrl_buf_1  <=  '0';
                    pos_hold_ctrl  <=  '0';
                    Shuf_Ctrl_1    <=  "111";
                    Shuf_Ctrl_2    <=  "010";
                    Shuf_Ctrl_3    <=  "101";
                    Shuf_Ctrl_4    <=  "100";
                    Shuf_Ctrl_5    <=  "011";
                    Shuf_Ctrl_6    <=  "110";
                    Shuf_Ctrl_7    <=  "001";
                    Shuf_Ctrl_8    <=  "000";
                    Type_Sel_1     <=  "100";
                    Type_Sel_2     <=  "101";
                    Type_Sel_3     <=  "010";
                    Type_Sel_4     <=  "011";
                    Type_Sel_5     <=  "111";
                    Type_Sel_6     <=  "110";
                    Type_Sel_7     <=  "001";
                    Type_Sel_8     <=  "000";
                    Bypass_Sel_0   <=  "000";
                    Bypass_Sel_1   <=  "000";
                    Bypass_Sel_2   <=  "000";
                    Bypass_Sel_3   <=  "000";
```

```
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "110";
                    DeShuf_Ctrl_2   <=  "001";
                    DeShuf_Ctrl_3   <=  "100";
                    DeShuf_Ctrl_4   <=  "011";
                    DeShuf_Ctrl_5   <=  "010";
                    DeShuf_Ctrl_6   <=  "101";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '1';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '1';
                    hold_seg_1      <=  '1';
                    hold_seg_2      <=  '1';
                    hold_seg_3      <=  '1';
                    hold_seg_4      <=  '1';
                    hold_seg_5      <=  '1';
                    hold_seg_6      <=  '1';
                    hold_seg_7      <=  '0';
                    in_ctrl_all_cb  <=  '1';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '1';
                    counter_en      <=  '0';
                    currentstate    <= state_14;
                WHEN state_14         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '0';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "000";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
```

```
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "000";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '0';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '0';
                    hold_seg_1      <=  '0';
                    hold_seg_2      <=  '0';
                    hold_seg_3      <=  '0';
                    hold_seg_4      <=  '0';
                    hold_seg_5      <=  '0';
                    hold_seg_6      <=  '0';
                    hold_seg_7      <=  '0';
                    in_ctrl_all_cb <=  '0';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '1';
                    counter_en      <=  '1';
                    currentstate    <= state_15;
                WHEN state_15         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '0';
```

```
                        Shuf_Ctrl_1     <=  "000";
                        Shuf_Ctrl_2     <=  "000";
                        Shuf_Ctrl_3     <=  "000";
                        Shuf_Ctrl_4     <=  "000";
                        Shuf_Ctrl_5     <=  "000";
                        Shuf_Ctrl_6     <=  "000";
                        Shuf_Ctrl_7     <=  "000";
                        Shuf_Ctrl_8     <=  "000";
                        Type_Sel_1      <=  "000";
                        Type_Sel_2      <=  "000";
                        Type_Sel_3      <=  "000";
                        Type_Sel_4      <=  "000";
                        Type_Sel_5      <=  "000";
                        Type_Sel_6      <=  "000";
                        Type_Sel_7      <=  "000";
                        Type_Sel_8      <=  "000";
                        Bypass_Sel_0    <=  "000";
                        Bypass_Sel_1    <=  "000";
                        Bypass_Sel_2    <=  "000";
                        Bypass_Sel_3    <=  "000";
                        Bypass_Sel_4    <=  "000";
                        Bypass_Sel_5    <=  "000";
                        Bypass_Sel_6    <=  "000";
                        Bypass_Sel_7    <=  "000";
                        DeShuf_Ctrl_0   <=  "000";
                        DeShuf_Ctrl_1   <=  "000";
                        DeShuf_Ctrl_2   <=  "000";
                        DeShuf_Ctrl_3   <=  "000";
                        DeShuf_Ctrl_4   <=  "000";
                        DeShuf_Ctrl_5   <=  "000";
                        DeShuf_Ctrl_6   <=  "000";
                        DeShuf_Ctrl_7   <=  "000";
                        Bypass_EN_0     <=  '0';
                        Bypass_EN_1     <=  '0';
                        Bypass_EN_2     <=  '0';
                        Bypass_EN_3     <=  '0';
                        Bypass_EN_4     <=  '0';
                        Bypass_EN_5     <=  '0';
                        Bypass_EN_6     <=  '0';
                        Bypass_EN_7     <=  '0';
                        Hold_Ctrl_0     <=  '0';
                        Hold_Ctrl_1     <=  '0';
                        Hold_Ctrl_2     <=  '0';
                        Hold_Ctrl_3     <=  '0';
                        Hold_Ctrl_4     <=  '0';
                        Hold_Ctrl_5     <=  '0';
                        Hold_Ctrl_6     <=  '0';
                        Hold_Ctrl_7     <=  '0';
                        DFF_Ctrl_0      <=  '0';
                        DFF_Ctrl_1      <=  '0';
                        DFF_Ctrl_2      <=  '0';
                        DFF_Ctrl_3      <=  '0';
                        DFF_Ctrl_4      <=  '0';
                        DFF_Ctrl_5      <=  '0';
                        DFF_Ctrl_6      <=  '0';
                        DFF_Ctrl_7      <=  '0';
                        hold_seg_0      <=  '0';
                        hold_seg_1      <=  '0';
                        hold_seg_2      <=  '0';
                        hold_seg_3      <=  '0';
                        hold_seg_4      <=  '0';
                        hold_seg_5      <=  '0';
                        hold_seg_6      <=  '0';
                        hold_seg_7      <=  '0';
                        in_ctrl_all_cb  <=  '0';
                        hold_all_out    <=  '0';
                        in_ctrl_all_out<=  '1';
                        counter_en      <=  '0';
```

```
                currentstate    <= state_16;
            WHEN state_16         =>
                hold_all_in     <=  '0';
                hold_buf_3      <=  '1';
                hold_buf_2      <=  '1';
                hold_buf_1      <=  '1';
                in_ctrl_buf_3   <=  '0';
                in_ctrl_buf_2   <=  '0';
                in_ctrl_buf_1   <=  '0';
                pos_hold_ctrl   <=  '0';
                Shuf_Ctrl_1     <=  "000";
                Shuf_Ctrl_2     <=  "000";
                Shuf_Ctrl_3     <=  "000";
                Shuf_Ctrl_4     <=  "000";
                Shuf_Ctrl_5     <=  "000";
                Shuf_Ctrl_6     <=  "000";
                Shuf_Ctrl_7     <=  "000";
                Shuf_Ctrl_8     <=  "000";
                Type_Sel_1      <=  "000";
                Type_Sel_2      <=  "000";
                Type_Sel_3      <=  "000";
                Type_Sel_4      <=  "000";
                Type_Sel_5      <=  "000";
                Type_Sel_6      <=  "000";
                Type_Sel_7      <=  "000";
                Type_Sel_8      <=  "000";
                Bypass_Sel_0    <=  "000";
                Bypass_Sel_1    <=  "000";
                Bypass_Sel_2    <=  "000";
                Bypass_Sel_3    <=  "000";
                Bypass_Sel_4    <=  "000";
                Bypass_Sel_5    <=  "000";
                Bypass_Sel_6    <=  "000";
                Bypass_Sel_7    <=  "000";
                DeShuf_Ctrl_0   <=  "000";
                DeShuf_Ctrl_1   <=  "000";
                DeShuf_Ctrl_2   <=  "000";
                DeShuf_Ctrl_3   <=  "000";
                DeShuf_Ctrl_4   <=  "000";
                DeShuf_Ctrl_5   <=  "000";
                DeShuf_Ctrl_6   <=  "000";
                DeShuf_Ctrl_7   <=  "000";
                Bypass_EN_0     <=  '0';
                Bypass_EN_1     <=  '0';
                Bypass_EN_2     <=  '0';
                Bypass_EN_3     <=  '0';
                Bypass_EN_4     <=  '0';
                Bypass_EN_5     <=  '0';
                Bypass_EN_6     <=  '0';
                Bypass_EN_7     <=  '0';
                Hold_Ctrl_0     <=  '0';
                Hold_Ctrl_1     <=  '0';
                Hold_Ctrl_2     <=  '0';
                Hold_Ctrl_3     <=  '0';
                Hold_Ctrl_4     <=  '0';
                Hold_Ctrl_5     <=  '0';
                Hold_Ctrl_6     <=  '0';
                Hold_Ctrl_7     <=  '0';
                DFF_Ctrl_0      <=  '0';
                DFF_Ctrl_1      <=  '0';
                DFF_Ctrl_2      <=  '0';
                DFF_Ctrl_3      <=  '0';
                DFF_Ctrl_4      <=  '0';
                DFF_Ctrl_5      <=  '0';
                DFF_Ctrl_6      <=  '0';
                DFF_Ctrl_7      <=  '0';
                hold_seg_0      <=  '0';
                hold_seg_1      <=  '0';
```

```
                hold_seg_2      <=  '0';
                hold_seg_3      <=  '0';
                hold_seg_4      <=  '0';
                hold_seg_5      <=  '0';
                hold_seg_6      <=  '0';
                hold_seg_7      <=  '0';
                in_ctrl_all_cb  <=  '0';
                hold_all_out    <=  '0';
                in_ctrl_all_out<=  '1';
                counter_en      <=  '0';
                currentstate    <= state_17;
            WHEN state_17         =>
                hold_all_in     <=  '0';
                hold_buf_3      <=  '1';
                hold_buf_2      <=  '1';
                hold_buf_1      <=  '1';
                in_ctrl_buf_3   <=  '0';
                in_ctrl_buf_2   <=  '0';
                in_ctrl_buf_1   <=  '0';
                pos_hold_ctrl   <=  '0';
                Shuf_Ctrl_1     <=  "000";
                Shuf_Ctrl_2     <=  "000";
                Shuf_Ctrl_3     <=  "000";
                Shuf_Ctrl_4     <=  "000";
                Shuf_Ctrl_5     <=  "000";
                Shuf_Ctrl_6     <=  "000";
                Shuf_Ctrl_7     <=  "000";
                Shuf_Ctrl_8     <=  "000";
                Type_Sel_1      <=  "000";
                Type_Sel_2      <=  "000";
                Type_Sel_3      <=  "000";
                Type_Sel_4      <=  "000";
                Type_Sel_5      <=  "000";
                Type_Sel_6      <=  "000";
                Type_Sel_7      <=  "000";
                Type_Sel_8      <=  "000";
                Bypass_Sel_0    <=  "000";
                Bypass_Sel_1    <=  "000";
                Bypass_Sel_2    <=  "000";
                Bypass_Sel_3    <=  "000";
                Bypass_Sel_4    <=  "000";
                Bypass_Sel_5    <=  "000";
                Bypass_Sel_6    <=  "000";
                Bypass_Sel_7    <=  "000";
                DeShuf_Ctrl_0   <=  "000";
                DeShuf_Ctrl_1   <=  "000";
                DeShuf_Ctrl_2   <=  "000";
                DeShuf_Ctrl_3   <=  "000";
                DeShuf_Ctrl_4   <=  "000";
                DeShuf_Ctrl_5   <=  "000";
                DeShuf_Ctrl_6   <=  "000";
                DeShuf_Ctrl_7   <=  "000";
                Bypass_EN_0     <=  '0';
                Bypass_EN_1     <=  '0';
                Bypass_EN_2     <=  '0';
                Bypass_EN_3     <=  '0';
                Bypass_EN_4     <=  '0';
                Bypass_EN_5     <=  '0';
                Bypass_EN_6     <=  '0';
                Bypass_EN_7     <=  '0';
                Hold_Ctrl_0     <=  '0';
                Hold_Ctrl_1     <=  '0';
                Hold_Ctrl_2     <=  '0';
                Hold_Ctrl_3     <=  '0';
                Hold_Ctrl_4     <=  '0';
                Hold_Ctrl_5     <=  '0';
                Hold_Ctrl_6     <=  '0';
                Hold_Ctrl_7     <=  '0';
```

```
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '0';
                    hold_seg_1      <=  '0';
                    hold_seg_2      <=  '0';
                    hold_seg_3      <=  '0';
                    hold_seg_4      <=  '0';
                    hold_seg_5      <=  '0';
                    hold_seg_6      <=  '0';
                    hold_seg_7      <=  '0';
                    in_ctrl_all_cb  <=  '0';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '1';
                    counter_en      <=  '0';
                    currentstate    <= state_18;
                WHEN state_18         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '0';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "000";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "000";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '0';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
```

```
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '0';
                    hold_seg_1      <=  '0';
                    hold_seg_2      <=  '0';
                    hold_seg_3      <=  '0';
                    hold_seg_4      <=  '0';
                    hold_seg_5      <=  '0';
                    hold_seg_6      <=  '0';
                    hold_seg_7      <=  '0';
                    in_ctrl_all_cb <=  '0';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '1';
                    counter_en      <=  '0';
                    currentstate    <= state_19;
                WHEN state_19         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '0';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "000";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "000";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
```

```
                    DeShuf_Ctrl_4  <=  "000";
                    DeShuf_Ctrl_5  <=  "000";
                    DeShuf_Ctrl_6  <=  "000";
                    DeShuf_Ctrl_7  <=  "000";
                    Bypass_EN_0    <=  '0';
                    Bypass_EN_1    <=  '0';
                    Bypass_EN_2    <=  '0';
                    Bypass_EN_3    <=  '0';
                    Bypass_EN_4    <=  '0';
                    Bypass_EN_5    <=  '0';
                    Bypass_EN_6    <=  '0';
                    Bypass_EN_7    <=  '0';
                    Hold_Ctrl_0    <=  '0';
                    Hold_Ctrl_1    <=  '0';
                    Hold_Ctrl_2    <=  '0';
                    Hold_Ctrl_3    <=  '0';
                    Hold_Ctrl_4    <=  '0';
                    Hold_Ctrl_5    <=  '0';
                    Hold_Ctrl_6    <=  '0';
                    Hold_Ctrl_7    <=  '0';
                    DFF_Ctrl_0     <=  '0';
                    DFF_Ctrl_1     <=  '0';
                    DFF_Ctrl_2     <=  '0';
                    DFF_Ctrl_3     <=  '0';
                    DFF_Ctrl_4     <=  '0';
                    DFF_Ctrl_5     <=  '0';
                    DFF_Ctrl_6     <=  '0';
                    DFF_Ctrl_7     <=  '0';
                    hold_seg_0     <=  '0';
                    hold_seg_1     <=  '0';
                    hold_seg_2     <=  '0';
                    hold_seg_3     <=  '0';
                    hold_seg_4     <=  '0';
                    hold_seg_5     <=  '0';
                    hold_seg_6     <=  '0';
                    hold_seg_7     <=  '0';
                    in_ctrl_all_cb <=  '0';
                    hold_all_out   <=  '0';
                    in_ctrl_all_out<=  '1';
                    counter_en     <=  '0';
                    currentstate   <= state_20;
                WHEN state_20        =>
                    hold_all_in    <=  '0';
                    hold_buf_3     <=  '1';
                    hold_buf_2     <=  '1';
                    hold_buf_1     <=  '1';
                    in_ctrl_buf_3  <=  '0';
                    in_ctrl_buf_2  <=  '0';
                    in_ctrl_buf_1  <=  '0';
                    pos_hold_ctrl  <=  '0';
                    Shuf_Ctrl_1    <=  "000";
                    Shuf_Ctrl_2    <=  "000";
                    Shuf_Ctrl_3    <=  "000";
                    Shuf_Ctrl_4    <=  "000";
                    Shuf_Ctrl_5    <=  "000";
                    Shuf_Ctrl_6    <=  "000";
                    Shuf_Ctrl_7    <=  "000";
                    Shuf_Ctrl_8    <=  "000";
                    Type_Sel_1     <=  "000";
                    Type_Sel_2     <=  "000";
                    Type_Sel_3     <=  "000";
                    Type_Sel_4     <=  "000";
                    Type_Sel_5     <=  "000";
                    Type_Sel_6     <=  "000";
                    Type_Sel_7     <=  "000";
                    Type_Sel_8     <=  "000";
                    Bypass_Sel_0   <=  "000";
                    Bypass_Sel_1   <=  "000";
```

