

**TUGAS AKHIR *METODA FORMAL*(EC 7030)**

**Verifikasi Radix-4 Pipelined 16-point Complex FFT  
CORE Dengan Menggunakan HOL Theorem Proving**

**Oleh:**

**DECKY SAMUEL (23203090)**

**DOSEN : DR.IR. BUDI RAHARJO**



**MAGISTER TEKNIK KOMPUTER  
JURUSAN TEKNIK ELEKTRO  
INSTITUT TEKNOLOGI BANDUNG  
2004**

# DAFTAR ISI

## DAFTAR ISI

- 1 PENDAHULUAN
  - 2 ANALISA ERROR DARI FAST FOURIER TRANSFORM
  - 3 FFT FIXED POINT UNTUK VERIFIKASI NETLIST
  - 4 KESIMPULAN
  - A DAFTAR DARI DEFINISI DAN LEMMAS DALAM HOL
- DAFTAR PUSTAKA

Zeon PDF Driver Trial  
www.zeon.com.tw

# Verifikasi Radix-4 Pipelined 16-point Complex FFT CORE Dengan Menggunakan HOL Theorem Proving

## Abstrak

Fast Fourier Transform (FFT) adalah suatu algoritma untuk menghitung discrete Fourier Transform (DFT) yang secara substansial dapat menyimpan waktu yang lebih dari pada metoda yang konvensional. Dua kelas dasar dari algoritma FFT adalah decimation-in-time (DIT) dan decimation-in-frequency (DIF). Verifikasi FFT telah dilakukan oleh Akbarpour dan Tahar dengan menggunakan HOL Theorem Proving. Pada laporan ini kami mencoba untuk melakukan verifikasi Radix-4 Pipelined 16-point Complex FFT Core yang tersedia sebagai model VHDL RTL dalam Library Xilinx.

## 1. Pendahuluan

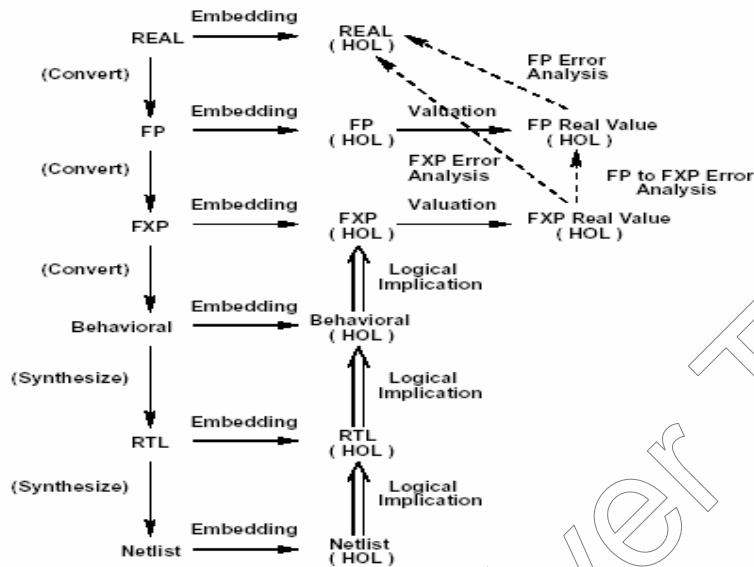
Discrete Fourier Transform (DFT) memainkan suatu peranan yang penting didalam analisis, design dan implementasi dari algoritma dan system pemrosesan sinyal waktu diskrit. Fast Fourier Transform (FFT) adalah metoda yang sangat efisien untuk menghitung koefisien dari Fourier diskrit ke suatu finite sekuen dari data yang kompleks. Karena substansi waktu yang tersimpan lebih dari pada metoda konvensional, fast fourier transform merupakan aplikasi temuan yang penting didalam sejumlah bidang yang berbeda seperti analisa spectrum, *speech and optical signal processing*, design filter digital. Algoritma FFT berdasarkan atas prinsip pokok dekomposisi perhitungan *discrete fourier transform* dari suatu sekuen sepanjang N kedalam transformasi diskrit Fourier secara berturut-turut lebih kecil. Cara prinsip ini diterapkan memimpin kearah suatu variasi dari algortima yang berbeda, dimana semuanya memperbandingkan peningkatan kecepatan perhitungan. Ada dua bentuk kanonik dan kelas dasar dari algortima FFT untuk sejumlah perkalian dan penambahan aritmatika sebagai suatu ukuran dari kompleksitas perhitungan yang proporsional ke N lebih dari  $N^2$  seperti didalam metoda konvensional. Cooley dan Tukey lebih dulu mengusulkan yang disebut decimation-in-time (DIT), nama tersebut diperoleh dari fakta bahwa

dalam proses mengatur perhitungan kedalam transformasi yang lebih kecil, urutan input  $x[n]$  (pemikiran secara umum sebagai urutan waktu) di dekomposisi berturut-turut kedalam sub urutan yang lebih kecil. Dalam kelas umum yang kedua dari algoritma yang diusulkan oleh Gentleman dan Sande, urutan dari koefisien discrete Fourier transform  $X[k]$  didekomposisi kedalam urutan yang lebih kecil, karena itu namanya, decimation-in-frequency (DIF). Didalam analisa teori dari discrete fourier transform, kita mengasumsikan secara umum bahwa nilai dan koefisien sinyal direpresentasikan dalam system bilangan real dan mengekspresikan ketepatan infinite. Ketika diterapkan dalam suatu hardware digital dengan tujuan khusus atau sebagai algoritma computer, kita harus merepresentasikan sinyal dan koefisien dalam beberapa system bilangan digital yang harus selalu menjadi ketepatan finite. Ada suatu masalah ketelitian yang tidak dapat dipisahkan dalam menghitung koefisien fourier, karena sinyal di representasikan dengan jumlah yang terbatas dari bit-bit dan operasi harus dilaksanakan dengan suatu batasan ketelitian oleh panjang kata yang terbatas ini. Ada suatu variasi dari tipe aritmatika yang digunakan dalam implementasi system digital. Diantara yang paling umum adalah floating- dan fixed-point. Disini, semua operand di representasikan dengan suatu format atau penandaan yang khusus suatu panjang kata yang tetap dan eksponen yang tetap, sedang struktur dan operasi control dari program ideal tanpa perubahan. Pada bagian implementasi, model fixed-point dari algoritma telah di transformasikan ke dalam gambaran target yang jauh lebih baik, juga menggunakan deskripsi hardware atau suatu bahasa pemrograman. Proses design ini dapat dibantu dengan sejumlah tool khusus CAD (Computer Aided Design) seperti SPW (Cadence), CoCentric (Synopys), dan Matlab-Simulink (Mathworks).

Didalam laporan ini kami menggambarkan verifikasi formal dari algoritma fast fourier transform menggunakan HOL theorem proving berdasarkan diagram komunikasi yang ditunjukkan dalam gambar 1. Dalam aliran design FFT, focus pertama pada transisi dari real ke tingkatan-tingkatan floating- dan fixed-point. Sesudah itu, model pertama dari spesifikasi real yang ideal dari algoritma FFT dan hubungan implementasi floating- dan fixed-point sebagai predikat dalam higher-

order logic. Untuk ini, dibuat penggunaan teori dalam HOL pada konstruksi dari bilangan real dan complex real, secara formal dari standard IEEE-754 berdasarkan aritmatika floating-point, dan formalitas dari aritmatika fixed-point. Kami menggunakan fungsi penilaian ke penemuan nilai real dari floating- dan fixed-point output FFT dan definisi error sebagai perbedaan diantara nilai yang ada dan hubungan output dari spesifikasi real ideal. Kami menetapkan lemmas pokok pada analisa error dari pembulatan floating- dan fixed-point dan operasi perhitungan aritmatika terhadap abstrak matematika. Akhirnya, kami menggunakan lemmas ini sebagai suatu model untuk memperoleh ekspresi untuk akumulasi dari *roundoff error* dalam algoritma floating- dan fixed-point dengan definisi yang berulang-ulang dan dalam kondisi initial, yang mempertimbangkan pengaruh dari kuantisasi input dan ketidakteelitian dalam koefisien, untuk setiap dua bentuk kanonikal dari realisasi: *decimation-in-time* (DIT) dan *decimation-in-frequency* (DIF). Selagi pekerjaan teoritis pada perhitungan error dalam kaitan dengan pengaruh ketepatan finite didalam realisasi algoritma FFT dengan aritmatika floating- dan fixed-point telah dipelajari secara ekstensif sejak enam tahun belakangan ini.