```
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '0';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '0';
                    hold_seg_1      <=  '0';
                    hold_seg_2      <=  '0';
                    hold_seg_3      <=  '0';
                    hold_seg_4      <=  '0';
                    hold_seg_5      <=  '0';
                    hold_seg_6      <=  '0';
                    hold_seg_7      <=  '0';
                    in_ctrl_all_cb  <=  '0';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '1';
                    counter_en      <=  '0';
                    currentstate    <= state_21;
                WHEN state_21         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
                    in_ctrl_buf_1   <=  '0';
                    pos_hold_ctrl   <=  '0';
                    Shuf_Ctrl_1     <=  "000";
                    Shuf_Ctrl_2     <=  "000";
                    Shuf_Ctrl_3     <=  "000";
                    Shuf_Ctrl_4     <=  "000";
                    Shuf_Ctrl_5     <=  "000";
                    Shuf_Ctrl_6     <=  "000";
                    Shuf_Ctrl_7     <=  "000";
                    Shuf_Ctrl_8     <=  "000";
```

```
                    Type_Sel_1      <=  "000";
                    Type_Sel_2      <=  "000";
                    Type_Sel_3      <=  "000";
                    Type_Sel_4      <=  "000";
                    Type_Sel_5      <=  "000";
                    Type_Sel_6      <=  "000";
                    Type_Sel_7      <=  "000";
                    Type_Sel_8      <=  "000";
                    Bypass_Sel_0    <=  "000";
                    Bypass_Sel_1    <=  "000";
                    Bypass_Sel_2    <=  "000";
                    Bypass_Sel_3    <=  "000";
                    Bypass_Sel_4    <=  "000";
                    Bypass_Sel_5    <=  "000";
                    Bypass_Sel_6    <=  "000";
                    Bypass_Sel_7    <=  "000";
                    DeShuf_Ctrl_0   <=  "000";
                    DeShuf_Ctrl_1   <=  "000";
                    DeShuf_Ctrl_2   <=  "000";
                    DeShuf_Ctrl_3   <=  "000";
                    DeShuf_Ctrl_4   <=  "000";
                    DeShuf_Ctrl_5   <=  "000";
                    DeShuf_Ctrl_6   <=  "000";
                    DeShuf_Ctrl_7   <=  "000";
                    Bypass_EN_0     <=  '0';
                    Bypass_EN_1     <=  '0';
                    Bypass_EN_2     <=  '0';
                    Bypass_EN_3     <=  '0';
                    Bypass_EN_4     <=  '0';
                    Bypass_EN_5     <=  '0';
                    Bypass_EN_6     <=  '0';
                    Bypass_EN_7     <=  '0';
                    Hold_Ctrl_0     <=  '0';
                    Hold_Ctrl_1     <=  '0';
                    Hold_Ctrl_2     <=  '0';
                    Hold_Ctrl_3     <=  '0';
                    Hold_Ctrl_4     <=  '0';
                    Hold_Ctrl_5     <=  '0';
                    Hold_Ctrl_6     <=  '0';
                    Hold_Ctrl_7     <=  '0';
                    DFF_Ctrl_0      <=  '0';
                    DFF_Ctrl_1      <=  '0';
                    DFF_Ctrl_2      <=  '0';
                    DFF_Ctrl_3      <=  '0';
                    DFF_Ctrl_4      <=  '0';
                    DFF_Ctrl_5      <=  '0';
                    DFF_Ctrl_6      <=  '0';
                    DFF_Ctrl_7      <=  '0';
                    hold_seg_0      <=  '0';
                    hold_seg_1      <=  '0';
                    hold_seg_2      <=  '0';
                    hold_seg_3      <=  '0';
                    hold_seg_4      <=  '0';
                    hold_seg_5      <=  '0';
                    hold_seg_6      <=  '0';
                    hold_seg_7      <=  '0';
                    in_ctrl_all_cb <=  '0';
                    hold_all_out    <=  '0';
                    in_ctrl_all_out<=  '0';
                    counter_en      <=  '0';
                    currentstate    <= state_22;
                WHEN state_22         =>
                    hold_all_in     <=  '0';
                    hold_buf_3      <=  '1';
                    hold_buf_2      <=  '1';
                    hold_buf_1      <=  '1';
                    in_ctrl_buf_3   <=  '0';
                    in_ctrl_buf_2   <=  '0';
```

```
                        in_ctrl_buf_1  <=  '0';
                        pos_hold_ctrl  <=  '0';
                        Shuf_Ctrl_1    <=  "000";
                        Shuf_Ctrl_2    <=  "000";
                        Shuf_Ctrl_3    <=  "000";
                        Shuf_Ctrl_4    <=  "000";
                        Shuf_Ctrl_5    <=  "000";
                        Shuf_Ctrl_6    <=  "000";
                        Shuf_Ctrl_7    <=  "000";
                        Shuf_Ctrl_8    <=  "000";
                        Type_Sel_1     <=  "000";
                        Type_Sel_2     <=  "000";
                        Type_Sel_3     <=  "000";
                        Type_Sel_4     <=  "000";
                        Type_Sel_5     <=  "000";
                        Type_Sel_6     <=  "000";
                        Type_Sel_7     <=  "000";
                        Type_Sel_8     <=  "000";
                        Bypass_Sel_0   <=  "000";
                        Bypass_Sel_1   <=  "000";
                        Bypass_Sel_2   <=  "000";
                        Bypass_Sel_3   <=  "000";
                        Bypass_Sel_4   <=  "000";
                        Bypass_Sel_5   <=  "000";
                        Bypass_Sel_6   <=  "000";
                        Bypass_Sel_7   <=  "000";
                        DeShuf_Ctrl_0  <=  "000";
                        DeShuf_Ctrl_1  <=  "000";
                        DeShuf_Ctrl_2  <=  "000";
                        DeShuf_Ctrl_3  <=  "000";
                        DeShuf_Ctrl_4  <=  "000";
                        DeShuf_Ctrl_5  <=  "000";
                        DeShuf_Ctrl_6  <=  "000";
                        DeShuf_Ctrl_7  <=  "000";
                        Bypass_EN_0    <=  '0';
                        Bypass_EN_1    <=  '0';
                        Bypass_EN_2    <=  '0';
                        Bypass_EN_3    <=  '0';
                        Bypass_EN_4    <=  '0';
                        Bypass_EN_5    <=  '0';
                        Bypass_EN_6    <=  '0';
                        Bypass_EN_7    <=  '0';
                        Hold_Ctrl_0    <=  '0';
                        Hold_Ctrl_1    <=  '0';
                        Hold_Ctrl_2    <=  '0';
                        Hold_Ctrl_3    <=  '0';
                        Hold_Ctrl_4    <=  '0';
                        Hold_Ctrl_5    <=  '0';
                        Hold_Ctrl_6    <=  '0';
                        Hold_Ctrl_7    <=  '0';
                        DFF_Ctrl_0     <=  '0';
                        DFF_Ctrl_1     <=  '0';
                        DFF_Ctrl_2     <=  '0';
                        DFF_Ctrl_3     <=  '0';
                        DFF_Ctrl_4     <=  '0';
                        DFF_Ctrl_5     <=  '0';
                        DFF_Ctrl_6     <=  '0';
                        DFF_Ctrl_7     <=  '0';
                        hold_seg_0     <=  '1';
                        hold_seg_1     <=  '1';
                        hold_seg_2     <=  '1';
                        hold_seg_3     <=  '1';
                        hold_seg_4     <=  '1';
                        hold_seg_5     <=  '1';
                        hold_seg_6     <=  '1';
                        hold_seg_7     <=  '1';
                        in_ctrl_all_cb <=  '1';
                        hold_all_out   <=  '0';
```