Setelah menangani transisi ke tingkatan-tingkatan floating- dan fixed-point, maka di terapkan ke representasi HDL. Pada point ini, kita menggunakan tehnik formal yang diketahui dengan baik ke model design FFT pada behavioral dan/atau level-level RT dan sintesis netlist gate level yang sesuai dengan HOL. Langkah selanjutnya membuktikan level-level yang berbeda dengan menggunakan hirarki pendekatan pembuktian dalam HOL [24]. Dalam pendekatan ini, hirarki pembuktian yang diimplementasi menyiratkan RTL (Register Transfer Level) yang ringkas. RTL ini terkait dengan pembuktian formal ke spesifikasi behavioral. Selanjutnya digunakan spesifikasi algoritma high level fixed-point yang sudah dikaitkan ke deskripsi floating-point dan spesifikasi real ideal melalui analisa error.



Gambar 1. Spesifikasi FFT dan Metodologi Verifikasi

## 2. Analisis Error Fast Fourier Transform

Dalam bagian ini, hasil utama untuk akumulasi roundoff dalam algoritma FFT menggunakan HOL theorem proving diperoleh dan diringkas. Pada kebanyakan bagian, diskusi berikut adalah phrase dalam bentuk *decimation-in-frequency* (DIT) ke algoritma radix-2. Bagaimanapun hasilnya bisa diterapkan hanya dengan modifikasi kecil ke bentuk *decimation-in-time* (DIT). Lagi pula, kebanyakan dari ide yang dikerjakan dalam analisis error dari algoritma radix-2 dapat di utilitaskan dalam analisis dari algoritma yang lainnya. Berikut ini pertama kali akan di gambarkan secara detail teori di samping analisis dan kemudian menjelaskan bagaimana analisis ini dibentuk dalam HOL.

Discrete fourier transform merupakan urutan  $\{x(n)\}_{n=0}^{N-1}$  dari yang didefinisikan sebagai

$$A(p) = \sum_{n=0}^{N-1} x(n)(W_N)^{np}, \quad p = 0, 1, 2, \dots, N-1 \quad (1)$$

dimana  $W_N = e^{-j2\pi/N}$  dan  $j = \sqrt{-1}$ . Untuk sederhananya, diskusi kita terbatas ke algoritma FFT radix-2 dimana jumlah dari point N ke transformasi Fourier yang memenuhi hubungan  $N = 2^m$ , m bernilai integer. Hasil dapat diperluas ke radix lain yang lebih dari 2. Dengan menggunakan metoda FFT, koefisien FFT  $A(p) = \sum_{n=0}^{N-1} x(n)(W_N)^{np}$ , dapat di hitung dalam langkah-langkah iterasi  $m = \log_2 N$ . Pada

setiap langkah, suatu array bilangan kompleks N yang dihasilkan dengan menggunakan hanya bilangan dalam array sebelumnya. Untuk menjelaskan algoritma FFT, ambil setiap integer p,  $p = 0, 1, 2, \dots, N-1$ , diperluas kedalam bentuk binary sebagai

$$p = 2^{m-1}p_0 + 2^{m-2}p_1 + \dots + 2p_{m-2} + p_{m-1}, \quad p_k = 0 \text{ atau } 1 \quad (2)$$

dan ambil  $p^*$  menandakan jumlah yang bersesuaian ke urutan bit yang berkebalikan dari p,

$$p^* = 2^{m-1}p_{m-1} + 2^{m-2}p_{m-2} + \dots + 2p_1 + p_0 \quad (3)$$

**Algoritma Decimation-in-frequency (DIF) FFT:**

Ambil  $\{A_k(p)\}_{p=0}^{N-1}$  menandakan perhitungan bilangan kompleks pada langkah ke-k.

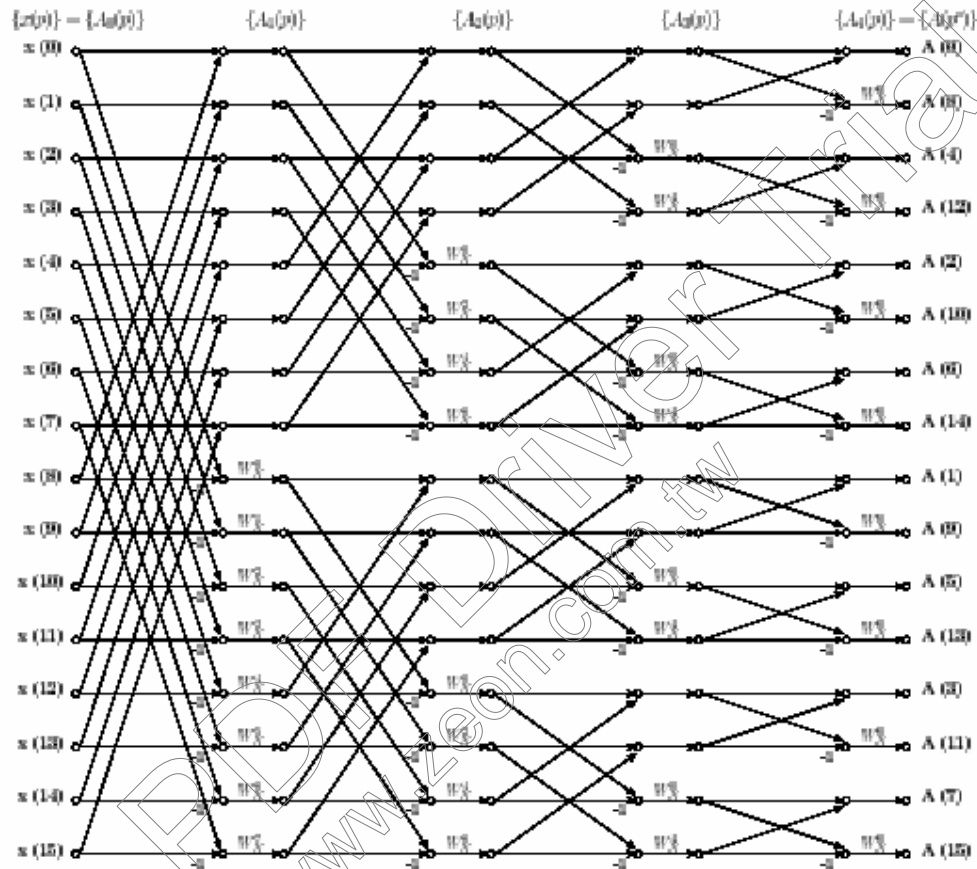
Maka algoritma FFT decimation-in-frequency (DIF) dapat di ekspresikan sebagai

$$A_{k+1}(p) = \begin{cases} A_k(p) + A_k(p + 2^{m-1-k}) & \text{Jika } p_k = 0 \\ [A_k(p - 2^{m-1-k}) - A_k(p)] w_k(p) & \text{Jika } p_k = 1 \end{cases} \quad (4)$$

dimana  $w_k(p)$  adalah perpankangan  $W_N$  yang diberikan dengan  $w_k(p) = (W_N)^{z_k(p)}$ , dimana

$$z_k(p) = 2^k (2^{m-1-k}p_k + 2^{m-1-k}p_{k+1} + \dots + 2p_{m-2} + p_{m-1}) - 2^{m-1}p_k$$

Persamaan (4) dilaksanakan untuk  $k = 0, 1, 2, \dots, m-1$ , dengan  $A_0(p) = x(p)$ . Hal itu menunjukkan bahwa pada langkah terakhir  $\{\dots\}$  koefisien discrete fourier dalam pesanan diatur kembali. Secara khusus,  $A_m(p) = A(p^*)$  dengan p dan  $p^*$  di perluas dan didefinisikan seperti dalam persamaan (2) dan (3) secara berturut-turut. Gambar 2 menunjukkan flowgraph sinyal dari perhitungan nyata untuk kasus  $N = 2^4$ .



Gambar 2. Flowgraph sinyal dari decimation-in-frequency FFT,  $N = 2^4$

### Algoritma Decimation-in-time (DIT) FFT

Ambil  $\{A_k(p)\}_{p=0}^{N-1}$  menandakan perhitungan bilangan kompleks pada langkah ke-k.