```
                        in_ctrl_all_out<=  '0';
                        counter_en     <=  '0';
                        currentstate   <= state_0;
                           END CASE;
                     ELSE
                           currentstate <= currentstate;
                     END IF;
               END IF;
         END PROCESS;
END behavioral;
```

I. Top Level FFT 64-titik

Dengan selesainya implementasi masing-masing komponen, pada top level ini akan digabungkan masing-masing blok komponen. Sehingga Top Level FFT 64-titik hanya bersifat port mapping saja. Berikut ini adalah kode VHDL untuk Top Level FFT 64-titik.

Fft_64p_16b_top.vhd

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

ENTITY fft_64p_16b_top IS
    PORT (
                In_Stream     :    IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
                Mode          :    IN     STD_LOGIC;
                Data_Start    :    IN     STD_LOGIC;
                clk           :    IN     STD_LOGIC;
                rst           :    IN     STD_LOGIC;
                Out_Stream    :    OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
                Data_Out      :    OUT    STD_LOGIC
          );
END fft_64p_16b_top;

ARCHITECTURE structural of fft_64p_16b_top IS
COMPONENT input_circuit IS
    PORT (
                D               : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Clk             : IN     STD_LOGIC;
                hold_all_seg    : IN     STD_LOGIC;
                hold_buf_1      : IN     STD_LOGIC;
                hold_buf_2      : IN     STD_LOGIC;
                hold_buf_3      : IN     STD_LOGIC;
                in_ctrl_buf_1   : IN     STD_LOGIC;
                in_ctrl_buf_2   : IN     STD_LOGIC;
                in_ctrl_buf_3   : IN     STD_LOGIC;
                pos_hold_ctrl   : IN     STD_LOGIC;
                mode            : IN     STD_LOGIC;
                rst             : IN     STD_LOGIC;
                Q0              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q1              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q2              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q3              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q4              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q5              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q6              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q7              : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
          );
END COMPONENT;
COMPONENT input_counter IS
    PORT (
                clk           :    IN     STD_LOGIC;
                rst           :    IN     STD_LOGIC;
                datastart     :    IN   STD_LOGIC;
                counter_o     :    OUT  STD_LOGIC_VECTOR (5 DOWNTO 0);
                mastertrig    :    OUT  STD_LOGIC
          );
END COMPONENT;
```

```
COMPONENT fft_8p_16b_top IS
    PORT (
                xt0                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt1                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt2                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt3                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt4                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt5                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt6                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xt7                 :       IN      STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf0                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf1                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf2                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf3                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf4                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf5                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf6                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0);
                xf7                 :       OUT     STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT interdimensional_multiplier IS
    PORT (
                clk                 :       IN  STD_LOGIC;
                rst                 :       IN  STD_LOGIC;
                SET_0_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_1_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_2_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_3_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_4_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_5_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_6_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_7_IN            :       IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
                Shuf_Ctrl_1         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_2         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_3         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_4         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_5         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_6         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_7         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Shuf_Ctrl_8         :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_1          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_2          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_3          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_4          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_5          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_6          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_7          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Type_Sel_8          :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_0        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_1        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_2        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_3        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_4        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_5        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_6        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_Sel_7        :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_0       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_1       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_2       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_3       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_4       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_5       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_6       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                DeShuf_Ctrl_7       :       IN  STD_LOGIC_VECTOR (2 DOWNTO 0);
                Bypass_EN_0         :       IN  STD_LOGIC;
                Bypass_EN_1         :       IN  STD_LOGIC;
                Bypass_EN_2         :       IN  STD_LOGIC;
                Bypass_EN_3         :       IN  STD_LOGIC;
```

```
                Bypass_EN_4         :       IN  STD_LOGIC;
                Bypass_EN_5         :       IN  STD_LOGIC;
                Bypass_EN_6         :       IN  STD_LOGIC;
                Bypass_EN_7         :       IN  STD_LOGIC;
                Hold_Ctrl_0         :       IN  STD_LOGIC;
                Hold_Ctrl_1         :       IN  STD_LOGIC;
                Hold_Ctrl_2         :       IN  STD_LOGIC;
                Hold_Ctrl_3         :       IN  STD_LOGIC;
                Hold_Ctrl_4         :       IN  STD_LOGIC;
                Hold_Ctrl_5         :       IN  STD_LOGIC;
                Hold_Ctrl_6         :       IN  STD_LOGIC;
                Hold_Ctrl_7         :       IN  STD_LOGIC;
                DFF_Ctrl_0          :       IN  STD_LOGIC;
                DFF_Ctrl_1          :       IN  STD_LOGIC;
                DFF_Ctrl_2          :       IN  STD_LOGIC;
                DFF_Ctrl_3          :       IN  STD_LOGIC;
                DFF_Ctrl_4          :       IN  STD_LOGIC;
                DFF_Ctrl_5          :       IN  STD_LOGIC;
                DFF_Ctrl_6          :       IN  STD_LOGIC;
                DFF_Ctrl_7          :       IN  STD_LOGIC;
                SET_0_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_1_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_2_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_3_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_4_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_5_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_6_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                SET_7_OUT           :       OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT cb_circuit IS
    PORT (
                D1              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D2              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D3              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D4              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D5              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D6              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D7              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D8              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                clk             :   IN      STD_LOGIC;
                hold_seg_0      :   IN      STD_LOGIC;
                hold_seg_1      :   IN      STD_LOGIC;
                hold_seg_2      :   IN      STD_LOGIC;
                hold_seg_3      :   IN      STD_LOGIC;
                hold_seg_4      :   IN      STD_LOGIC;
                hold_seg_5      :   IN      STD_LOGIC;
                hold_seg_6      :   IN      STD_LOGIC;
                hold_seg_7      :   IN      STD_LOGIC;
                in_ctrl_all_seg:    IN      STD_LOGIC;
                rst             :   IN      STD_LOGIC;
                Q1              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q2              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q3              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q4              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q5              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q6              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q7              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0);
                Q8              :   OUT     STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT output_circuit IS
    PORT (
                D1              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D2              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D3              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D4              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
                D5              :   IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
```

```
                D6              :    IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
                D7              :    IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
                D8              :    IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
                clk             :    IN     STD_LOGIC;
                hold_all_seg    :    IN     STD_LOGIC;
                in_ctrl_all_seg:     IN     STD_LOGIC;
                mode            :    IN     STD_LOGIC;
                rst             :    IN     STD_LOGIC;
                Q               :    OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT output_counter IS
    PORT (
                clk             :    IN     STD_LOGIC;
                rst             :    IN     STD_LOGIC;
                dataind         :    IN     STD_LOGIC;
                counter_o       :    OUT    STD_LOGIC_VECTOR (5 DOWNTO 0);
                datavalid       :    OUT    STD_LOGIC
        );
END COMPONENT;
COMPONENT master_control IS
    PORT (
                 clk                : IN  STD_LOGIC;
                 rst                : IN  STD_LOGIC;
                 mastertrig         : IN  STD_LOGIC;
                hold_all_in         : OUT STD_LOGIC;
                hold_buf_3          : OUT STD_LOGIC;
                hold_buf_2          : OUT STD_LOGIC;
                hold_buf_1          : OUT STD_LOGIC;
                in_ctrl_buf_3       : OUT STD_LOGIC;
                in_ctrl_buf_2       : OUT STD_LOGIC;
                in_ctrl_buf_1       : OUT STD_LOGIC;
                pos_hold_ctrl       : OUT STD_LOGIC;
                Shuf_Ctrl_1         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_2         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_3         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_4         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_5         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_6         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_7         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Shuf_Ctrl_8         : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_1          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_2          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_3          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_4          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_5          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_6          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_7          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Type_Sel_8          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_0        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_1        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_2        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_3        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_4        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_5        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_6        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_Sel_7        : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_0       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_1       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_2       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_3       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_4       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_5       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_6       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                DeShuf_Ctrl_7       : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                Bypass_EN_0         : OUT STD_LOGIC;
                Bypass_EN_1         : OUT STD_LOGIC;
                Bypass_EN_2         : OUT STD_LOGIC;
```

```
                Bypass_EN_3       : OUT STD_LOGIC;
                Bypass_EN_4       : OUT STD_LOGIC;
                Bypass_EN_5       : OUT STD_LOGIC;
                Bypass_EN_6       : OUT STD_LOGIC;
                Bypass_EN_7       : OUT STD_LOGIC;
                Hold_Ctrl_0       : OUT STD_LOGIC;
                Hold_Ctrl_1       : OUT STD_LOGIC;
                Hold_Ctrl_2       : OUT STD_LOGIC;
                Hold_Ctrl_3       : OUT STD_LOGIC;
                Hold_Ctrl_4       : OUT STD_LOGIC;
                Hold_Ctrl_5       : OUT STD_LOGIC;
                Hold_Ctrl_6       : OUT STD_LOGIC;
                Hold_Ctrl_7       : OUT STD_LOGIC;
                DFF_Ctrl_0        : OUT STD_LOGIC;
                DFF_Ctrl_1        : OUT STD_LOGIC;
                DFF_Ctrl_2        : OUT STD_LOGIC;
                DFF_Ctrl_3        : OUT STD_LOGIC;
                DFF_Ctrl_4        : OUT STD_LOGIC;
                DFF_Ctrl_5        : OUT STD_LOGIC;
                DFF_Ctrl_6        : OUT STD_LOGIC;
                DFF_Ctrl_7        : OUT STD_LOGIC;
                hold_seg_0        : OUT STD_LOGIC;
                hold_seg_1        : OUT STD_LOGIC;
                hold_seg_2        : OUT STD_LOGIC;
                hold_seg_3        : OUT STD_LOGIC;
                hold_seg_4        : OUT STD_LOGIC;
                hold_seg_5        : OUT STD_LOGIC;
                hold_seg_6        : OUT STD_LOGIC;
                hold_seg_7        : OUT STD_LOGIC;
                in_ctrl_all_cb    : OUT STD_LOGIC;
                hold_all_out      : OUT STD_LOGIC;
                in_ctrl_all_out   : OUT STD_LOGIC;
                counter_en        : OUT STD_LOGIC
        );
END COMPONENT;

SIGNAL INPUT_CIRCUIT_DATA_0_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_1_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_2_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_3_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_4_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_5_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_6_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_CIRCUIT_DATA_7_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INPUT_COUNTER_MTRIG_OUT    : STD_LOGIC;
SIGNAL FFT_8P_FIRST_DATA_0_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_1_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_2_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_3_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_4_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_5_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_6_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_FIRST_DATA_7_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_0_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_1_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_2_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_3_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_4_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_5_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_6_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL INTERDIM_MULT_DATA_7_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_0_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_1_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_2_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_3_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_4_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_5_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL CB_CIRCUIT_DATA_6_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
```

```
SIGNAL CB_CIRCUIT_DATA_7_OUT      : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_0_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_1_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_2_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_3_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_4_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_5_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_6_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL FFT_8P_SECOND_DATA_7_OUT   : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL OUTPUT_CIRCUIT_DATA_OUT    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL OUTPUT_COUNTER_DATA_VAL    : STD_LOGIC;
SIGNAL master_out_hold_all_in     : STD_LOGIC;
SIGNAL master_out_hold_buf_3      : STD_LOGIC;
SIGNAL master_out_hold_buf_2      : STD_LOGIC;
SIGNAL master_out_hold_buf_1      : STD_LOGIC;
SIGNAL master_out_in_ctrl_buf_3   : STD_LOGIC;
SIGNAL master_out_in_ctrl_buf_2   : STD_LOGIC;
SIGNAL master_out_in_ctrl_buf_1   : STD_LOGIC;
SIGNAL master_out_pos_hold_ctrl   : STD_LOGIC;
SIGNAL master_out_Shuf_Ctrl_1     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_2     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_3     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_4     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_5     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_6     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_7     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Shuf_Ctrl_8     : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_1      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_2      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_3      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_4      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_5      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_6      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_7      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Type_Sel_8      : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_0    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_1    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_2    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_3    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_4    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_5    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_6    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_Sel_7    : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_0   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_1   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_2   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_3   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_4   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_5   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_6   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_DeShuf_Ctrl_7   : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL master_out_Bypass_EN_0     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_1     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_2     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_3     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_4     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_5     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_6     : STD_LOGIC;
SIGNAL master_out_Bypass_EN_7     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_0     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_1     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_2     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_3     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_4     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_5     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_6     : STD_LOGIC;
SIGNAL master_out_Hold_Ctrl_7     : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_0      : STD_LOGIC;
```

```
SIGNAL master_out_DFF_Ctrl_1      : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_2      : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_3      : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_4      : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_5      : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_6      : STD_LOGIC;
SIGNAL master_out_DFF_Ctrl_7      : STD_LOGIC;
SIGNAL master_out_hold_seg_0      : STD_LOGIC;
SIGNAL master_out_hold_seg_1      : STD_LOGIC;
SIGNAL master_out_hold_seg_2      : STD_LOGIC;
SIGNAL master_out_hold_seg_3      : STD_LOGIC;
SIGNAL master_out_hold_seg_4      : STD_LOGIC;
SIGNAL master_out_hold_seg_5      : STD_LOGIC;
SIGNAL master_out_hold_seg_6      : STD_LOGIC;
SIGNAL master_out_hold_seg_7      : STD_LOGIC;
SIGNAL master_out_in_ctrl_all_cb : STD_LOGIC;
SIGNAL master_out_hold_all_out    : STD_LOGIC;
SIGNAL master_out_in_ctrl_all_out: STD_LOGIC;
SIGNAL master_out_counter_en      : STD_LOGIC;

BEGIN
    Out_Stream <= OUTPUT_CIRCUIT_DATA_OUT;
    Data_Out   <= OUTPUT_COUNTER_DATA_VAL;

INPUT_UNIT :
    input_circuit
         PORT MAP (
                              D            =>In_Stream,
                              clk          =>clk,
                              hold_all_seg =>master_out_hold_all_in,
                              hold_buf_1   =>master_out_hold_buf_1,
                              hold_buf_2   =>master_out_hold_buf_2,
                              hold_buf_3   =>master_out_hold_buf_3,
                              in_ctrl_buf_1=>master_out_in_ctrl_buf_1,
                              in_ctrl_buf_2=>master_out_in_ctrl_buf_2,
                              in_ctrl_buf_3=>master_out_in_ctrl_buf_3,
                              pos_hold_ctrl=>master_out_pos_hold_ctrl,
                              mode         =>Mode,
                              rst          =>rst,
                              Q0           =>INPUT_CIRCUIT_DATA_0_OUT,
                              Q1           =>INPUT_CIRCUIT_DATA_1_OUT,
                              Q2           =>INPUT_CIRCUIT_DATA_2_OUT,
                              Q3           =>INPUT_CIRCUIT_DATA_3_OUT,
                              Q4           =>INPUT_CIRCUIT_DATA_4_OUT,
                              Q5           =>INPUT_CIRCUIT_DATA_5_OUT,
                              Q6           =>INPUT_CIRCUIT_DATA_6_OUT,
                              Q7           =>INPUT_CIRCUIT_DATA_7_OUT
                );
IN_COUNT :
    input_counter
         PORT MAP (
                       clk           =>clk,
                       rst           =>rst,
                       datastart     =>Data_Start,
                       mastertrig    =>INPUT_COUNTER_MTRIG_OUT
                );
FFT_8P_FIRSTSTAGE :
    fft_8p_16b_top
         PORT MAP (
                              xt0 =>INPUT_CIRCUIT_DATA_0_OUT,
                              xt1 =>INPUT_CIRCUIT_DATA_1_OUT,
                              xt2 =>INPUT_CIRCUIT_DATA_2_OUT,
                              xt3 =>INPUT_CIRCUIT_DATA_3_OUT,
                              xt4 =>INPUT_CIRCUIT_DATA_4_OUT,
                              xt5 =>INPUT_CIRCUIT_DATA_5_OUT,
                              xt6 =>INPUT_CIRCUIT_DATA_6_OUT,
                              xt7 =>INPUT_CIRCUIT_DATA_7_OUT,
                              xf0 =>FFT_8P_FIRST_DATA_0_OUT,
```

```
                        xf1 =>FFT_8P_FIRST_DATA_1_OUT,
                        xf2 =>FFT_8P_FIRST_DATA_2_OUT,
                        xf3 =>FFT_8P_FIRST_DATA_3_OUT,
                        xf4 =>FFT_8P_FIRST_DATA_4_OUT,
                        xf5 =>FFT_8P_FIRST_DATA_5_OUT,
                        xf6 =>FFT_8P_FIRST_DATA_6_OUT,
                        xf7 =>FFT_8P_FIRST_DATA_7_OUT
            );
INTERDIM_MULT :
   interdimensional_multiplier
        PORT MAP (
                    clk                 =>clk,
                    rst                 =>rst,
                    SET_0_IN            =>FFT_8P_FIRST_DATA_0_OUT,
                    SET_1_IN            =>FFT_8P_FIRST_DATA_1_OUT,
                    SET_2_IN            =>FFT_8P_FIRST_DATA_2_OUT,
                    SET_3_IN            =>FFT_8P_FIRST_DATA_3_OUT,
                    SET_4_IN            =>FFT_8P_FIRST_DATA_4_OUT,
                    SET_5_IN            =>FFT_8P_FIRST_DATA_5_OUT,
                    SET_6_IN            =>FFT_8P_FIRST_DATA_6_OUT,
                    SET_7_IN            =>FFT_8P_FIRST_DATA_7_OUT,
                    Shuf_Ctrl_1         =>master_out_Shuf_Ctrl_1,
                    Shuf_Ctrl_2         =>master_out_Shuf_Ctrl_2,
                    Shuf_Ctrl_3         =>master_out_Shuf_Ctrl_3,
                    Shuf_Ctrl_4         =>master_out_Shuf_Ctrl_4,
                    Shuf_Ctrl_5         =>master_out_Shuf_Ctrl_5,
                    Shuf_Ctrl_6         =>master_out_Shuf_Ctrl_6,
                    Shuf_Ctrl_7         =>master_out_Shuf_Ctrl_7,
                    Shuf_Ctrl_8         =>master_out_Shuf_Ctrl_8,
                    Type_Sel_1          =>master_out_Type_Sel_1,
                    Type_Sel_2          =>master_out_Type_Sel_2,
                    Type_Sel_3          =>master_out_Type_Sel_3,
                    Type_Sel_4          =>master_out_Type_Sel_4,
                    Type_Sel_5          =>master_out_Type_Sel_5,
                    Type_Sel_6          =>master_out_Type_Sel_6,
                    Type_Sel_7          =>master_out_Type_Sel_7,
                    Type_Sel_8          =>master_out_Type_Sel_8,
                    Bypass_Sel_0        =>master_out_Bypass_Sel_0,
                    Bypass_Sel_1        =>master_out_Bypass_Sel_1,
                    Bypass_Sel_2        =>master_out_Bypass_Sel_2,
                    Bypass_Sel_3        =>master_out_Bypass_Sel_3,
                    Bypass_Sel_4        =>master_out_Bypass_Sel_4,
                    Bypass_Sel_5        =>master_out_Bypass_Sel_5,
                    Bypass_Sel_6        =>master_out_Bypass_Sel_6,
                    Bypass_Sel_7        =>master_out_Bypass_Sel_7,
                    DeShuf_Ctrl_0       =>master_out_DeShuf_Ctrl_0,
                    DeShuf_Ctrl_1       =>master_out_DeShuf_Ctrl_1,
                    DeShuf_Ctrl_2       =>master_out_DeShuf_Ctrl_2,
                    DeShuf_Ctrl_3       =>master_out_DeShuf_Ctrl_3,
                    DeShuf_Ctrl_4       =>master_out_DeShuf_Ctrl_4,
                    DeShuf_Ctrl_5       =>master_out_DeShuf_Ctrl_5,
                    DeShuf_Ctrl_6       =>master_out_DeShuf_Ctrl_6,
                    DeShuf_Ctrl_7       =>master_out_DeShuf_Ctrl_7,
                    Bypass_EN_0         =>master_out_Bypass_EN_0,
                    Bypass_EN_1         =>master_out_Bypass_EN_1,
                    Bypass_EN_2         =>master_out_Bypass_EN_2,
                    Bypass_EN_3         =>master_out_Bypass_EN_3,
                    Bypass_EN_4         =>master_out_Bypass_EN_4,
                    Bypass_EN_5         =>master_out_Bypass_EN_5,
                    Bypass_EN_6         =>master_out_Bypass_EN_6,
                    Bypass_EN_7         =>master_out_Bypass_EN_7,
                    Hold_Ctrl_0         =>master_out_Hold_Ctrl_0,
                    Hold_Ctrl_1         =>master_out_Hold_Ctrl_1,
                    Hold_Ctrl_2         =>master_out_Hold_Ctrl_2,
                    Hold_Ctrl_3         =>master_out_Hold_Ctrl_3,
                    Hold_Ctrl_4         =>master_out_Hold_Ctrl_4,
                    Hold_Ctrl_5         =>master_out_Hold_Ctrl_5,
                    Hold_Ctrl_6         =>master_out_Hold_Ctrl_6,
```

```
                    Hold_Ctrl_7         =>master_out_Hold_Ctrl_7,
                    DFF_Ctrl_0          =>master_out_DFF_Ctrl_0,
                    DFF_Ctrl_1          =>master_out_DFF_Ctrl_1,
                    DFF_Ctrl_2          =>master_out_DFF_Ctrl_2,
                    DFF_Ctrl_3          =>master_out_DFF_Ctrl_3,
                    DFF_Ctrl_4          =>master_out_DFF_Ctrl_4,
                    DFF_Ctrl_5          =>master_out_DFF_Ctrl_5,
                    DFF_Ctrl_6          =>master_out_DFF_Ctrl_6,
                    DFF_Ctrl_7          =>master_out_DFF_Ctrl_7,
                    SET_0_OUT           =>INTERDIM_MULT_DATA_0_OUT,
                    SET_1_OUT           =>INTERDIM_MULT_DATA_1_OUT,
                    SET_2_OUT           =>INTERDIM_MULT_DATA_2_OUT,
                    SET_3_OUT           =>INTERDIM_MULT_DATA_3_OUT,
                    SET_4_OUT           =>INTERDIM_MULT_DATA_4_OUT,
                    SET_5_OUT           =>INTERDIM_MULT_DATA_5_OUT,
                    SET_6_OUT           =>INTERDIM_MULT_DATA_6_OUT,
                    SET_7_OUT           =>INTERDIM_MULT_DATA_7_OUT
                );
CB_UNIT :
    cb_circuit
        PORT MAP (
                            D1              =>INTERDIM_MULT_DATA_0_OUT,
                            D2              =>INTERDIM_MULT_DATA_1_OUT,
                            D3              =>INTERDIM_MULT_DATA_2_OUT,
                            D4              =>INTERDIM_MULT_DATA_3_OUT,
                            D5              =>INTERDIM_MULT_DATA_4_OUT,
                            D6              =>INTERDIM_MULT_DATA_5_OUT,
                            D7              =>INTERDIM_MULT_DATA_6_OUT,
                            D8              =>INTERDIM_MULT_DATA_7_OUT,
                            clk             =>clk,
                            hold_seg_0      =>master_out_hold_seg_0,
                            hold_seg_1      =>master_out_hold_seg_1,
                            hold_seg_2      =>master_out_hold_seg_2,
                            hold_seg_3      =>master_out_hold_seg_3,
                            hold_seg_4      =>master_out_hold_seg_4,
                            hold_seg_5      =>master_out_hold_seg_5,
                            hold_seg_6      =>master_out_hold_seg_6,
                            hold_seg_7      =>master_out_hold_seg_7,
                            in_ctrl_all_seg=>master_out_in_ctrl_all_cb,
                            rst             =>rst,
                            Q1              =>CB_CIRCUIT_DATA_0_OUT,
                            Q2              =>CB_CIRCUIT_DATA_1_OUT,
                            Q3              =>CB_CIRCUIT_DATA_2_OUT,
                            Q4              =>CB_CIRCUIT_DATA_3_OUT,
                            Q5              =>CB_CIRCUIT_DATA_4_OUT,
                            Q6              =>CB_CIRCUIT_DATA_5_OUT,
                            Q7              =>CB_CIRCUIT_DATA_6_OUT,
                            Q8              =>CB_CIRCUIT_DATA_7_OUT

                );
FFT_8P_SECONDSTAGE :
    fft_8p_16b_top
        PORT MAP (
                            xt0 =>CB_CIRCUIT_DATA_0_OUT,
                            xt1 =>CB_CIRCUIT_DATA_1_OUT,
                            xt2 =>CB_CIRCUIT_DATA_2_OUT,
                            xt3 =>CB_CIRCUIT_DATA_3_OUT,
                            xt4 =>CB_CIRCUIT_DATA_4_OUT,
                            xt5 =>CB_CIRCUIT_DATA_5_OUT,
                            xt6 =>CB_CIRCUIT_DATA_6_OUT,
                            xt7 =>CB_CIRCUIT_DATA_7_OUT,
                            xf0 =>FFT_8P_SECOND_DATA_0_OUT,
                            xf1 =>FFT_8P_SECOND_DATA_1_OUT,
                            xf2 =>FFT_8P_SECOND_DATA_2_OUT,
                            xf3 =>FFT_8P_SECOND_DATA_3_OUT,
                            xf4 =>FFT_8P_SECOND_DATA_4_OUT,
                            xf5 =>FFT_8P_SECOND_DATA_5_OUT,
                            xf6 =>FFT_8P_SECOND_DATA_6_OUT,
```

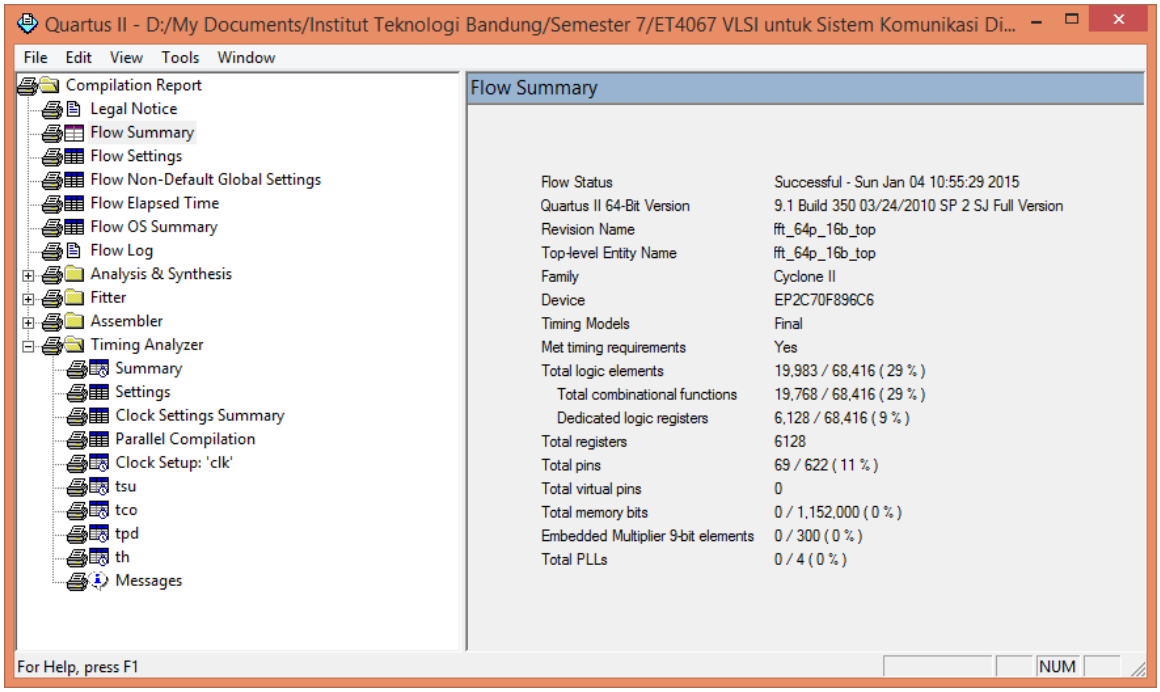
```
                            xf7 =>FFT_8P_SECOND_DATA_7_OUT
                );
Output_Block :
    output_circuit
          PORT MAP (
                        D1                  =>FFT_8P_SECOND_DATA_0_OUT,
                        D2                  =>FFT_8P_SECOND_DATA_1_OUT,
                        D3                  =>FFT_8P_SECOND_DATA_2_OUT,
                        D4                  =>FFT_8P_SECOND_DATA_3_OUT,
                        D5                  =>FFT_8P_SECOND_DATA_4_OUT,
                        D6                  =>FFT_8P_SECOND_DATA_5_OUT,
                        D7                  =>FFT_8P_SECOND_DATA_6_OUT,
                        D8                  =>FFT_8P_SECOND_DATA_7_OUT,
                        clk                 =>clk,
                        hold_all_seg        =>master_out_hold_all_out,
                        in_ctrl_all_seg     =>master_out_in_ctrl_all_out,
                        mode                =>Mode,
                        rst                 =>rst,
                        Q                   =>OUTPUT_CIRCUIT_DATA_OUT
                 );
OUT_COUNT :
    output_counter
          PORT MAP (
                                clk             =>clk,
                                rst             =>rst,
                                dataind          =>master_out_counter_en,
                                datavalid       =>OUTPUT_COUNTER_DATA_VAL
                );
MASTER_CTRL :
    master_control
         PORT MAP (
                       clk              =>clk,
                       rst              =>rst,
                        mastertrig      =>INPUT_COUNTER_MTRIG_OUT,
                        hold_all_in     =>master_out_hold_all_in,
                        hold_buf_3      =>master_out_hold_buf_3,
                        hold_buf_2      =>master_out_hold_buf_2,
                        hold_buf_1      =>master_out_hold_buf_1,
                        in_ctrl_buf_3   =>master_out_in_ctrl_buf_3,
                        in_ctrl_buf_2   =>master_out_in_ctrl_buf_2,
                        in_ctrl_buf_1   =>master_out_in_ctrl_buf_1,
                        pos_hold_ctrl   =>master_out_pos_hold_ctrl,
                        Shuf_Ctrl_1     =>master_out_Shuf_Ctrl_1,
                        Shuf_Ctrl_2     =>master_out_Shuf_Ctrl_2,
                        Shuf_Ctrl_3     =>master_out_Shuf_Ctrl_3,
                        Shuf_Ctrl_4     =>master_out_Shuf_Ctrl_4,
                        Shuf_Ctrl_5     =>master_out_Shuf_Ctrl_5,
                        Shuf_Ctrl_6     =>master_out_Shuf_Ctrl_6,
                        Shuf_Ctrl_7     =>master_out_Shuf_Ctrl_7,
                        Shuf_Ctrl_8     =>master_out_Shuf_Ctrl_8,
                        Type_Sel_1      =>master_out_Type_Sel_1,
                        Type_Sel_2      =>master_out_Type_Sel_2,
                        Type_Sel_3      =>master_out_Type_Sel_3,
                        Type_Sel_4      =>master_out_Type_Sel_4,
                        Type_Sel_5      =>master_out_Type_Sel_5,
                        Type_Sel_6      =>master_out_Type_Sel_6,
                        Type_Sel_7      =>master_out_Type_Sel_7,
                        Type_Sel_8      =>master_out_Type_Sel_8,
                        Bypass_Sel_0    =>master_out_Bypass_Sel_0,
                        Bypass_Sel_1    =>master_out_Bypass_Sel_1,
                        Bypass_Sel_2    =>master_out_Bypass_Sel_2,
                        Bypass_Sel_3    =>master_out_Bypass_Sel_3,
                        Bypass_Sel_4    =>master_out_Bypass_Sel_4,
                        Bypass_Sel_5    =>master_out_Bypass_Sel_5,
                        Bypass_Sel_6    =>master_out_Bypass_Sel_6,
                        Bypass_Sel_7    =>master_out_Bypass_Sel_7,
                        DeShuf_Ctrl_0   =>master_out_DeShuf_Ctrl_0,
                        DeShuf_Ctrl_1   =>master_out_DeShuf_Ctrl_1,
```

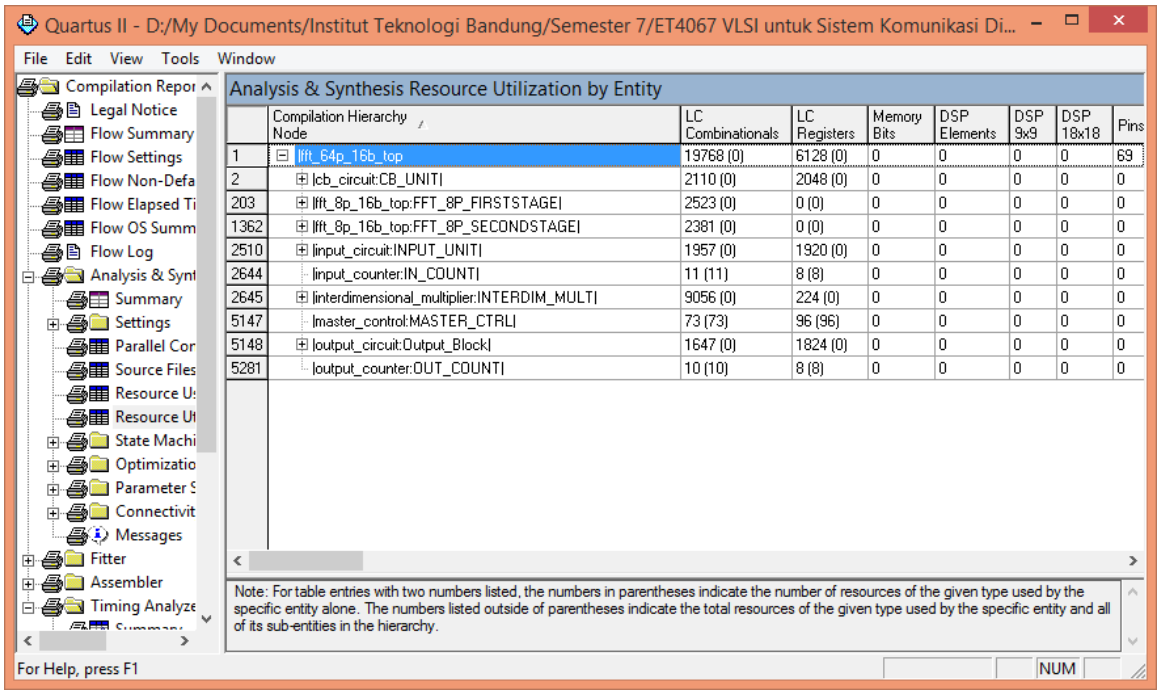
```
                    DeShuf_Ctrl_2   =>master_out_DeShuf_Ctrl_2,
                    DeShuf_Ctrl_3   =>master_out_DeShuf_Ctrl_3,
                    DeShuf_Ctrl_4   =>master_out_DeShuf_Ctrl_4,
                    DeShuf_Ctrl_5   =>master_out_DeShuf_Ctrl_5,
                    DeShuf_Ctrl_6   =>master_out_DeShuf_Ctrl_6,
                    DeShuf_Ctrl_7   =>master_out_DeShuf_Ctrl_7,
                    Bypass_EN_0     =>master_out_Bypass_EN_0,
                    Bypass_EN_1     =>master_out_Bypass_EN_1,
                    Bypass_EN_2     =>master_out_Bypass_EN_2,
                    Bypass_EN_3     =>master_out_Bypass_EN_3,
                    Bypass_EN_4     =>master_out_Bypass_EN_4,
                    Bypass_EN_5     =>master_out_Bypass_EN_5,
                    Bypass_EN_6     =>master_out_Bypass_EN_6,
                    Bypass_EN_7     =>master_out_Bypass_EN_7,
                    Hold_Ctrl_0     =>master_out_Hold_Ctrl_0,
                    Hold_Ctrl_1     =>master_out_Hold_Ctrl_1,
                    Hold_Ctrl_2     =>master_out_Hold_Ctrl_2,
                    Hold_Ctrl_3     =>master_out_Hold_Ctrl_3,
                    Hold_Ctrl_4     =>master_out_Hold_Ctrl_4,
                    Hold_Ctrl_5     =>master_out_Hold_Ctrl_5,
                    Hold_Ctrl_6     =>master_out_Hold_Ctrl_6,
                    Hold_Ctrl_7     =>master_out_Hold_Ctrl_7,
                    DFF_Ctrl_0      =>master_out_DFF_Ctrl_0,
                    DFF_Ctrl_1      =>master_out_DFF_Ctrl_1,
                    DFF_Ctrl_2      =>master_out_DFF_Ctrl_2,
                    DFF_Ctrl_3      =>master_out_DFF_Ctrl_3,
                    DFF_Ctrl_4      =>master_out_DFF_Ctrl_4,
                    DFF_Ctrl_5      =>master_out_DFF_Ctrl_5,
                    DFF_Ctrl_6      =>master_out_DFF_Ctrl_6,
                    DFF_Ctrl_7      =>master_out_DFF_Ctrl_7,
                    hold_seg_0      =>master_out_hold_seg_0,
                    hold_seg_1      =>master_out_hold_seg_1,
                    hold_seg_2      =>master_out_hold_seg_2,
                    hold_seg_3      =>master_out_hold_seg_3,
                    hold_seg_4      =>master_out_hold_seg_4,
                    hold_seg_5      =>master_out_hold_seg_5,
                    hold_seg_6      =>master_out_hold_seg_6,
                    hold_seg_7      =>master_out_hold_seg_7,
                    in_ctrl_all_cb  =>master_out_in_ctrl_all_cb,
                    hold_all_out    =>master_out_hold_all_out,
                    in_ctrl_all_out =>master_out_in_ctrl_all_out,
                    counter_en      =>master_out_counter_en
                );
END structural;
```


+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

J. Analisis dan Sintesis pada Desain Prosesor FFT/IFFT 64-titik

Dengan menggunakan semua kode VHDL yang dipaparkan sebelumnya, sintesis hardware dilakukan. Sintesis dilakukan menggunakan Altera Quartus II 9.1sp2 pada Device Altera Cyclone II EP2C70F896C6. Analisis dan sintesis berhasil dilakukan dengan *summary* sebagai berikut.