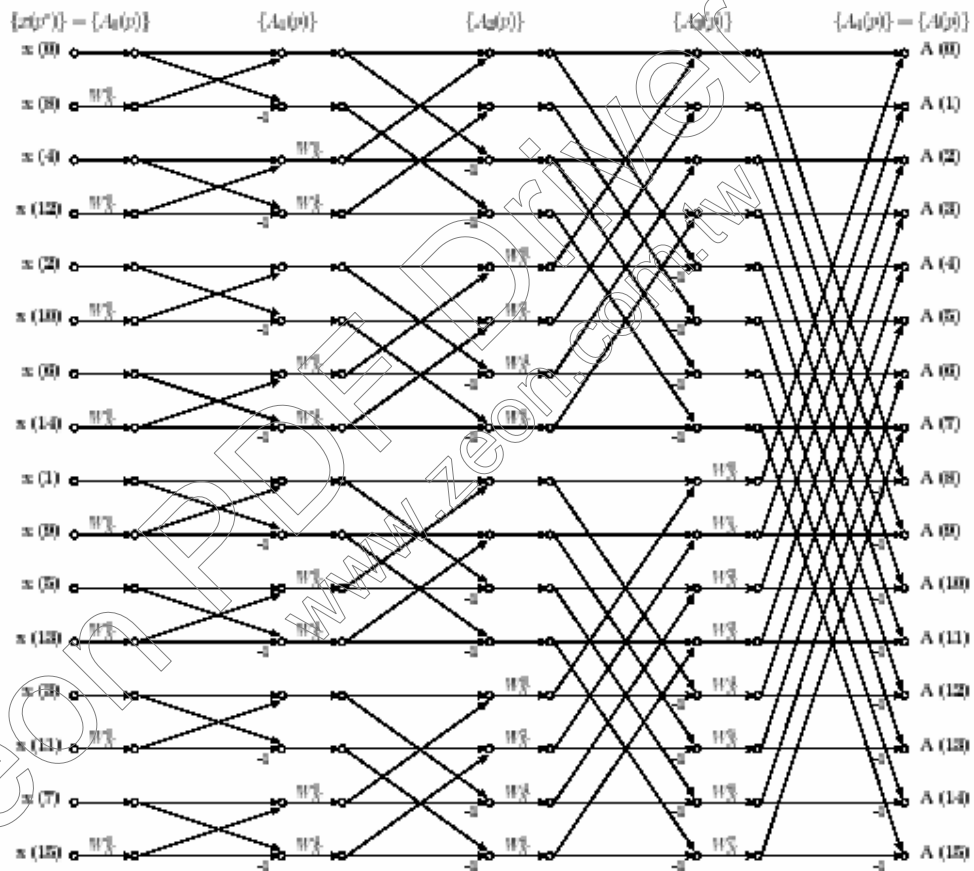
Maka algoritma decimation-in-time (DIT) FFT dapat di ekspresikan sebagai

$$A_{k+1}(p) = \begin{cases} A_k(p) + W_k(p) A_k(p + 2^k) & \text{Jika } p_{m-1-k} = 0 \\ A_k(p - 2^k) - W_k(p) A_k(p) & \text{Jika } p_{m-1-k} = 1 \end{cases} \quad (5)$$

dimana  $w_k(p)$  adalah perpangkatan  $W_N$  yang diberikan dengan  $w_k(p) = (W_N)^{z_k(p)}$ ,  
dimana

$$Z_k(p) = 2^{m-1-k} (2^k p_{m-1-k} + 2^{k-1} p_{m-k} + \dots + 2p_{m-2} + p_{m-1}) - 2^{m-1} p_{m-1-k}$$

Persamaan (5) dilaksanakan untuk  $k = 0, 1, 2, \dots, m - 1$ , dengan  $A_0(p) = x(p^*)$  dengan  $p$  dan  $p^*$  diperluas dan didefinisikan seperti pada persamaan (2) dan (3) secara berturut-turut. Hal itu menunjukkan bahwa pada langkah terakhir  $\{A_m(p)\}_{p=0}^{N-1}$  koefisien discrete fourier dalam pesan normal. Secara khusus,  $A_m(p) = A(p)$ . Gambar 3 menunjukkan flowgraph sinyal dari perhitungan nyata untuk kasus  $N = 2^4$ .



Gambar 2. Flowgraph sinyal dari decimation-in-time FFT,  $N = 2^4$

Ada tiga sumber umum dari error berhubungan dengan algoritma FFT, dinamakan:

1. *input quantization* : disebabkan oleh kuantisasi dari sinyal input  $\{x_n\}$  ledakan suatu set dari level-level discrete.
2. *coefficient accuracy* : disebabkan oleh representasi dari koefisien  $\{w_k(p)\}$  dengan suatu panjang word tidak terbatas.
3. *round-off accumulation* : disebabkan oleh akumulasi dari roundoff error pada operasi aritmatika.

Oleh karena itu, array sebenarnya dihitung dengan menggunakan persamaan (4) dan (5) yang secara umum berbeda dari  $\{A_k(p)\}_{p=0}^{N-1}$ . Kami menandakan floating- dan fixed-point yang sebenarnya menghitung array dengan  $\{A'_k(p)\}_{p=0}^{N-1}$  dan  $\{A''_k(p)\}_{p=0}^{N-1}$  secara berturut-turut. Maka, didefinisikan error yang bersesuaian dari elemet ke-p pada langkah k sebagai

$$e_k(p) = A'_k(p) - A_k(p) \quad (6)$$

$$e'_k(p) = A''_k(p) - A_k(p) \quad (7)$$

$$e''_k(p) = A''_k(p) - A'_k(p) \quad (8)$$

dimana  $e_k(p)$  dan  $e'_k(p)$  didefinisikan sebagai error diantara implementasi floating- dan fixed-point yang sebenarnya dan spesifikasi real ideal secara berturut-turut.  $e''_k(p)$  adalah error transisi dari level-level floating- dan fixed-point. Maka

$$e(p) = e_m(p^*) \quad (9)$$

$$e'(p) = e'_m(p^*) \quad (10)$$

$$e''(p) = e''_m(p^*) \quad (11)$$

error yang bersesuaian yang dilakukan dalam perhitungan dari koefisien fourier  $\{A(p)\}_{p=0}^{N-1}$  dengan menggunakan metoda FFT.

Itu telah jelas dari diskusi diatas bahwa algoritma fast fourier transform dari persamaan (4), kami mempunyai

$$A'_{k+1}(p) = \begin{cases} fl(A'_k(p) + A'_k(p + 2^{m-1-k})) & \text{jika } p_k = 0 \\ fl([A'_k(p - 2^{m-1-k}) - A'_k(p)] w_k(p)) & \text{jika } p_k = 1 \end{cases} \quad (12)$$

dan

$$A_{k+1}''(p) = \begin{cases} fxp(A_k''(p) + A_k''(p + 2^{m-l-k})) & \text{jika } p_k = 0 \\ fxp([A_k''(p - 2^{m-l-k}) - A_k''(p)] w_k(p)) & \text{jika } p_k = 1 \end{cases} \quad (13)$$

dimana notasi  $fl(.)$  digunakan untuk menandakan bahwa operasi dibentuk menggunakan aritmatika floating- dan fixed-point secara berturut-turut.

Dalam menganalisis pengaruh floating-point roundoff, pengaruh dari pembulatan akan direpresentasikan secara multiplikasi. Ambil \* menandakan operasi aritmatika +, -, X, /, jika p merepresentasikan ketepatan dari format floating- dan fixed-point, maka

$$fl(x * y) = (x * y) (1 + \delta), \text{ dimana } |\delta| \leq 2^{-p} \quad (14)$$

Notasi  $fl(.)$  digunakan untuk menandai bahwa operasi dibentuk menggunakan aritmatika floating-point. Teorema di kaitkan dengan operasi aritmatika floating-point seperti penambahan, pengurangan, perkalian, dan pembagian ke matematika abstrak berhubungan dengan error yang berseuaian.

Ketika error pembulatan untuk aritmatika floating-point masuk kedalam system yang bermultiplikasi, itu adalah suatu komponen aditif untuk aritmatika floating-point. Di dalam kasus teorema analisis error yang pokok untuk operasi aritmatika fixed-point yang berlawanan dengan counterpart matematika bastrak mereka dapat state sebagai

$$fxp(x * y) = (x * y) + \varepsilon, \text{ dimana } |\varepsilon| \leq 2^{-\text{fracbits}(x)} \quad (15)$$

dan fracbits adalah jumlah dari bit yang ke kanan dari poin binary dalam format fixed-point X. Notasi  $fxp(.)$  digunakan untuk menandakan bahwa operasi di bentuk menggunakan aritmatika fixed-point. Kami membuktikan persamaan (14) dan (15) sebagai teorema didalam *higer-order-logic* dengan HOL.

Dalam persamaan (4)  $\{A_k(p)\}$  adalah bilangan kompleks, sehingga bagian real dan imajiner dihitung secara terpisah. Ambil

$$\begin{aligned}
B_k(p) &= \text{Re} [A_k(p)] & C_k(p) &= \text{Im} [A_k(p)] \\
U_k(p) &= \text{Re} [W_k(p)] & V_k(p) &= \text{Im} [W_k(p)]
\end{aligned} \tag{16}$$

dimana notasi  $\text{Re}[\cdot]$  dan  $\text{Im}[\cdot]$  menandakan secara berturut-turut bagian real dan imajiner dari kuantitas bagian dalam kurung  $[\cdot]$ . Persamaan (4) dapat di tulis kembali sebagai

$$\begin{aligned}
B_{k+1}(p) &= B_k(p) + B_k(q) \\
C_{k+1}(p) &= C_k(p) + C_k(q)
\end{aligned} \quad \text{Jika } p_k = 0 \tag{17}$$

$$\begin{aligned}
B_{k+1}(p) &= [B_k(r) - B_k(p) U_k(p)] - [C_k(r) + C_k(p)]V_k(p) \\
C_{k+1}(p) &= [C_k(r) - C_k(p) U_k(p)] - [B_k(r) + B_k(p)]V_k(p)
\end{aligned} \quad \text{Jika } p_k = 1 \tag{18}$$

dimana  $q = p + 2^{m-1-k}$  dan  $r = p - 2^{m-1-k}$ . Pada penggunaan tanda petik satu dan tanda petik dua menandakan hasil perhitungan floating- dan fixed-point yang sebenarnya, bagian real dan imajiner dari  $A'_{k+1}(p)$  dan  $A''_{k+1}(p)$  menurut persamaan (12) dan (13) yang diberikan dengan

$$\begin{aligned}
B'_{k+1}(p) &= B'_k(p) + B'_k(q) \\
C'_{k+1}(p) &= C'_k(p) + C'_k(q)
\end{aligned} \quad \text{Jika } p_k = 0 \tag{19}$$

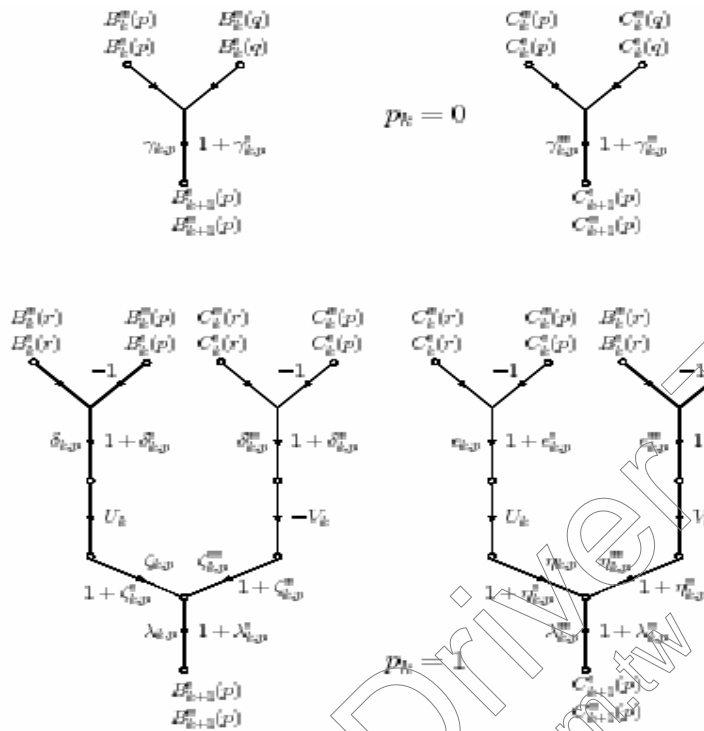
$$\begin{aligned}
B'_{k+1}(p) &= [B'_k(r) - B'_k(p) U_k(p)] - [C'_k(r) + C'_k(p)]V_k(p) \\
C'_{k+1}(p) &= [C'_k(r) - C'_k(p) U_k(p)] - [B'_k(r) + B'_k(p)]V_k(p)
\end{aligned} \quad \text{Jika } p_k = 1 \tag{20}$$

dan

$$\begin{aligned}
B''_{k+1}(p) &= B''_k(p) + B''_k(q) \\
C''_{k+1}(p) &= C''_k(p) + C''_k(q)
\end{aligned} \quad \text{Jika } p_k = 0 \tag{21}$$

$$\begin{aligned}
B''_{k+1}(p) &= [B''_k(r) - B''_k(p) U_k(p)] - [C''_k(r) + C''_k(p)]V_k(p) \\
C''_{k+1}(p) &= [C''_k(r) - C''_k(p) U_k(p)] - [B''_k(r) + B''_k(p)]V_k(p)
\end{aligned} \quad \text{Jika } p_k = 1 \tag{22}$$

Flowgraph error yang sesuai menunjukkan pengaruh dari roundoff error menggunakan teorema floating- dan fixed-point yang pokok menurut persamaan (14) dan (15) secara berturut-turut, seperti yang ditunjukkan dalam Gambar 4, yang juga mengindikasikan urutan dari perhitungan.



Gambar 4. Error Flowgraph untuk decimation-in-frequency FFT

Jumlah  $\gamma'_{k,p}$ ,  $\gamma''_{k,p}$ ,  $\sigma'_{k,p}$ ,  $\sigma''_{k,p}$ ,  $\epsilon'_{k,p}$ ,  $\epsilon''_{k,p}$ ,  $\zeta'_{k,p}$ ,  $\zeta''_{k,p}$ ,  $\eta'_{k,p}$ ,  $\eta''_{k,p}$ ,  $\lambda'_{k,p}$ , dan  $\lambda''_{k,p}$  adalah error yang disebabkan oleh *roundoff* floating-point pada setiap langkah aritmatika. Jumlah error sesuai untuk *roundoff* fixed-point adalah  $\gamma_{k,p}$ ,  $\gamma''_{k,p}$ ,  $\sigma_{k,p}$ ,  $\sigma''_{k,p}$ ,  $\epsilon_{k,p}$ ,  $\epsilon''_{k,p}$ ,  $\zeta_{k,p}$ ,  $\zeta''_{k,p}$ ,  $\eta_{k,p}$ ,  $\eta''_{k,p}$ ,  $\lambda_{k,p}$ , dan  $\lambda''_{k,p}$ . Oleh karena itu, bagian real dan imajiner dari output floating- dan fixed-point  $A''_{k+1}(p)$  dan  $A'_{k+1}(p)$  terlihat dengan eksplisit diberikan oleh

$$\left. \begin{aligned} B'_{k+1}(p) &= [B'_k(p) + B'_k(q)](1 + \gamma'_{k,p}) \\ C'_{k+1}(p) &= [C'_k(p) + C'_k(q)](1 + \gamma'_{k,p}) \end{aligned} \right\} \text{ if } p_k = 0 \tag{23}$$

$$\left. \begin{aligned} B'_{k+1}(p) &= [B'_k(r) - B'_k(p)] U_k(p)(1 + \delta'_{k,p})(1 + \zeta'_{k,p})(1 + \lambda'_{k,p}) \\ &\quad - [C'_k(r) - C'_k(p)] V_k(p)(1 + \delta'_{k,p})(1 + \zeta'_{k,p})(1 + \lambda'_{k,p}) \\ C'_{k+1}(p) &= [C'_k(r) - C'_k(p)] U_k(p)(1 + \epsilon'_{k,p})(1 + \eta'_{k,p})(1 + \lambda'_{k,p}) \\ &\quad + [B'_k(r) - B'_k(p)] V_k(p)(1 + \epsilon'_{k,p})(1 + \eta'_{k,p})(1 + \lambda'_{k,p}) \end{aligned} \right\} \text{ if } p_k = 1$$

dan

$$\begin{aligned}
& \left. \begin{aligned} B_{k+1}^w(p) &= [B_k^w(p) + B_k^w(q)] + \gamma_{k,p}^w \\ C_{k+1}^w(p) &= [C_k^w(p) + C_k^w(q)] + \gamma_{k,p}^w \end{aligned} \right\} \text{ if } p_k = 0 \\
& \left. \begin{aligned} B_{k+1}^w(p) &= [B_k^w(r) - B_k^w(p) + \delta_{k,p}^w] U_k(p) + \zeta_{k,p}^w \\ & \quad ([C_k^w(r) - C_k^w(p) + \delta_{k,p}^w] V_k(p) + \zeta_{k,p}^w) + \lambda_{k,p}^w \\ C_{k+1}^w(p) &= [C_k^w(r) - C_k^w(p) + \epsilon_{k,p}^w] U_k(p) + \eta_{k,p}^w + \\ & \quad ([B_k^w(r) - B_k^w(p) + \epsilon_{k,p}^w] V_k(p) + \eta_{k,p}^w) + \lambda_{k,p}^w \end{aligned} \right\} \text{ if } p_k = 1
\end{aligned} \tag{24}$$