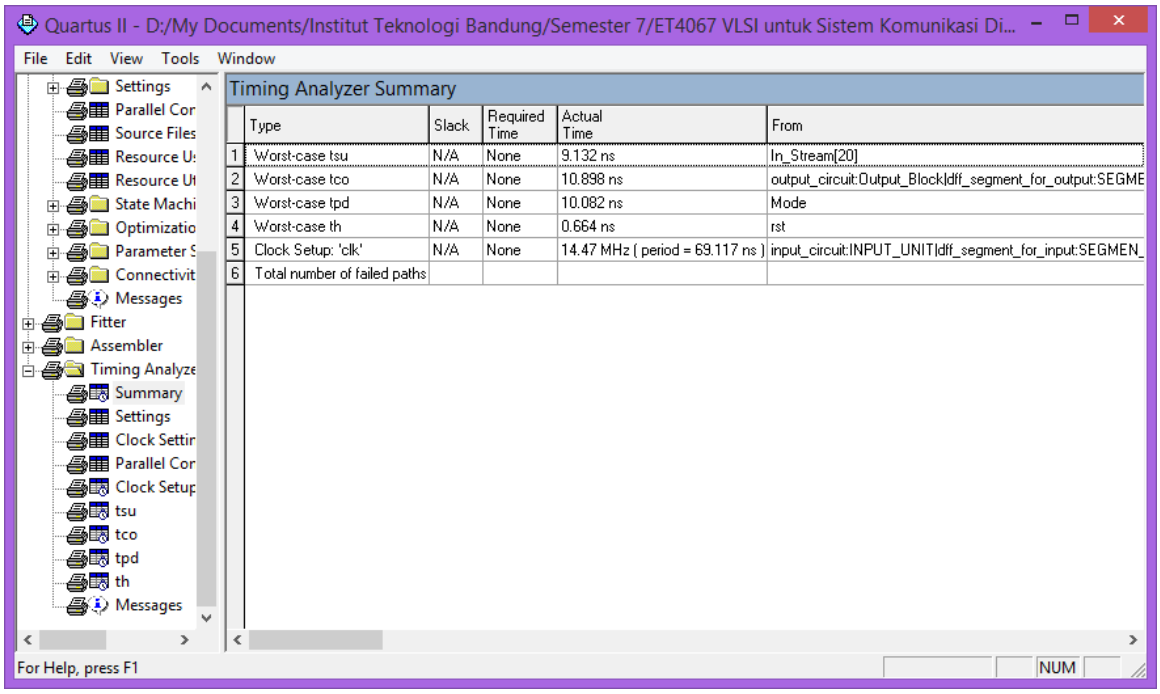




Dari hasil sintesis diketahui bahwa untuk mengimplementasikan prosesor FFT/IFFT 64-titik memerlukan hampir 19.983 logic element termasuk 6.128 register atau menempati 30% dari FPGA Cyclone II. Total penggunaan logic element tersebut dapat dijabarkan sebagai berikut. Interdimensional Twiddle Factor Multiplier menjadi blok yang paling banyak mengkonsumsi logic element yaitu sekitar 9.056 logic element atau hampir 50% dari total logic element yang digunakan. Sisanya, digunakan untuk dua buah blok FFT 8-titik yang membutuhkan masing-masing sekitar 2.500

logic element, Internal Register yang menggunakan 2.110 logic element, Input Unit yang menggunakan 1.957 logic element, dan Output Unit yang menggunakan 1.647 logic element. Master control sendiri hanya memerlukan kurang dari 100 logic element untuk merealiasikannya.

Selanjutnya dilakukan analisis *timing* dari rangkaian yang telah dibuat menggunakan Timing Analyzer pada Altera Quartus II 9.1sp2.



Rangkaian yang telah dibuat dapat berjalan pada clock 14 MHz. Dengan demikian, periode clock adalah sekitar 70ns. Dengan asumsi input buffer dan output buffer telah penuh, setiap operasi FFT/IFFT 64-titik yang dilakukan membutuhkan 23 clock cycle, dihitung dari state master control. Dengan demikian, total waktu pemrosesan untuk sebuah set data menggunakan prosesor FFT/IFFT 64-titik ini adalah sekitar 1600ns atau sekitar 1,6µs. Dengan demikian rangkaian ini telah memenuhi persyaratan waktu pemrosesan maksimum untuk WLAN 802.11a sebesar 4µs per set data.

Tahap-tahap rangkaian dalam memproses data diberikan pada waveform berikut. Waveform ini dihasilkan dari simulasi menggunakan ModelSim sehingga sinyal internal dari rangkaian dapat dilihat untuk mempermudah analisis.

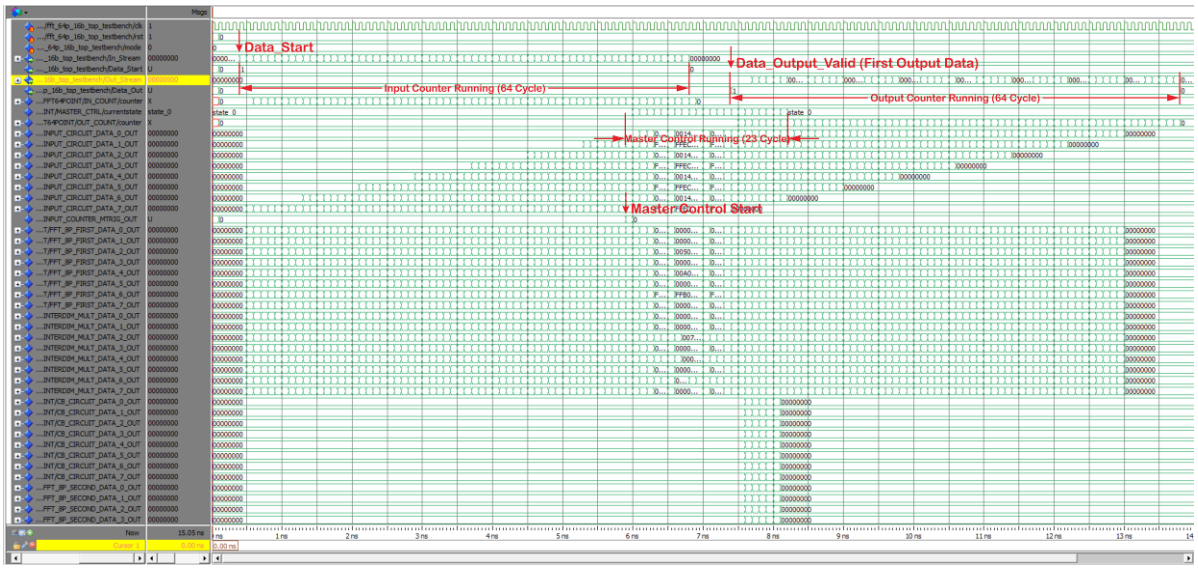


Diagram Waveform mulai dari Input Data, Pemrosesan Data, Hingga Output Data

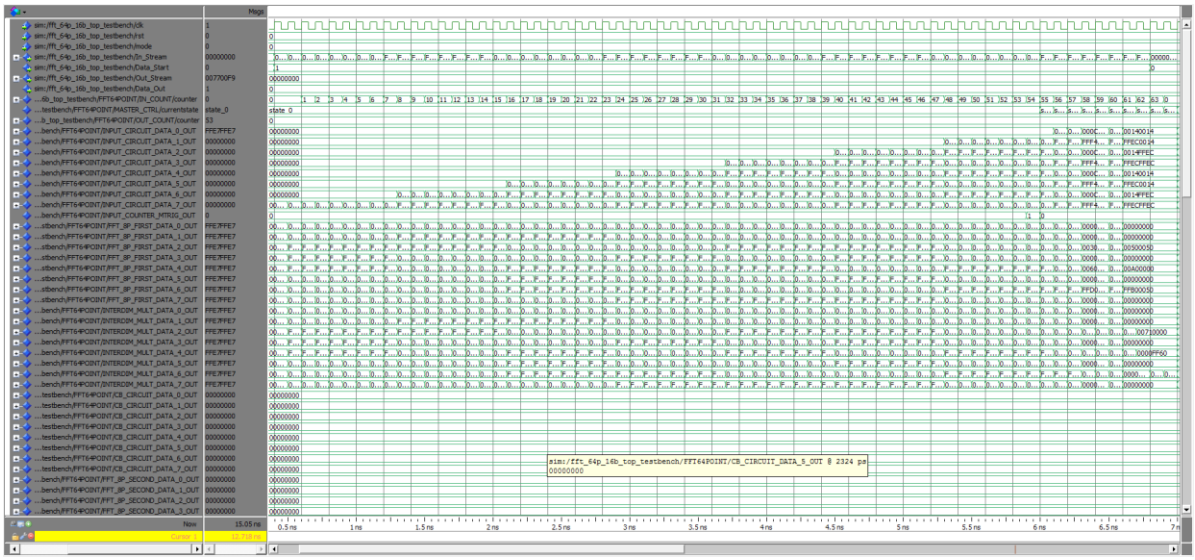


Diagram Waveform Input Data

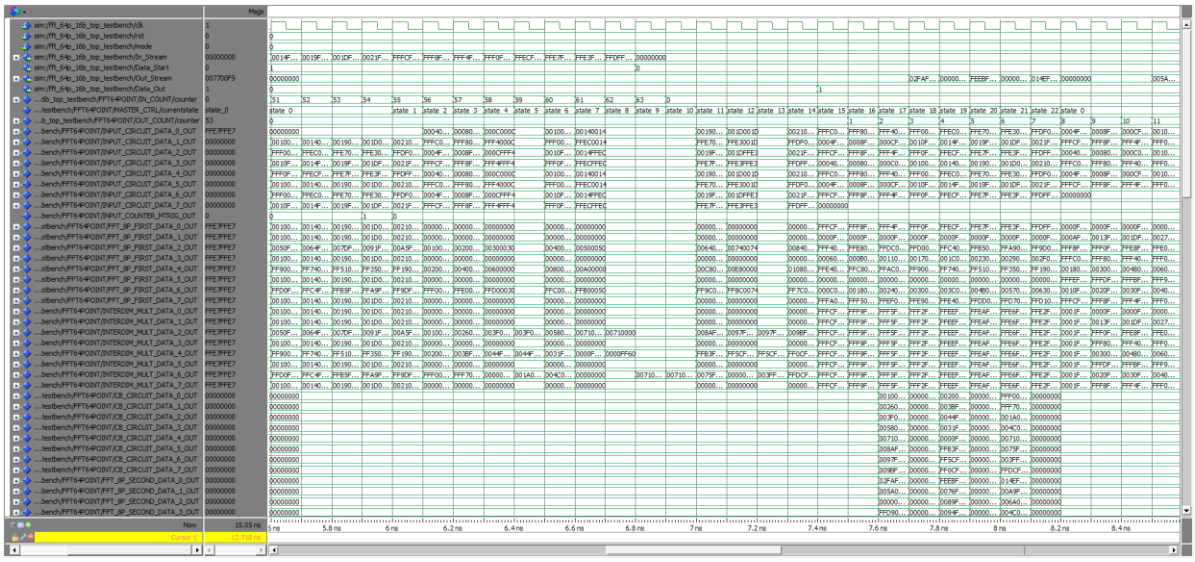


Diagram Waveform mulai Pemrosesan Data

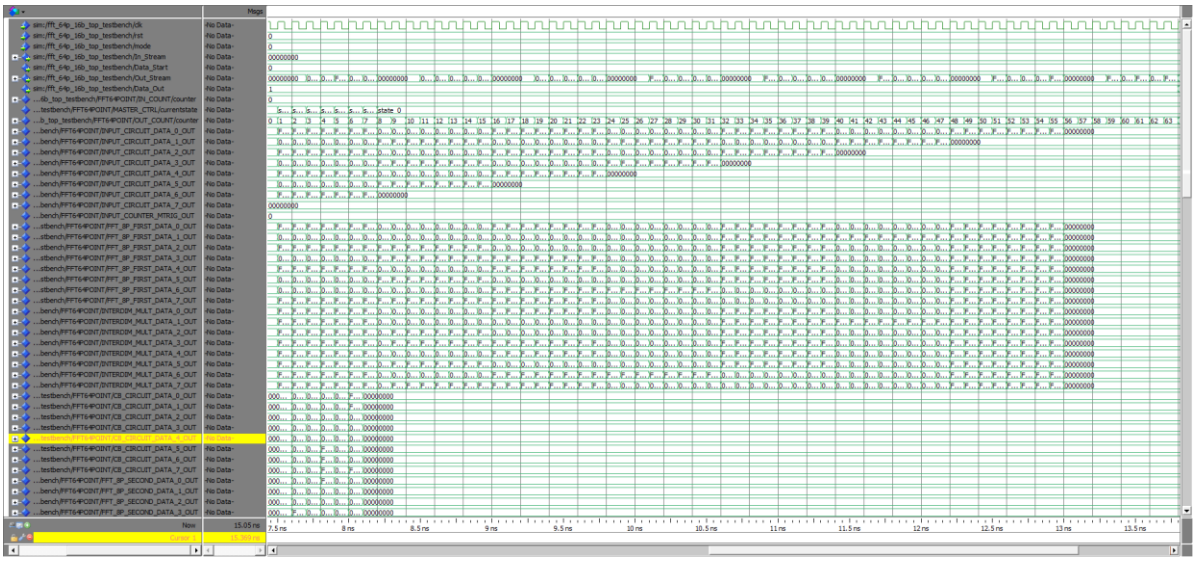


Diagram Waveform Output Data

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

K. Test Bench dan Pengujian

Untuk melakukan pengujian rangkaian ini, digunakan testbench sebagai berikut. Hal ini diperlukan karena input data yang diberikan cukup banyak (64 buah data) sehingga dengan adanya testbench ini, proses pengujian dapat dilakukan lebih mudah. Uji coba dilakukan membandingkan hasil perhitungan menggunakan MATLAB dengan algoritma yang telah diberikan pada percobaan sebelumnya untuk dibandingkan dengan hasil output rangkaian.

fft_64p_16b_top_testbench.vhd

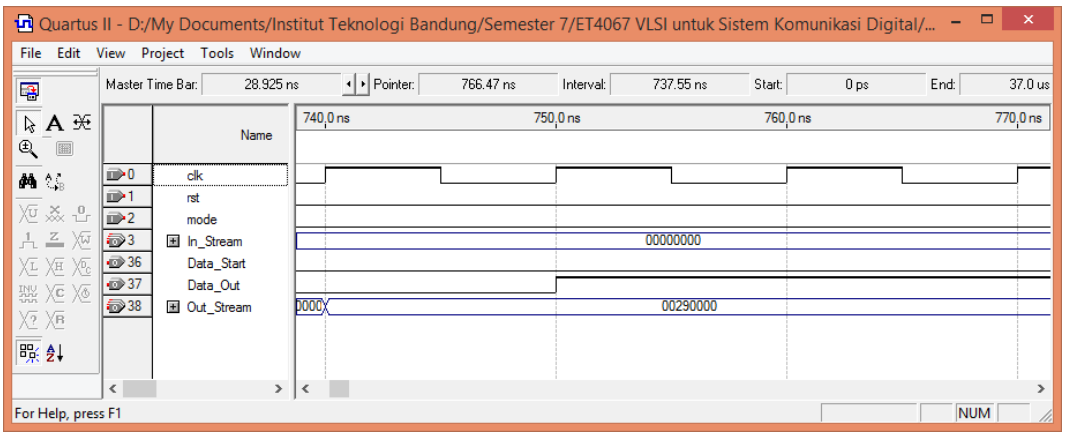
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fft_64p_16b_top_testbench IS
  PORT   (
              In_Stream    :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
              Data_Start   :      OUT    STD_LOGIC;
              clk          :      IN     STD_LOGIC;
              rst          :      IN     STD_LOGIC;
              mode         :      IN     STD_LOGIC;
              Out_Stream   :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
              Data_Out     :      OUT    STD_LOGIC
          );
END fft_64p_16b_top_testbench;