Error  $e_k(p)$ ,  $e'_k(p)$ , dan  $e''_k(p)$  didefinisikan didalam persamaan (6), (7), dan (8) adalh komplek dan dapat ditulis kembali sebagai

$$e_k(p) = B_k^{\cdot}(p) - B_k(p) + j[C_k^{\cdot}(p) - C_k(p)] \tag{25}$$

$$e'_k(p) = B_k^{\prime\prime}(p) - B_k(p) + j[C_k^{\prime\prime}(p) - C_k(p)] \tag{26}$$

$$e''_k(p) = B_k^{\prime\prime}(p) - B_k(p) + j[C_k^{\prime\prime}(p) - C_k^{\prime\prime}(p)] \tag{27}$$

$$k = 1, 2, \dots, m, p = 0, 1, \dots, N-1$$

dengan

$$e_0(p) = e'_0(p) = e''_0(p) = 0, \quad p = 0, 1, \dots, N-1 \tag{28}$$

Dari persamaan (18), (24), (25), (26), dan (27), kita mempunyai

### 1. FFT Real ke Floating-Point :

$$e_{k+1}(p) = \begin{cases} e_k(p) + e_k(q) + f_k(p) & \text{jika } p_k = 0 \\ [e_k(r) - e_k(p)] W_k(p) + f_k(p) & \text{jika } p_k = 1 \end{cases} \tag{29}$$

dimana  $f_k(p)$  diberikan oleh

$$f_k(p) = \begin{cases} \gamma_{k,p}^w [B_k^w(p) + B_k^w(q)] + j \gamma_{k,p}^w [C_k^w(p) + C_k^w(q)] & \text{if } p_k = 0 \\ [(1 + \delta_{k,p}^w)(1 + \zeta_{k,p}^w)(1 + \lambda_{k,p}^w) - 1] [B_k^w(r) - B_k^w(p)] U_k(p) \\ - [(1 + \delta_{k,p}^w)(1 + \zeta_{k,p}^w)(1 + \lambda_{k,p}^w) - 1] [C_k^w(r) - C_k^w(p)] V_k(p) \\ + j [(1 + \epsilon_{k,p}^w)(1 + \eta_{k,p}^w)(1 + \lambda_{k,p}^w) - 1] [C_k^w(r) - C_k^w(p)] U_k(p) \\ + j [(1 + \epsilon_{k,p}^w)(1 + \eta_{k,p}^w)(1 + \lambda_{k,p}^w) - 1] [B_k^w(r) - B_k^w(p)] V_k(p) & \text{if } p_k = 1 \end{cases} \tag{30}$$

### 2. FFT Real ke Fixed\_Point:

$$e_{k+1}^{\cdot}(p) = \begin{cases} e_k^{\cdot}(p) + e_k^{\cdot}(q) + f_k^{\cdot}(p) & \text{jika } p_k = 0 \\ [e_k^{\cdot}(r) - e_k^{\cdot}(p)] W_k(p) + f_k^{\cdot}(p) & \text{jika } p_k = 1 \end{cases} \tag{31}$$

dimana  $f'_k(p)$  diberikan oleh

$$f'_k(p) = \begin{cases} \gamma_{k,p} + j\gamma_{k,p}^{**} & \text{if } p_k = 0 \\ \delta_{k,p}U_k(p) + \zeta_{k,p} - \delta_{k,p}^{**}V_k(p) - \zeta_{k,p}^{**} + \lambda_{k,p} + \\ j(\epsilon_{k,p}U_k(p) + \eta_{k,p} + \epsilon_{k,p}^{**}V_k(p) + \eta_{k,p}^{**} + \lambda_{k,p}^{**}) & \text{if } p_k = 1 \end{cases} \quad (32)$$

### 3. FFT Floating- ke Fixed-Point:

$$e_{k+1}^{**}(p) = \begin{cases} e_k^{**}(p) + e_k^{**}(q) + f'_k(p) - f_k(p) & \text{if } p_k = 0 \\ [e_k^{**}(r) - e_k^{**}(p)] w_k(p) + f'_k(p) - f_k(p) & \text{if } p_k = 1 \end{cases} \quad (33)$$

dimana  $f_k(p)$  dan  $f'_k(p)$  diberikan oleh persamaan (30) dan (32).

Akumulasi dari error roundoff ditentukan melalui persamaan rekursif (29), (30), (31), (32), (33) dengan kondisi intial yang diberikan dengan persamaan (28). Dalam HOL, kami pertama mengkonstruksi bilangan kompleks pada real sama dengan [9]. Kami mendefinisikan dalam HOL suatu tipe baru untuk bilangan kompleks, ke dalam bijeksi dengan  $\mathbb{C}$ . Bijeksi ditulis didalam HOL sebagai  $\text{complex} : \mathbb{R}^2 \rightarrow \mathbb{C}$  dan  $\text{cords} : \mathbb{C} \rightarrow \mathbb{R}^2$ . Kami menggunakan singkatan menyenangkan untuk bagian real (Re) dan imajiner (Im) dari bilangan kompleks, dan juga definisi operasi aritmatika seperti penambahan, pengurangan, perkalian dan pembagian pada bilangan kompleks. Kami menggunakan symbol yang sama (+, -, x) untuk C dan R. Dengan cara yang sama, kami mengkonstruksikan bilangan kompleks pada bilangan floating- dan fixed-point. Tipe untuk bilangan kompleks floating- dan fixed-point didefinisikan dalam bijeksi dengan float x float dan fpx x fpx, dengan bijeksi ditulis dalam HOL sebagai  $\text{float\_complex} : \text{float}^2 \rightarrow \mathbb{C}$ ,  $\text{float\_coords} : \mathbb{C} \rightarrow \text{float}^2$ , dan  $\text{fxp\_complex} : \text{fxp}^2 \rightarrow \mathbb{C}$  dan  $\text{fxp\_coords} : \mathbb{C} \rightarrow \text{fxp}^2$  secara berturut-turut. Singkatan untuk bagian real dan imajiner dari bilangan kompleks floating- dan fixed-point disebut dengan (*float\_Re*), (*float\_Im*), dan (*fxp\_Re*), (*fxp\_Im*) secara berturut-turut. Akhirnya, kami mendefinisikan operasi aritmatika pada bilangan kompleks floating- dan fixed-point. Untuk operas aritmatika kompleks floating-point digunakan symbol yang umum, dan kasus fixed-point digunakan symbol *fxo\_complex\_add*, *fxp\_complex\_sub*, dan *fxp\_complex\_mul*. Maka kami definisikan prinsip akar-N pada kesatuan ( $e^{j2\pi/N}$

$= \cos(2\pi n/N) - j \sin(2\pi n/N)$ , dan pangkatnya ( $OMEGA$ ) seperti suatu bilangan kompleks menggunakan fungsi sinus dan cosinus yang tersedia dalam teori dari pustaka real HOL [6]. Kami men-spesifikasi-kan ekspresi dalam HOL untuk ekspansi bilangan natural kedalam bentuk biner dalam urutan normal dan yang diatur kembali menurut persamaan (2) dan (3). Yang tersebut diatas memungkinkan kita menspesifikasikan algoritma FFT kedalam level-level abstraksi real ( $REAL\_FFT$ ), floating ( $FLOAT\_FFT$ ), dan fixed-point ( $FXP\_FFT$ ) menggunakan definisi yang rekursif dalam HOL seperti di gambarkan dalam persamaan (4), (12), dan (13). Maka kami mendefinisikan bagian real dan imajiner dari algoritma FFT ( $FFT\_REAL$ ,  $FFT\_IMAGE$ ) dan pangkat dari prinsip akar\_N pada kesatuan ( $OMEGA\_REAL$ ,  $OMEGA\_IMAGE$ ) menurut persamaan (16). Berikutnya, kami membuktikan dalam lemmas yang terpisah bahwa bagian real dan imajiner dari algoritma dalam level-level real, floating- dan fixed-point dapat dikembangkan seperti persamaan (18), (20), dan (22). Maka kami membuktikan lemmas untuk menjelaskan error dalam setiap langkah-langkah aritmatika dalam bagian real dan imajiner dari algoritma floating- dan fixed-point menurut persamaan (24), dan (25). Kami membuktikan lemmas ini menggunakan lemmas analisis error pokok untuk operasi aritmatika dasar [9] menurut persamaan (14) dan (15), Maka kami mendefinisikan dalam HOL error dari elemen ke- $p$  dari algoritma floating- ( $REAL\_TO\_FLOAT\_FFT\_ERROR$ ) dan fixed-point ( $REAL\_TO\_FXP\_FFT\_ERROR$ ) FFT pada langkah  $k$ , dan error yang sesuai dalam transisi dari floating- dan fixed-point ( $FLOAT\_TO\_FXP\_FFT\_ERROR$ ), menurut persamaan (6), (7), dan (8). Setelah itu, kami membuktikan lemmas untuk menulis kembali error sebagai bilangan kompleks menggunakan bagian real dan imajiner menurut persamaan (25), (26), dan (27). Akhirnya, kami membuktikan untuk menetapkan akumulasi dari error roundoff dalam algoritma floating- dan fixed-point FFT dengan persamaan rekursif dan kondisi intial menurut persamaan (28), (29), (30), (31), (32), dan (33). Kode HOL yang sesuai di ditampilkan dalam Appendix A.



Sangat mudah membuktikan bahwa sebagai  $n_0$  dan  $n_1$  didapat pada semua nilai yang mungkin dalam indeks range,  $n$  diperoleh melalui semua nilai yang mungkin dari 0 sampai 15 dengan tidak ada nilai yang diulangi. Ini juga benar untuk indeks frekuensi dari  $p$ . Dengan menggunakan pemetaan indeks, kita dapat mengekspresikan algoritma radix-4 FFT secara rekursif sebagai

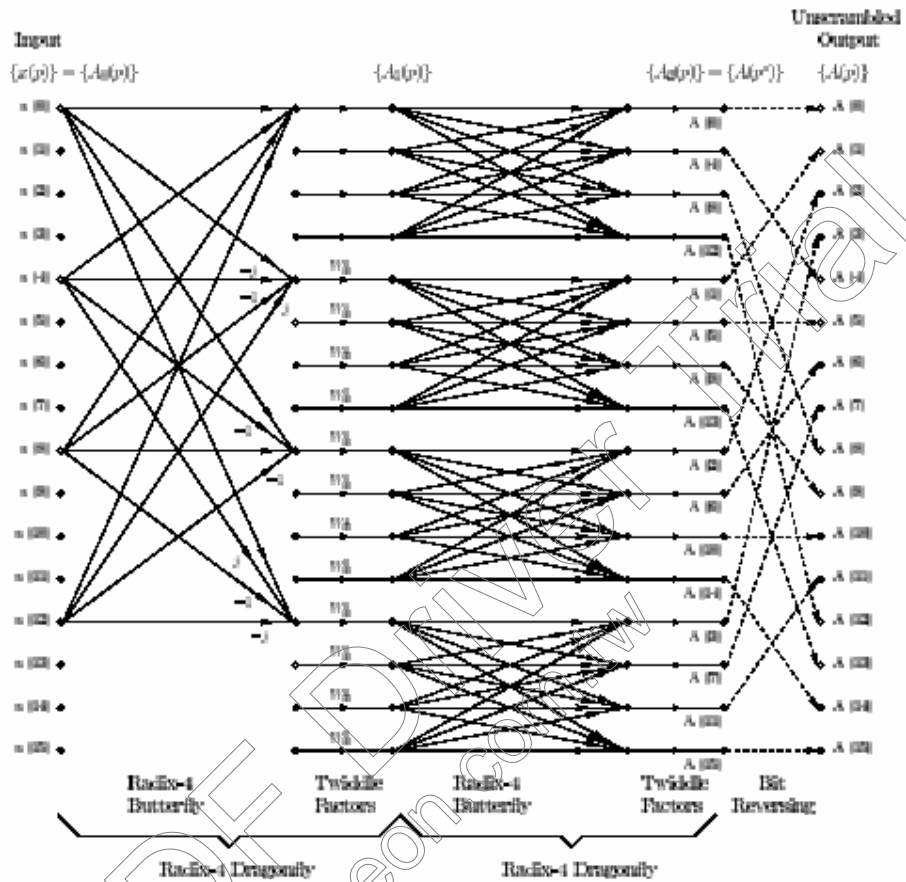
$$A_1(p_0, n_0) = \sum_{n_1=0}^3 x(n_1, n_0) (W_{16})^{Ap_0n_1} \quad (36)$$

$$A_2(p_0, n_1) = \sum_{n_0=0}^3 A_1(p_1, n_0) (W_{16})^{(Ap_1+p_0)n_0} \quad (37)$$

Hasil akhir dapat ditulis sebagai

$$A(p_1, p_0) = A_2(p_0, p_1) \quad (38)$$

Sehingga, seperti dalam algoritma radix-2, hasil dalam urutan terbalik. Berdasarkan persamaan (36), (37), dan (38) kami dapat mengembangkan suatu flowgraph sinyal untuk algoritma radix-4 16-point FFT seperti ditunjukkan Gambar 6. Graph disusun dari dua stage radix-4 dragonfly yang suksesif. Setiap radix-4 dragonfly adalah suatu kombinasi suksesif dari radix-4 butterfly dengan factor perkalian empat twiddle. Untuk mengurangi kebingungan dalam graph ini kami telah menunjukkan hanya satu radix-4 butterflies dalam stage pertama. Juga, kami tidak menunjukkan perkalian untuk radix-4 butterflies dalam stage kedua karena mereka sama mewakili butterfly dari stage pertama. Gambar 6 juga mengilustrasikan prosedur penguraian untuk algoritma radix-4.



Gambar 6. Signal flowgraph dari radix-4 16-point FFT

Dalam HOL, kami pertama-tama memodelkan deskripsi RTL dari radix-4 butterfly sebagai predikat didalam higher-order-logic. Blok mendapatkan suatu vector dari empat input data kompleks dan membentuk operasi seperti yang dilukiskan dalam flowgraph dari Gambar 6, untuk menghasilkan suatu vektor dari empat sinyal output kompleks. Bagian real dan imajiner dari sinyal input dan output direpresentasikan sebagai 16\_bit Boolean word. Kami mendefinisikan fungsi terpisah dalam HOL untuk operasi aritmatika seperti penambahan, pengurangan, dan perkalian pada 2's complement kompleks 16-bit Boolean word. Maka, kami membangun struktur butterfly yang lengkap menggunakan kombinasi yang sesuai dari operasi primitive. Setelah itu, kami menggambarkan blok radix-4 dragonfly (*DRAGONFLY\_RTL*) sebagai suatu konjungsi dari radix-4 butterfly dan

empat factor 16-bit twiddle perkalian kompleks seperti yang ditunjukkan dalam Gambar 6. Akhirnya, kami memodelkan deskripsi RTL yang lengkap dari struktur radix-4 16-point FFT (*DIF\_FFT\_RTL*) dalam HOL. Blok FFT didefinisikan sebagai konjungsi dari 8 instantiations dari blok radix-4 dragonfly menurut gambar 6, dengan mengaplikasikan kejadian waktu yang wajar dari sinyal input dan output ke tiap blok. Berikut dengan langkah yang sama, kami menggambarkan suatu struktur fixed-point radix-4 FFT (*DIF\_FFT\_FXP*) dalam HOL menggunakan type data fixed-point kompleks dan operasi aritmatika.

Untuk verifikasi formal, kami pertama-tama membuktikan bahwa deskripsi FFT RTL menyiratkan model fixed-point yang sesuai.

$$\begin{aligned} \vdash_{thm} \quad & \forall N X_r X_i W_r W_i A_r A_i. \\ & DIF\_FFT\_RTL\ N\ X_r\ X_i\ W_r\ W_i\ A_r\ A_i \Rightarrow \\ & DIF\_FFT\_FXP\ N\ FXP\_VECT\_COMPLEX(X_r, X_i) \\ & \quad FXP\_VECT\_COMPLEX(W_r, W_i) \\ & \quad FXP\_VECT\_COMPLEX(A_r, A_i) \end{aligned} \quad (39)$$

Pembuktian dari blok FFT kemudian dipecah-pecahkan kedalam pembuktian yang sesuai dari blok dragonfly.

$$\begin{aligned} \vdash_{thm} \quad & \forall N A_r A_i W_r W_i Q_r Q_i. \\ & DRAGONFLY\_RTL\ N\ A_r\ A_i\ W_r\ W_i\ Q_r\ Q_i \Rightarrow \\ & DRAGONFLY\_FXP\ N\ FXP(A_r)\ FXP(A_i) \\ & \quad FXP(W_r)\ FXP(W_i)\ FXP(Q_r)\ FXP(Q_i) \end{aligned} \quad (40)$$

Kami menggunakan fungsi abstraksi data FXP dan *FXP\_VECT\_COMPLEX* untuk mengkonversikan vector kompleks dari 16-bit 2 complement Boolean words kedalam vector fixed-point yang sesuai. Kami juga telah menggambarkan algoritma fixed-point radix-4 16-point (*FXP\_FFT\_ALGORITHM*) menggunakan definisi persamaan (36), (37) dan (38). Maka kami membuktikan bahwa perluasan deksripsi fixed-point berdasarkan flowgraph dari Gambar 6 menyiratkan format tertutup yang sesuai untuk membuat deskripsi algoritma fixed-point.

$$\begin{aligned} \vdash_{thm} \quad & \forall N X W A. \\ & DIF\_FFT\_FXP\ N\ X\ W\ A \Rightarrow \\ & \forall p. A(p) = FXP\_FFT\_ALGORITHM\ N\ X\ W\ p \end{aligned} \quad (39)$$

Dengan cara ini kami menyelesaikan verifikasi dari implementasi RTL untuk spesifikasi algoritma fixed-point dari algoritma FFT radix-4 16-point. Code HOL yang sesuai di tampilkan dalam Appendix A.

## 5. Kesimpulan

Dalam penulisan ini, kami sudah menjelaskan bagaimana algoritma FFT di terapkan pada radix-4 pipelined FFT Core Complex yang tersedia sebagai VHDL RTL pada Library Xilinx. Dari verifikasi maka dibuktikan bahwa implementasi FFT RTL menyiratkan spesifikasi yang sesuai pada tingkatan fixed-point hirarki verifikasi yang klasik dalam HOL, karenanya menghilangkan perbedaan diantara implementasi hardware dan spesifikasi matematika tingkat tinggi.

## Daftar Pustaka

1. B. Akbarpour, S. Tahar, dan A. Dekdouk, "Formalization of Cadence SPW Fixed-Point Arithmetic in HOL," In *Integrated Formal Methods*, LNCS 2335, Springer-Verlag, 2002.
2. B. Akbarpour, S. Tahar, "Modelling SystemC Fixed-Point Arithmetic in HOL," In *Formal Methods and Software Engineering*, LNCS 2885, Springer-Verlag, 2003.
3. R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, dan J. Vantassel, "Experience with Embedding Hardware Description Languages in HOL," In *Theorem Provers in Circuit Design*, North-Holland, 1992.
4. E. O. Brigham, "The Fast Fourier Transform," Prentice Hall, 1974.
5. W.T. Cochran et. al., "What is the Fast Fourier Transform," *IEEE Transaction on Audio and Electroacoustics*, AU-15, 1997.
6. M.J.C. Gordon, T.F. Melham, "Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic," Cambridge University Press, 1993.