ARCHITECTURE structural of fft_64p_16b_top_testbench IS
COMPONENT fft_64p_16b_top IS
  PORT   (
              In_Stream    :      IN     STD_LOGIC_VECTOR(31 DOWNTO 0);
              Mode         :      IN     STD_LOGIC;
              Data_Start   :      IN     STD_LOGIC;
              clk          :      IN     STD_LOGIC;
              rst          :      IN     STD_LOGIC;
              Out_Stream   :      OUT    STD_LOGIC_VECTOR(31 DOWNTO 0);
              Data_Out     :      OUT    STD_LOGIC
          );
END COMPONENT;

TYPE TABLE_TYPE IS ARRAY (0 TO 63) of STD_LOGIC_VECTOR (31 DOWNTO 0);
TYPE FSMSTATE IS (
  aftreset1, aftreset2, aftreset3, counting, waiting
  );
CONSTANT INPUT_STREAM : TABLE_TYPE :=
(
  x"00040004", --1
  x"00080008", --2
  x"000C000C", --3
  x"00100010", --4
  x"00140014", --5
  x"00190019", --6
  x"001D001D", --7
  x"00210021", --8
  x"FFFC0004", --9
  x"FFF80008", --10
  x"FFF4000C", --11
  x"FFF00010", --12
  x"FFEC0014", --13
  x"FFE70019", --14
  x"FFE3001D", --15
  x"FFDF0021", --16
  x"0004FFFC", --17
  x"0008FFF8", --18
  x"000CFFF4", --19
  x"0010FFF0", --20
  x"0014FFEC", --21
  x"0019FFE7", --22
  x"001DFFE3", --23
  x"0021FFDF", --24
```

```
   x"FFFCFFFC", --25
   x"FFF8FFF8", --26
   x"FFF4FFF4", --27
   x"FFF0FFF0", --28
   x"FFECFFEC", --29
   x"FFE7FFE7", --30
   x"FFE3FFE3", --31
   x"FFDFFFDF", --32
   x"00040004", --33
   x"00080008", --34
   x"000C000C", --35
   x"00100010", --36
   x"00140014", --37
   x"00190019", --38
   x"001D001D", --39
   x"00210021", --40
   x"FFFC0004", --41
   x"FFF80008", --42
   x"FFF4000C", --43
   x"FFF00010", --44
   x"FFEC0014", --45
   x"FFE70019", --46
   x"FFE3001D", --47
   x"FFDF0021", --48
   x"0004FFFC", --49
   x"0008FFF8", --50
   x"000CFFF4", --51
   x"0010FFF0", --52
   x"0014FFEC", --53
   x"0019FFE7", --54
   x"001DFFE3", --55
   x"0021FFDF", --56
   x"FFFCFFFC", --57
   x"FFF8FFF8", --58
   x"FFF4FFF4", --59
   x"FFF0FFF0", --60
   x"FFECFFEC", --61
   x"FFE7FFE7", --62
   x"FFE3FFE3", --63
   x"FFDFFFDF"  --64
);
SIGNAL counter      : INTEGER;
SIGNAL D_Start      : STD_LOGIC;
SIGNAL D_Stream     : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL currentstate: FSMSTATE;
  BEGIN
  In_Stream <= D_Stream;
  Data_Start<= D_Start;
  PROCESS(clk,rst,counter,currentstate)
  BEGIN
        IF rst='1' THEN
          counter <= 0;
          currentstate <= aftreset1;
          D_Stream<= x"00000000";
        ELSE
          IF ((clk'EVENT) AND (clk='1'))  THEN
              CASE currentstate IS
                        WHEN aftreset1   =>
                         D_Start <= '0';
                         D_Stream<= x"00000000";
                         counter <= 0;
                         currentstate <= aftreset2;
                        WHEN aftreset2   =>
                         D_Start <= '0';
                         D_Stream<= x"00000000";
                         counter <= 0;
                         currentstate <= aftreset3;
                        WHEN aftreset3   =>
```

```
                        D_Start <= '0';
                        D_Stream<= x"00000000";
                        counter <= 0;
                        currentstate <= counting;
                      WHEN counting   =>
                       IF (counter=0) THEN
                              D_Start <= '1';
                              D_Stream<= INPUT_STREAM(counter);
                              counter <= counter + 1;
                              currentstate <= counting;
                       ELSIF (counter>0) AND (counter<64) THEN
                              D_Start <= '1';
                              D_Stream<= INPUT_STREAM(counter);
                              counter <= counter + 1;
                              currentstate <= counting;
                       ELSE
                              D_Start <= '0';
                              D_Stream<= x"00000000";
                              counter <= 0;
                              currentstate <= waiting;
                       END IF;
                      WHEN waiting   =>
                              D_Start <= '0';
                              D_Stream<= x"00000000";
                              counter <= 0;
                              currentstate <= waiting;
            END CASE;
         END IF;
       END IF;
END PROCESS;

FFT64POINT :
       fft_64p_16b_top
        PORT MAP (
                        In_Stream     =>D_Stream,
                        Mode          =>mode,
                        Data_Start    =>D_start,
                        clk               =>clk,
                        rst               =>rst,
                        Out_Stream    =>Out_Stream,
                        Data_Out      =>Data_Out
                        );
END structural;
```

- **Uji coba I (FFT – Data Dasar)**

  Uji coba pertama ini dilakukan dengan memberikan data $0{,}01 + j0$ pada $B(0)$ dan membiarkan data yang lain kosong. Operasi yang dicoba adalah FFT.



Waveform pada Simulasi dengan Altera Quartus II 9.1sp2

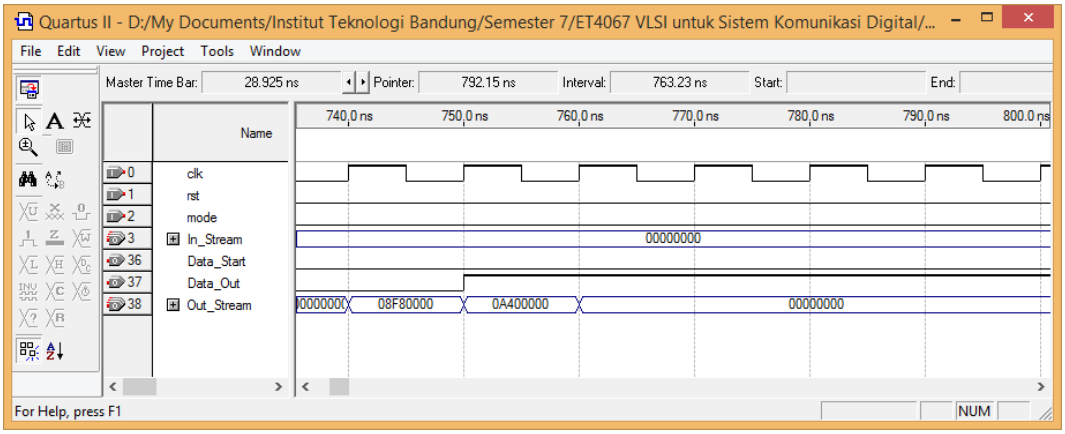
B64_Fixed | A64_Fixed

64x1 cell

	1	2	3	4	5	6	7	8	9
1	00290000								
2	00290000								
3	00290000								
4	00290000								
5	00290000								
6	00290000								
7	00290000								
8	00290000								
9	00290000								
10	00290000								
11	00290000								
12	00290000								
13	00290000								
14	00290000								

Hasil Perhitungan menggunakan MATLAB

Uji coba kedua ini dilakukan dengan memberikan data $0{,}01 + j0$ pada $B(0)$ hingga $B(63)$. Operasi yang dicoba adalah FFT.



Waveform pada Simulasi dengan Altera Quartus II 9.1sp2

Editor - fft64point.m

B64_Fixed | A64_Fixed

64x1 cell

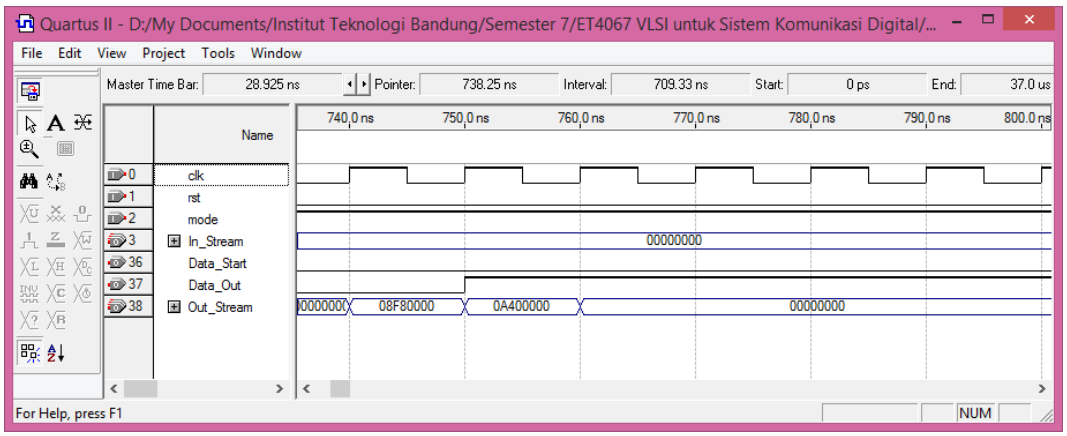
	1	2	3	4	5	6	7	8	9
1	0A3D0000								
2	00000000								
3	00000000								
4	00000000								
5	00000000								
6	00000000								
7	00000000								
8	00000000								
9	00000000								
10	00000000								
11	00000000								
12	00000000								
13	00000000								
14	00000000								

Hasil Perhitungan menggunakan MATLAB

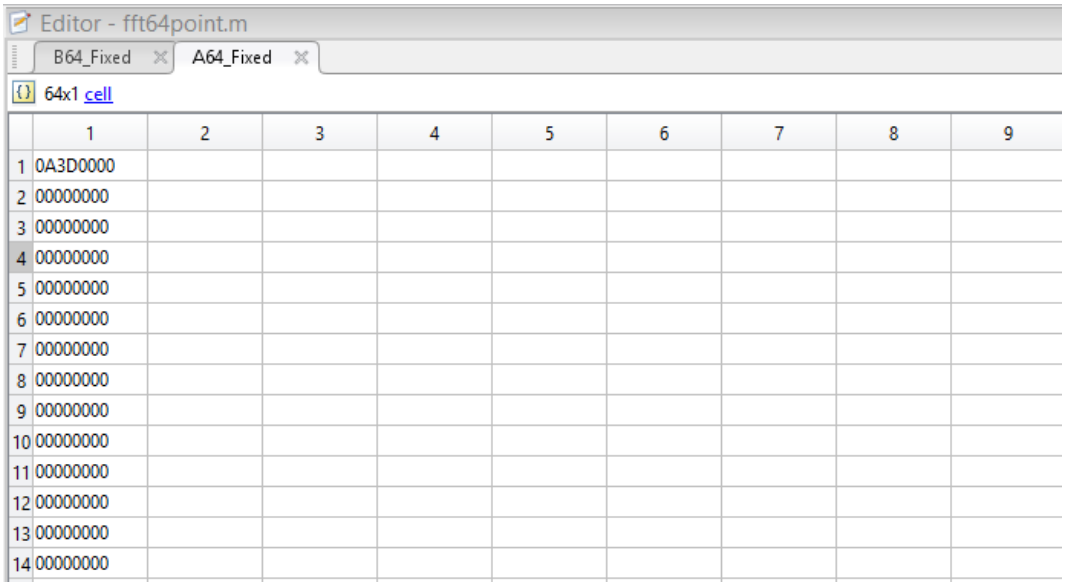
+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

- **Uji coba II (IFFT – Data Dasar) (Hasil akhir harus dibagi panjang FFT = 64)**

  Uji coba pertama ini dilakukan dengan memberikan data $0.01 + j0$ untuk $B(0)$ hingga $B(63)$. Operasi yang dicoba adalah IFFT.



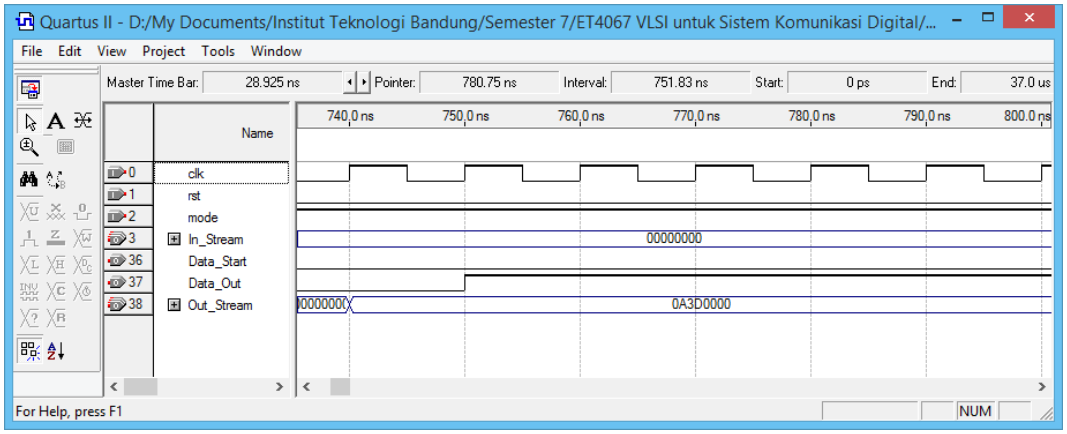
Waveform pada Simulasi dengan Altera Quartus II 9.1sp2



	1	2	3	4	5	6	7	8	9
1	0A3D0000								
2	00000000								
3	00000000								
4	00000000								
5	00000000								
6	00000000								
7	00000000								
8	00000000								
9	00000000								
10	00000000								
11	00000000								
12	00000000								
13	00000000								
14	00000000								

Hasil Perhitungan menggunakan MATLAB

Uji coba kedua ini dilakukan dengan memberikan data $0{,}64 + j0$ pada $B(0)$ dan membiarkan data yang lain kosong. Operasi yang dicoba adalah IFFT.



Waveform pada Simulasi dengan Altera Quartus II 9.1sp2

Editor - fft64point.m

B64_Fixed | A64_Fixed | A64

64x1 cell

	1	2	3	4	5	6	7	8	9
1	0A3D0000								
2	0A3D0000								
3	0A3D0000								
4	0A3D0000								
5	0A3D0000								
6	0A3D0000								
7	0A3D0000								
8	0A3D0000								
9	0A3D0000								
10	0A3D0000								
11	0A3D0000								
12	0A3D0000								
13	0A3D0000								
14	0A3D0000								

Hasil Perhitungan menggunakan MATLAB

- **Uji coba III (FFT dan IFFT – Data Arbitrary)**

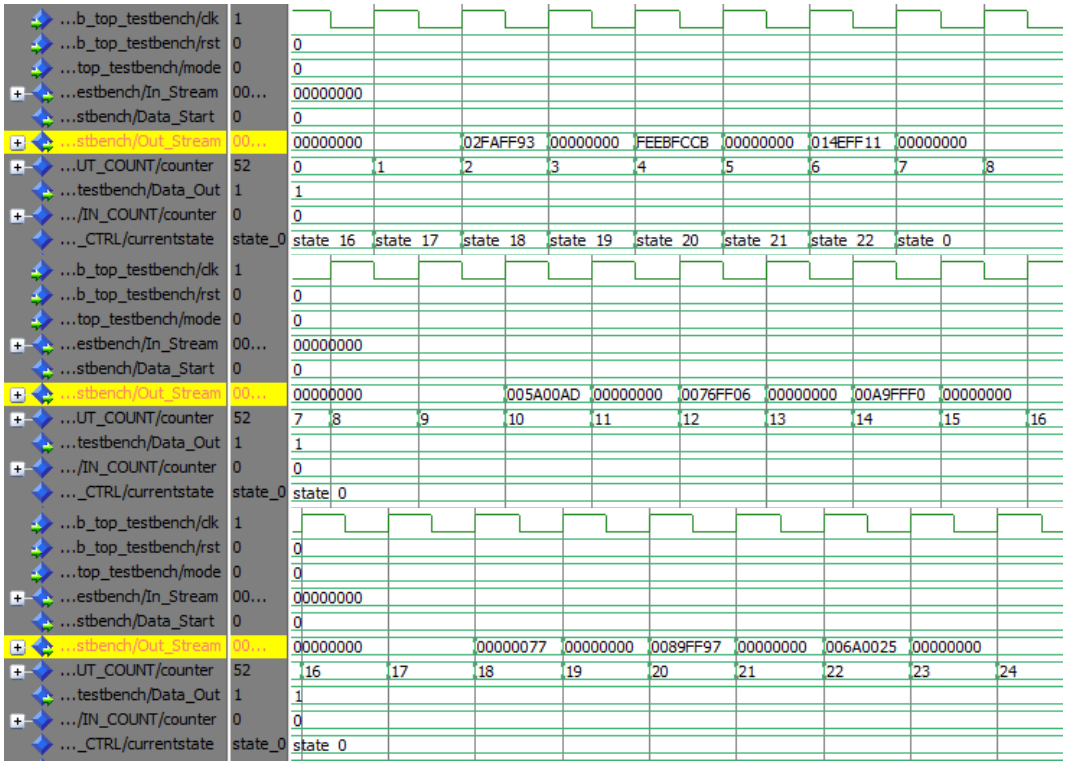
  Uji coba ini dilakukan dengan memberikan data arbitrary pada input FFT untuk dioperasikan menggunakan FFT 64-titik. Selanjutnya, akan diujicoba operasi IFFT 64-titik dengan data yang berasal dari FFT 64-titik yang dilakukan sebelumnya. Dengan demikian, pembandingan dapat dilakukan. Input data pada MATLAB diberikan sebagai berikut.

```
B64(1)  = 0.001 + j*0.001;
B64(2)  = 0.002 + j*0.002;
B64(3)  = 0.003 + j*0.003;
B64(4)  = 0.004 + j*0.004;
B64(5)  = 0.005 + j*0.005;
B64(6)  = 0.006 + j*0.006;
B64(7)  = 0.007 + j*0.007;
B64(8)  = 0.008 + j*0.008;
B64(9)  = -0.001 + j*0.001;
B64(10) = -0.002 + j*0.002;
B64(11) = -0.003 + j*0.003;
B64(12) = -0.004 + j*0.004;
B64(13) = -0.005 + j*0.005;
B64(14) = -0.006 + j*0.006;
B64(15) = -0.007 + j*0.007;
B64(16) = -0.008 + j*0.008;
B64(17) = 0.001 - j*0.001;
B64(18) = 0.002 - j*0.002;
B64(19) = 0.003 - j*0.003;
B64(20) = 0.004 - j*0.004;
B64(21) = 0.005 - j*0.005;
B64(22) = 0.006 - j*0.006;
B64(23) = 0.007 - j*0.007;
B64(24) = 0.008 - j*0.008;
B64(25) = -0.001 - j*0.001;
B64(26) = -0.002 - j*0.002;
B64(27) = -0.003 - j*0.003;
B64(28) = -0.004 - j*0.004;
B64(29) = -0.005 - j*0.005;
B64(30) = -0.006 - j*0.006;
B64(31) = -0.007 - j*0.007;
B64(32) = -0.008 - j*0.008;
B64(33) = 0.001 + j*0.001;
B64(34) = 0.002 + j*0.002;
B64(35) = 0.003 + j*0.003;
B64(36) = 0.004 + j*0.004;
```

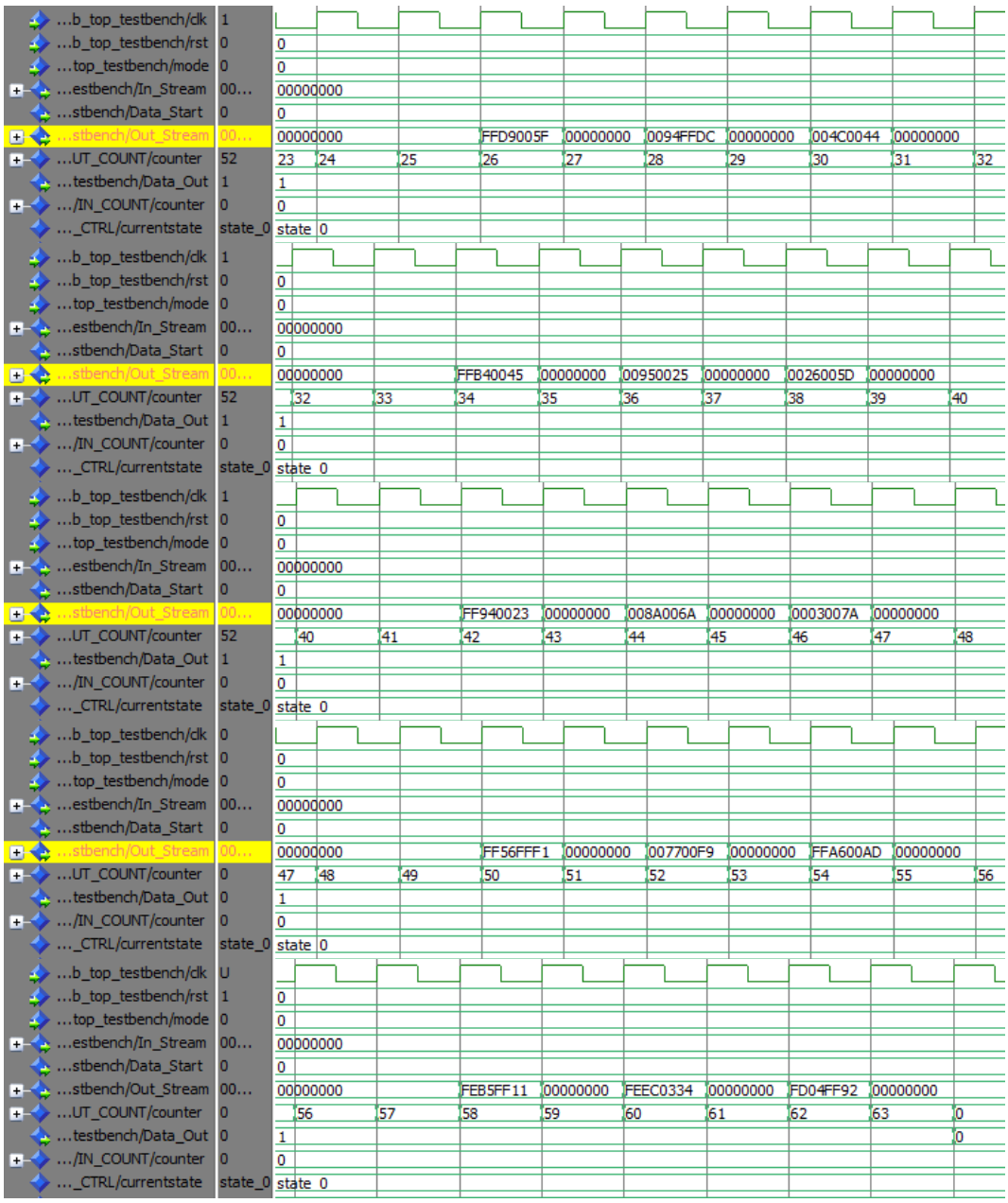
```
B64(37) = 0.005 + j*0.005;
B64(38) = 0.006 + j*0.006;
B64(39) = 0.007 + j*0.007;
B64(40) = 0.008 + j*0.008;
B64(41) = -0.001 + j*0.001;
B64(42) = -0.002 + j*0.002;
B64(43) = -0.003 + j*0.003;
B64(44) = -0.004 + j*0.004;
B64(45) = -0.005 + j*0.005;
B64(46) = -0.006 + j*0.006;
B64(47) = -0.007 + j*0.007;
B64(48) = -0.008 + j*0.008;
B64(49) = 0.001 - j*0.001;
B64(50) = 0.002 - j*0.002;
B64(51) = 0.003 - j*0.003;
B64(52) = 0.004 - j*0.004;
B64(53) = 0.005 - j*0.005;
B64(54) = 0.006 - j*0.006;
B64(55) = 0.007 - j*0.007;
B64(56) = 0.008 - j*0.008;
B64(57) = -0.001 - j*0.001;
B64(58) = -0.002 - j*0.002;
B64(59) = -0.003 - j*0.003;
B64(60) = -0.004 - j*0.004;
B64(61) = -0.005 - j*0.005;
B64(62) = -0.006 - j*0.006;
B64(63) = -0.007 - j*0.007;
B64(64) = -0.008 - j*0.008;
```

- FFT

  Berikut ini adalah waveform output hasil pengujian yang dilakukan. Pembandingan hasil menggunakan Fixed Point Q4.12 baik pada MATLAB maupun pada rangkaian yang diujicoba.

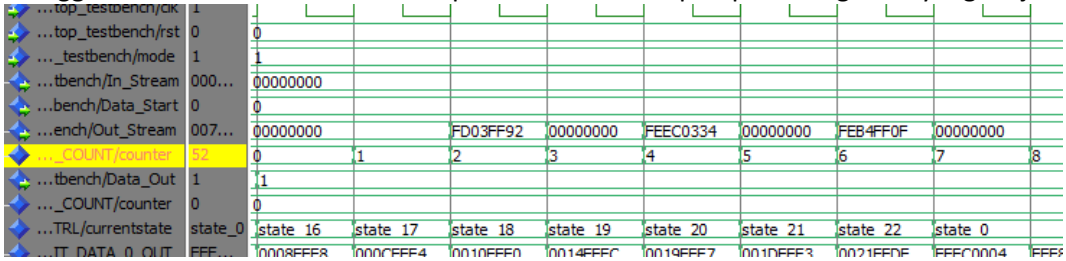


+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id



- IFFT (hasil akhir harus dibagi 64)

  Berikut ini adalah waveform output hasil pengujian yang dilakukan. Pembandingan hasil menggunakan Fixed Point Q4.12 baik pada MATLAB maupun pada rangkaian yang diujicoba.



+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

+6281395141700 | hanindhito@bagus.my.id | http://www.bagus.my.id

- Perbandingan Data

	FFT		IFFT	
A	**MATLAB**	**Rangkaian**	**MATLAB**	**Rangkaian**
$A[0]$	00000000	00000000	00000000	00000000
$A[1]$	00000000	00000000	00000000	00000000
$A[2]$	02FEFF97	02FAFF93	FD02FF97	FD03FF92
$A[3]$	00000000	00000000	00000000	00000000
$A[4]$	FEF5FCC8	FEEBFCCB	FEF50338	FEEC0334
$A[5]$	00000000	00000000	00000000	00000000
$A[6]$	0150FF19	014EFF11	FEB0FF19	FEB4FF0F
$A[7]$	00000000	00000000	00000000	00000000
$A[8]$	00000000	00000000	00000000	00000000
$A[9]$	00000000	00000000	00000000	00000000
$A[10]$	005800A8	005A00AD	FFA800A8	FFA500AE
$A[11]$	00000000	00000000	00000000	00000000
$A[12]$	006FFF0B	0076FF06	006F00F5	007600F9
$A[13]$	00000000	00000000	00000000	00000000
$A[14]$	00A5FFEE	00A9FFF0	FF5BFFEE	FF58FFF0
$A[15]$	00000000	00000000	00000000	00000000
$A[16]$	00000000	00000000	00000000	00000000
$A[17]$	00000000	00000000	00000000	00000000
$A[18]$	0002007A	00000077	FFFE007A	00030078
$A[19]$	00000000	00000000	00000000	00000000
$A[20]$	008CFF93	0089FF97	008C006D	008A006A
$A[21]$	00000000	00000000	00000000	00000000
$A[22]$	006D0024	006A0025	FF930024	FF940021
$A[23]$	00000000	00000000	00000000	00000000
$A[24]$	00000000	00000000	00000000	00000000
$A[25]$	00000000	00000000	00000000	00000000
$A[26]$	FFD7005D	FFD9005F	0029005D	0025005E
$A[27]$	00000000	00000000	00000000	00000000
$A[28]$	0093FFDF	0094FFDC	00930021	00960024
$A[29]$	00000000	00000000	00000000	00000000
$A[30]$	004A0043	004C0044	FFB60043	FFB20045
$A[31]$	00000000	00000000	00000000	00000000
$A[32]$	00000000	00000000	00000000	00000000
$A[33]$	00000000	00000000	00000000	00000000
$A[34]$	FFB60043	FFB40045	004A0043	004D0044
$A[35]$	00000000	00000000	00000000	00000000
$A[36]$	00930021	00950025	0093FFDF	0094FFDC
$A[37]$	00000000	00000000	00000000	00000000
$A[38]$	0029005D	0026005D	FFD7005D	FFDA0061
$A[39]$	00000000	00000000	00000000	00000000
$A[40]$	00000000	00000000	00000000	00000000
$A[41]$	00000000	00000000	00000000	00000000
$A[42]$	FF930024	FF940023	006D0024	006B0024
$A[43]$	00000000	00000000	00000000	00000000
$A[44]$	008C006D	008A006A	008CFF93	008AFF97
$A[45]$	00000000	00000000	00000000	00000000
$A[46]$	FFFE007A	0003007A	0002007A	FFFE0078
$A[47]$	00000000	00000000	00000000	00000000
$A[48]$	00000000	00000000	00000000	00000000
$A[49]$	00000000	00000000	00000000	00000000
$A[50]$	FF5BFFEE	FF56FFF1	00A5FFEE	00A9FFF2
$A[51]$	00000000	00000000	00000000	00000000
$A[52]$	006F00F5	007700F9	006FFF0B	0076FF06
$A[53]$	00000000	00000000	00000000	00000000
$A[54]$	FFA800A8	FFA600AD	005800A8	005A00AF
$A[55]$	00000000	00000000	00000000	00000000
$A[56]$	00000000	00000000	00000000	00000000
$A[57]$	00000000	00000000	00000000	00000000
$A[58]$	FEB0FF19	FEB5FF11	0150FF19	014FFF10
$A[59]$	00000000	00000000	00000000	00000000
$A[60]$	FEF50338	FEEC0334	FEF5FCC8	FEEAFCCC
$A[61]$	00000000	00000000	00000000	00000000
$A[62]$	FD02FF97	FD04FF92	02FEFF97	02FCFF93
$A[63]$	00000000	00000000	00000000	00000000

- **Analisis Hasil Ujicoba**

  Ujicoba yang dilakukan memberikan hasil yang positif bahwa rangkaian prosesor FFT/IFFT 64-titik telah berfungsi dengan benar.

  - Ujicoba pertama dan kedua dilakukan dengan data-data dasar. Hasil ujicoba memberikan hasil yang baik. Prosesor FFT/IFFT 64-titik yang dirancang dapat berjalan dengan baik pada kedua mode operasi yaitu FFT atau IFFT.

    Dalam ujicoba pertama dan kedua, FFT mentransformasikan satu set data sangat sederhana. Kemudian hasil transformasi data tersebut diolah menggunakan IFFT sehingga diperoleh set data yang merupakan input FFT di awal tadi. Hal ini dapat dilihat dengan jelas bahwa proses transformasi maju dan transformasi mundur dapat berjalan dengan baik. **Dengan mengingat rumus berikut, data pada operasi IFFT harus dibagi dengan panjang dari FFT tersebut yaitu 64.** Cara ini sangat mudah dilakukan yaitu menambahkan right shifter di tahap akhir output untuk operasi IFFT.

$$B[n] = \frac{1}{N}\left[\sum_{k=0}^{N-1} A^*[k]e^{-j2\pi kn/N}\right]^*$$

  - Ujicoba ketiga dilakukan dengan data arbitrary. Hasil ujicoba menunjukkan bahwa fungsionalitas prosesor FFT/IFFT telah benar. Terjadi sedikit perbedaan antara data yang diolah dengan MATLAB dengan data yang diolah dengan prosesor FFT/IFFT 64-titik ini. Perbedaan terjadi maksimum hingga 3-bit LSB. Hal ini disebabkan oleh akumulasi error dalam perhitungan terutama perkalian. Pada perkalian, terjadi pemotongan bit yang berakibat pada kurang presisinya hasil perkalian yang dilakukan. Oleh karena itu, implementasi rounding menjadi sangat penting di sini.

L. Revisi Desain

Revisi desain dilakukan secara minor, yaitu penambahan unit scalling pada tahap output agar operasi IFFT 64-titik dapat dilakukan secara penuh. Unit scalling ini terletak setelah unit swapping dan berfungsi untuk membagi hasil perhitungan IFFT dengan panjang FFT/IFFT yang dilakukan. Dalam kasus ini adalah 64. Modifikasi rangkaian tersebut diberikan pada kode sebagai berikut. Rangkaian scalling terdiri atas right shifter sebanyak enam kali dan sebuah multiplexer.

rshift_6_16b_to_16b.vhd

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

ENTITY rshift_6_16b_to_16b IS
   PORT (
            Data_In    :  IN     std_logic_vector (15 DOWNTO 0);
            Data_Out   :  OUT    std_logic_vector (15 DOWNTO 0)
         );
END rshift_6_16b_to_16b;

ARCHITECTURE structural of rshift_6_16b_to_16b IS
BEGIN
        Data_Out <=     Data_In(15) & Data_In(15) & Data_In(15) &
                        Data_In(15) & Data_In(15) & Data_In(15) &
                        Data_In (15 DOWNTO 6);
END structural;
```