7. J. R. Harrison, "Constructing the Real Number in HOL," "Formal Methods in System Design, Kluwer, 1994.
8. J. R. Harrison, "A Machine-Checked Theory of Floating-Point Arithmetic," In Theorem Proving in Higher Order Logic, LNCS 1690, Springer-Verlag, 1999.
9. B. Akbarpour, S. Tahar, "Verification of The Fast Fourier Transform using HOL Theorem Proving", Technical Report, Concordia University, Montreal-Canada, 2004.

ZEON PDF Driver  
www.zeon.com.tw

# Appendix A

## Tampilan dari Definisi dan Lemmas dalam HOL

Dalam bagian ini kami menyediakan tampilan lengkap dari definisi dan lemmas dalam HOL untuk spesifikasi dan verifikasi algoritma FFT pada level-level yang berbeda dari abstraksi.

1. Spesifikasi dari suatu N radix-2 decimation-in-frequency FFT menggunakan bilangan real dan kompleks:
  - Konstruksi Bilangan kompleks
  - Hubungan yang timbal balik dari tipe bijeksi:

```
complex_tybij =  
   $\vdash_{def} (\forall a. \text{complex } (\text{coords } a) = a) \wedge$   
     $\forall r. \text{K T } r = (\text{coords } (\text{complex } r) = r)$ 
```

- Untuk singkatan bagian real dan imajiner:

```
Re =  $\vdash_{def} \forall z. \text{Re } z = \text{FST } (\text{coords } z)$   
Im =  $\vdash_{def} \forall z. \text{Im } z = \text{SND } (\text{coords } z)$ 
```

- Unit imajiner:

```
ii =  $\vdash_{def} ii = \text{complex } (0,1)$ 
```

- Definisi dari operasi aritmatika:

```
compadd =  $\vdash_{def} \forall a b. \text{compadd } a b = (\text{FST } a + \text{FST } b, \text{SND } a + \text{SND } b)$   
compsub =  $\vdash_{def} \forall a b. \text{compsub } a b = (\text{FST } a - \text{FST } b, \text{SND } a - \text{SND } b)$   
compmul =  
   $\vdash_{def} \forall a b.$   
    compmul a b =  
       $(\text{FST } a * \text{FST } b - \text{SND } a * \text{SND } b, \text{FST } a * \text{SND } b - \text{SND } a * \text{FST } b)$ 
```

- Operasi aritmatika pada bilangan kompleks:

```

complex_add =
  ⊢def ∀ w z. w complex_add z = complex (compadd (coords w) (coords z))

complex_sub =
  ⊢def ∀ w z. w complex_sub z = complex (compsub (coords w) (coords z))

complex_mul =
  ⊢def ∀ w z. w complex_mul z = complex (compmul (coords w) (coords z))

```

- Operasi aritmatika pada bilangan kompleks:

```

complex_add =
  ⊢def ∀ w z. w complex_add z = complex (compadd (coords w) (coords z))

complex_sub =
  ⊢def ∀ w z. w complex_sub z = complex (compsub (coords w) (coords z))

complex_mul =
  ⊢def ∀ w z. w complex_mul z = complex (compmul (coords w) (coords z))

```

- Definisi dari jumlah finite pada bilangan kompleks:

```

complex_sumc =
  ⊢def (∀ n f. complex_sumc n 0 f = complex (0,0)) ∧
  ∀ n m f. complex_sumc n (SUC m) f = complex_sumc n m f + f (n + m)

COMPLEX_SUM_DEF = ⊢def ∀ m n f. complex_sum (m,n) f = complex_sumc m n f

complex_sum =
  ⊢def ∀ f n n.
  (complex_sum (n,0) f = complex (0,0)) ∧
  (complex_sum (n,SUC m) f = complex_sum (n,m) f + f (n + m))

```

- Prinsip Akar-N pada unity:

```

principal_root_1 =
  ⊢def ∀ n N.
  principal_root_1 n N =
  complex (cos -2 * pi * & n / & N, sin -2 * pi * & n / & N)

```

- Discrete Fourier Transform:

```
DFT =
  ⊢def ∀ x N.
    DFT x N =
      (λ p. complex_sum (0,N) (λ i. x i * principal_root_1 (i * p) N))
```

- Ekspansi dari bilangan natural kedalam bentuk biner:

```
DIG = ⊢def ∀ n m. DIG n m = (m DIV 2 ** n) MOD 2

Binary_Form = ⊢def ∀ p m. Binary_Form p m = (λ k. DIG (m - 1 - k) p)
```

- Logaritma basis dua:

```
Log_2 = ⊢def ∀ p. Log_2 p = @k. p = 2 ** k
```

- Definisi dari jumlah finite pada bilangan natural:

```
num_sumc =
  ⊢def (∀ n f. num_sumc n 0 f = 0) ∧
    (∀ n m f. num_sumc n (SUC m) f = num_sumc n m f + f (n + m))

NUM_SUM_DEF = ⊢def ∀ m n f. num_sum (m,n) f = num_sumc m n f

num_sum =
  ⊢def ∀ f n m.
    (num_sum (n,0) f = 0) ∧
    (num_sum (n,SUC m) f = num_sum (n,m) f + f (n + m))
```

- Pangkat dari akar-N pada unity:

```
Z =
  ⊢def ∀ k p N. Z k p N = 2 ** k * num_sum (k,Log_2 N - k)
    (λ i. 2 ** (Log_2 N - 1 - i) * DIG i p) -
    2 ** (Log_2 N - 1) * DIG k p
```

- Urutan dari bentuk biner yang diatur kembali dari bilangan natural:

```
p_star = ⊢def ∀ p m. p_star p m = num_sum (0,m) (λ i. 2 ** m * DIG i p)
```

- Orde N algoritma fast Fouriertransform dalam domain real ideal:

```

FFT =
  ⊢def (∀ x N. FFT x N 0 = (λ p. x p)) ∧
    ∀ x N k.
      FFT x N (SUC k) =
        (λ p.
          (if DIG k p = 0 then
            FFT x N k p + FFT x N k (p + 2 ** (Log2 N - 1 - k))
          else
            (FFT x N k (p - 2 ** (Log2 N - 1 - k)) - FFT x N k p) *
              OMEGA k p N))

```

- Perhitungan untuk bagian real dan imajiner dari algoritma FFT:

$$\text{FFT\_REAL} = \vdash_{\text{def}} \forall x N k p. \text{FFT\_REAL } x N k p = \text{Re} (\text{FFT } x N k p)$$

$$\text{FFT\_IMAGE} = \vdash_{\text{def}} \forall x N k p. \text{FFT\_IMAGE } x N k p = \text{Im} (\text{FFT } x N k p)$$

- Bagian Real dan imajiner dari pangkat dari prinsip akar-N pada unity:

$$\text{OMEGA\_REAL} = \vdash_{\text{def}} \forall k p N. \text{OMEGA\_REAL } k p N = \text{Re} (\text{OMEGA } k p N)$$

$$\text{OMEGA\_IMAGE} = \vdash_{\text{def}} \forall k p N. \text{OMEGA\_IMAGE } k p N = \text{Im} (\text{OMEGA } k p N)$$

- Penulisan kembali algoritma FFT menggunakan bagian real dan imajiner:

```

Lemma 1:
  ∀ x N k p.
  (if DIG k p = 0 then
    (FFT_REAL x N (SUC k) p =
      FFT_REAL x N k p +
      FFT_REAL x N k (p + 2 ** (Log_2 N - 1 - k))) ∧
    (FFT_IMAGE x N (SUC k) p =
      FFT_IMAGE x N k p +
      FFT_IMAGE x N k (p + 2 ** (Log_2 N - 1 - k)))
  else
    (FFT_REAL x N (SUC k) p =
      (FFT_REAL x N k (p - 2 ** (Log_2 N - 1 - k)) -
      FFT_REAL x N k p) * OMEGA_REAL k p N -
      (FFT_IMAGE x N k (p - 2 ** (Log_2 N - 1 - k)) -
      FFT_IMAGE x N k p) * OMEGA_IMAGE k p N) ∧
    (FFT_IMAGE x N (SUC k) p =
      (FFT_IMAGE x N k (p - 2 ** (Log_2 N - 1 - k)) -
      FFT_IMAGE x N k p) * OMEGA_REAL k p N +
      (FFT_REAL x N k (p - 2 ** (Log_2 N - 1 - k)) -
      FFT_REAL x N k p) * OMEGA_IMAGE k p N)
  )