scalling_out.vhd

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

ENTITY scalling_out IS
   PORT (
```

```
            A32     :       IN      STD_LOGIC_VECTOR (31 DOWNTO 0);
            Swap    :       IN      STD_LOGIC;
            R32     :       OUT     STD_LOGIC_VECTOR (31 DOWNTO 0)
         );
END scalling_out;

ARCHITECTURE structural OF scalling_out IS
COMPONENT mux_2to1_32b IS
   PORT (
            D1      :       IN      std_logic_vector (31 DOWNTO 0);
            D2      :       IN      std_logic_vector (31 DOWNTO 0);
            Y       :       OUT     std_logic_vector (31 DOWNTO 0);
            S       :       IN      std_logic
         );
END COMPONENT;
COMPONENT rshift_6_16b_to_16b IS
   PORT (
            Data_In :       IN      std_logic_vector (15 DOWNTO 0);
            Data_Out:       OUT     std_logic_vector (15 DOWNTO 0)
         );
END COMPONENT;
SIGNAL REAL_A32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_A32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_R32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_R32             : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL MUX_0_OUT            : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL REAL_SCALE           : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL IMAG_SCALE           : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL TEMP_SCALE           : STD_LOGIC_VECTOR (31 DOWNTO 0);

BEGIN
   REAL_A32                         <= A32(31 DOWNTO 16);
   IMAG_A32                         <= A32(15 DOWNTO 0);
   REAL_R32                         <= REAL_SCALE;
   IMAG_R32                         <= IMAG_SCALE;
   TEMP_SCALE(31 DOWNTO 16)         <= REAL_R32;
   TEMP_SCALE(15 DOWNTO 0)          <= IMAG_R32;
   -- Port Mapping
   SCALE_0 :
        rshift_6_16b_to_16b
         PORT MAP
            (
             Data_In  =>REAL_A32,
             Data_Out =>REAL_SCALE
            );
   SCALE_1 :
        rshift_6_16b_to_16b
         PORT MAP
            (
             Data_In  =>IMAG_A32,
             Data_Out =>IMAG_SCALE
            );
   MUX_0 :
        mux_2to1_32b
         PORT MAP
            (
             D1     =>A32,
             D2     =>TEMP_SCALE,
             Y      =>R32,
             S      =>Swap
            );
END structural;
```

output_circuit.vhd

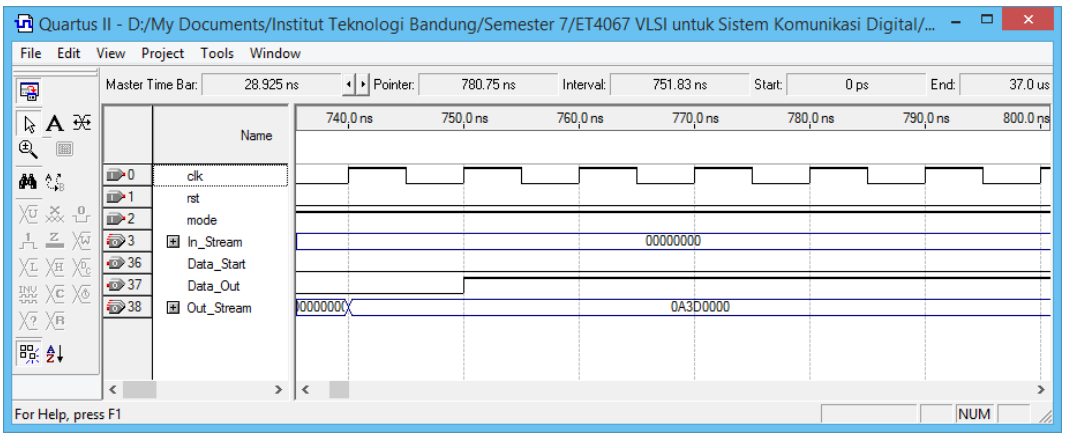
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY output_circuit IS
```

```
   PORT (
              D1                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D2                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D3                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D4                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D5                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D6                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D7                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              D8                :     IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk               :     IN  STD_LOGIC;
              hold_all_seg      :     IN  STD_LOGIC;
              in_ctrl_all_seg   :     IN  STD_LOGIC;
              mode              :    IN   STD_LOGIC;
              rst               :     IN   STD_LOGIC;
              Q                 :     OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
END output_circuit;

ARCHITECTURE structural of output_circuit IS
COMPONENT dff_segment_for_output IS
   PORT (
              D7     : IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
              D1     : IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk    : IN   STD_LOGIC;
              hold   : IN   STD_LOGIC;
              in_sel : IN   STD_LOGIC;
              rst    : IN   STD_LOGIC;
              Q      : OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
END COMPONENT;

COMPONENT dff_with_hold_input_ctrl IS
   PORT (
              D1     : IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
              D2     : IN   STD_LOGIC_VECTOR (31 DOWNTO 0);
              clk    : IN   STD_LOGIC;
              hold   : IN   STD_LOGIC;
              in_sel : IN   STD_LOGIC;
              rst    : IN   STD_LOGIC;
              Q      : OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
END COMPONENT;
COMPONENT real_imaginary_interchange IS
   PORT (
              A32   : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
              Swap  : IN     STD_LOGIC;
              R32   : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
END COMPONENT;
COMPONENT scalling_out IS
   PORT (
              A32   : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
              Swap  : IN     STD_LOGIC;
              R32   : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
END COMPONENT;

SIGNAL SWAP_OUT         :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SCALE_OUT        :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL LEAD_BUF_OUT     :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_0_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_1_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_2_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_3_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_4_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_5_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SEGMENT_6_OUT    :    STD_LOGIC_VECTOR (31 DOWNTO 0);
   BEGIN
```

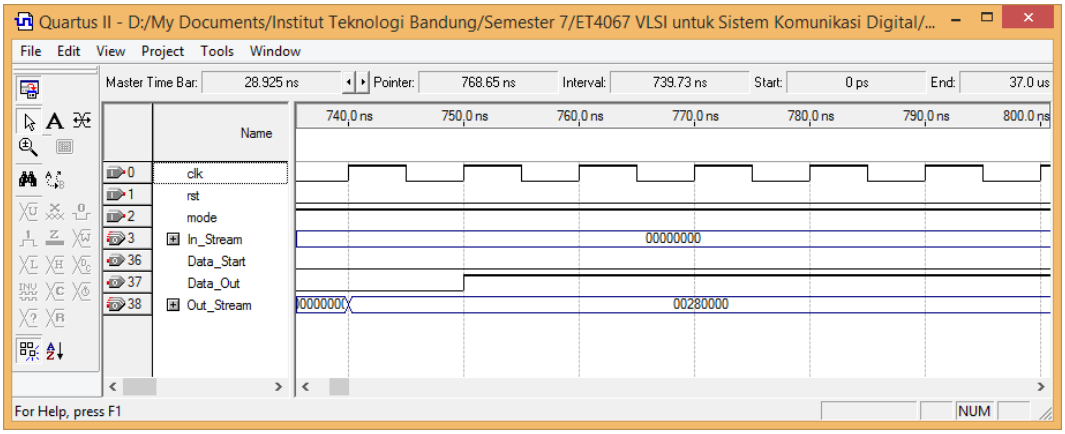
```
    Q                <=      SCALE_OUT;
    -- Port Map
    LEAD_BUF :
          dff_with_hold_input_ctrl
              PORT MAP (
                    D1      =>std_logic_vector(to_unsigned(0,32)),
                    D2      =>D8,
                    clk     =>clk,
                    hold    =>hold_all_seg,
                    in_sel  =>in_ctrl_all_seg,
                    rst     =>rst,
                    Q       =>LEAD_BUF_OUT
                   );
    SEGMEN_6 :
          dff_segment_for_output
              PORT MAP (
                    D7      =>LEAD_BUF_OUT,
                    D1      =>D7,
                    clk     =>clk,
                    hold    =>hold_all_seg,
                    in_sel  =>in_ctrl_all_seg,
                    rst     =>rst,
                    Q       =>SEGMENT_6_OUT
                   );
    SEGMEN_5 :
          dff_segment_for_output
              PORT MAP (
                    D7      =>SEGMENT_6_OUT,
                    D1      =>D6,
                    clk     =>clk,
                    hold    =>hold_all_seg,
                    in_sel  =>in_ctrl_all_seg,
                    rst     =>rst,
                    Q       =>SEGMENT_5_OUT
                   );
    SEGMEN_4 :
          dff_segment_for_output
              PORT MAP (
                    D7      =>SEGMENT_5_OUT,
                    D1      =>D5,
                    clk     =>clk,
                    hold    =>hold_all_seg,
                    in_sel  =>in_ctrl_all_seg,
                    rst     =>rst,
                    Q       =>SEGMENT_4_OUT
                   );
    SEGMEN_3 :
          dff_segment_for_output
              PORT MAP (
                    D7      =>SEGMENT_4_OUT,
                    D1      =>D4,
                    clk     =>clk,
                    hold    =>hold_all_seg,
                    in_sel  =>in_ctrl_all_seg,
                    rst     =>rst,
                    Q       =>SEGMENT_3_OUT
                   );
    SEGMEN_2 :
          dff_segment_for_output
              PORT MAP (
                    D7      =>SEGMENT_3_OUT,
                    D1      =>D3,
                    clk     =>clk,
                    hold    =>hold_all_seg,
                    in_sel  =>in_ctrl_all_seg,
                    rst     =>rst,
                    Q       =>SEGMENT_2_OUT
                   );
```

```
    SEGMEN_1 :
         dff_segment_for_output
             PORT MAP (
                    D7       =>SEGMENT_2_OUT,
                    D1       =>D2,
                    clk      =>clk,
                    hold     =>hold_all_seg,
                    in_sel   =>in_ctrl_all_seg,
                    rst      =>rst,
                    Q        =>SEGMENT_1_OUT
                   );
    SEGMEN_0 :
         dff_segment_for_output
             PORT MAP (
                    D7       =>SEGMENT_1_OUT,
                    D1       =>D1,
                    clk      =>clk,
                    hold     =>hold_all_seg,
                    in_sel   =>in_ctrl_all_seg,
                    rst      =>rst,
                    Q        =>SEGMENT_0_OUT
                   );
     SWAP :
         real_imaginary_interchange
             PORT MAP (
                    A32=>SEGMENT_0_OUT,
                    Swap=>mode,
                    R32 =>SWAP_OUT
                   );
    SCALE :
         scalling_out
             PORT MAP (
                    A32=>SWAP_OUT,
                    Swap=>mode,
                    R32 =>SCALE_OUT
                   );
    END structural;
```

Hasil modifikasi tersebut kemudian dibandingkan perubahannya. Berikut ini adalah perbandingan antara uji coba kedua pada IFFT sebelum penambahan blok scaling dan sesudah penambahan blok scaling.



Sebelum Penambahan Blok Scaling



Setelah Penambahan Blok Scaling

Input		Intermediate		Output
B(0)..B(63) = 00290000	➔FFT➔	A(0) = 0A400000 ELSE = 00000000	➔IFFT➔	B'(0)..B'(63) = 00290000

Dengan demikian terlihat bahwa operasi IFFT dapat dilakukan dengan baik pada prosesor FFT/IFFT 64-titik yang telah diimplementasikan ini.

M. Kesimpulan

- Rangkaian prosesor FFT/IFFT 64-point yang telah diimplementasikan sesuai dengan paper yang disebutkan pada awal laporan ini dapat berfungsi dengan baik untuk kedua operasi FFT dan IFFT.
- Operasi FFT 64-titik dan IFFT 64-titik dapat dilakukan dalam waktu 23 clock cycle dengan periode clock minimum sebesar 70ns untuk clock 14MHz. Hal ini membuat operasi FFT 64-titik dan IFFT 64-titik dapat dilakukan dalam waktu 1,6µs sehingga prosesor FFT/IFFT 64-point ini sesuai dengan spesifikasi minimum WLAN 802.11a yang mengharuskan operasi FFT/IFFT 64-point diselesaikan dalam waktu 4µs.
- Diperlukan revisi cukup sederhana pada bagian tahap akhir output untuk membagi hasil yang dikeluarkan dari mode IFFT dengan panjang FFT/IFFT yang bersangkutan yaitu 64. Revisi sederhana ini dilakukan dengan menambahkan right shifter sebanyak enam kali. Revisi ini telah berhasil dilakukan.
- Diperlukan improvisasi terhadap sistem rounding agar diperoleh hasil yang lebih akurat.