```

## 2. Analisis error FFT real ke floating-point:

- Mengkonstruksi bilangan kompleks pada bilangan floating-point:

```

float_complex_tybij =
  ⊢def (∀ a. float_complex (float_coords a) = a) ∧
  ∀ r. K T r = (float_coords (float_complex r) = r)

```

- Bagian real dan imajiner dari bilangan kompleks floating-point:

```

float_Re = ⊢def ∀ z. float_Re z = FST (float_coords z)

```

```

float_Im = ⊢def ∀ z. float_Im z = SND (float_coords z)

```

- Operasi aritmatika pada bilangan kompleks floating-point:

```

float_compadd =
  ⊢def ∀ a b. float_compadd a b = (FST a + FST b, SND a + SND b)

float_compsub =
  ⊢def ∀ a b. float_compsub a b = (FST a - FST b, SND a - SND b)

float_compmul =
  ⊢def ∀ a b.
    float_compmul a b =
      (FST a * FST b - SND a * SND b, FST a * SND b - SND a * FST b)

float_complex_add =
  ⊢def ∀ w z.
    w float_complex_add z =
      float_complex (float_compadd (float_coords w) (float_coords z))

float_complex_sub =
  ⊢def ∀ w z.
    w float_complex_sub z =
      float_complex (float_compsub (float_coords w) (float_coords z))

float_complex_mul =
  ⊢def ∀ w z.
    w float_complex_mul z =
      float_complex (float_compmul (float_coords w) (float_coords z))

```

- Definisi dari jumlah finite pada bilangan kompleks floating-point:

```

float_complex_sumc =
  ⊢def (∀ n f.
    float_complex_sumc n 0 f =
      float_complex (float (0,0,0), float (0,0,0))) ∧
    ∀ n m f.
      float_complex_sumc n (SUC m) f =
        float_complex_sumc n m f + f (n + m)

FLOAT_COMPLEX_SUM_DEF =
  ⊢def ∀ m n f. float_complex_sum (m,n) f = float_complex_sumc m n f

float_complex_sum =
  ⊢def ∀ f n m.
    (float_complex_sum (n,0) f =
      float_complex (float (0,0,0), float (0,0,0))) ∧
    (float_complex_sum (n,SUC m) f =
      float_complex_sum (n,m) f + f (n + m))

```

- Pembulatan ke format kompleks floating-point:

```
float_complex_round =
  ⊢def ∀ z.
    float_complex_round z =
      float_complex
        (float (round float_format To_nearest (Re z)),
         float (round float_format To_nearest (Im z)))
```

- Penilaian bilangan kompleks floating-point:

```
float_complex_Val =
  ⊢def ∀ z. float_complex_Val z =
    complex (Val (float_Re z), Val (float_Im z))
```

- Pangkat kompleks floating-point dari principal akar\_N pada unity:

```
FLOAT_OMEGA =
  ⊢def ∀ k p N. FLOAT_OMEGA k p N = float_complex_round (OMEGA k p N)
```

- Order N algoritma fast Fourier dalam domain floating-point yang nyata:

```
FLOAT_FFT =
  ⊢def (∀ x N. FLOAT_FFT x N 0 = (λ p. float_complex_round (x p))) ∧
  ∀ x N k.
    FLOAT_FFT x N (SUC k) =
      (λ p.
        (if DIG k p = 0 then
          FLOAT_FFT x N k p +
          FLOAT_FFT x N k (p + 2 ** (Log_2 N - 1 - k))
        else
          (FLOAT_FFT x N k (p - 2 ** (Log_2 N - 1 - k)) -
           FLOAT_FFT x N k p) * FLOAT_OMEGA k p N))
```

- Perhitungan error dari elemen ke-p dari floating-point FFT pada langkah k:

```
FLOAT_FFT_ERROR =
  ⊢def ∀ x N k p.
    FLOAT_FFT_ERROR x N k p =
      float_complex_Val (FLOAT_FFT x N k p) - FFT x N k p
```

- Error dalam perhitungan dari koefisien Fourier menggunakan floating-point FFT:

```

FLOAT_FFT_FINAL_ERROR =
   $\vdash_{def} \forall x N p.$ 
    FLOAT_FFT_FINAL_ERROR x N p =
    FLOAT_FFT_ERROR x N (Log_2 N) (p_star p (Log_2 N))

```

- Bagian real dan imajiner dari algoritma floating-point:

```

FLOAT_FFT_REAL =
   $\vdash_{def} \forall x N k p.$  FLOAT_FFT_REAL x N k p = float_Re (FLOAT_FFT x N k p)

FLOAT_FFT_IMAGE =
   $\vdash_{def} \forall x N k p.$  FLOAT_FFT_IMAGE x N k p = float_Im (FLOAT_FFT x N k p)

```

- Bagian real dan imajiner dari pangkat kompleks floating point dari principal akar-N pada unity:

```

FLOAT_OMEGA_REAL =
   $\vdash_{def} \forall k p N.$  FLOAT_OMEGA_REAL k p N = float_Re (FLOAT_OMEGA k p N)

FLOAT_OMEGA_IMAGE =
   $\vdash_{def} \forall k p N.$  FLOAT_OMEGA_IMAGE k p N = float_Im (FLOAT_OMEGA k p N)

```

- Penulisan kembali algoritma floating-point menggunakan bagian real dan imajiner:

Lemma 2:

$\forall x \in \mathbb{C}^N$

(if  $\text{DIG}(k, p) = 0$  then

$(\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } (\text{SUC } k) \text{ } p =$

$\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } k \text{ } p +$

$\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } k \text{ } (p + 2^{**} (\text{Log}_2 N - 1 - k))) \wedge$

$(\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } (\text{SUC } k) \text{ } p =$

$\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } k \text{ } p +$

$\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } k \text{ } (p + 2^{**} (\text{Log}_2 N - 1 - k)))$ )

else

$(\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } (\text{SUC } k) \text{ } p =$

$(\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } k \text{ } (p - 2^{**} (\text{Log}_2 N - 1 - k)) -$

$\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } k \text{ } p) * \text{FLOAT\_OMEGA\_REAL } k \text{ } p \text{ } N -$

$(\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } k \text{ } (p - 2^{**} (\text{Log}_2 N - 1 - k)) -$

$\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } k \text{ } p) * \text{FLOAT\_OMEGA\_IMAGE } k \text{ } p \text{ } N) \wedge$

$(\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } (\text{SUC } k) \text{ } p =$

$(\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } k \text{ } (p - 2^{**} (\text{Log}_2 N - 1 - k)) -$

$\text{FLOAT\_FFT\_IMAGE } x \text{ } N \text{ } k \text{ } p) * \text{FLOAT\_OMEGA\_REAL } k \text{ } p \text{ } N +$

$(\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } k \text{ } (p - 2^{**} (\text{Log}_2 N - 1 - k)) -$

$\text{FLOAT\_FFT\_REAL } x \text{ } N \text{ } k \text{ } p) * \text{FLOAT\_OMEGA\_IMAGE } k \text{ } p \text{ } N)$ )

- Penjelasan suatu error untuk setiap langkah-langkah aritmatika dalam bagian real dan imajiner dari algoritma floating-point FFT:

Lemma 3:

```

∀ x N k p.
  ∃ e.
    ∀ i.
      1 ≤ i ∧ i ≤ 12 ⇒
        e i ≤ 1 / 2 pow 24 ∧
        (if DIG k p = 0 then
          (Val (FLOAT_FFT_REAL x N (SUC k) p) =
            (Val (FLOAT_FFT_REAL x N k p) +
              Val
                (FLOAT_FFT_REAL x N k (p + 2 ** (Log_2 N - 1 - k)))) *
              (1 + e 1)) ∧
          (Val (FLOAT_FFT_IMAGE x N (SUC k) p) =
            (Val (FLOAT_FFT_IMAGE x N k p) +
              Val
                (FLOAT_FFT_IMAGE x N k
                  (p + 2 ** (Log_2 N - 1 - k)))) * (1 + e 2))
        else
          (Val (FLOAT_FFT_REAL x N (SUC k) p) =
            (Val
              (FLOAT_FFT_REAL x N k (p - 2 ** (Log_2 N - 1 - k))) -
              Val (FLOAT_FFT_REAL x N k p)) *
            Val (FLOAT_OMEGA_REAL k p N) * (1 + e 3) * (1 + e 4) *
            (1 + e 5) -
            (Val
              (FLOAT_FFT_IMAGE x N k (p - 2 ** (Log_2 N - 1 - k))) -
              Val (FLOAT_FFT_IMAGE x N k p)) *
            Val (FLOAT_OMEGA_IMAGE k p N) * (1 + e 6) * (1 + e 7) *
            (1 + e 5)) ∧

```

- Penulisan kembali error dari elemen ke-p dari floating-point pada setiap k sebagai bilangan kompleks:

Lemma 4:

```

∀ x N k p.
  FLOAT_FFT_ERROR x N k p =
  complex
    (Val (FLOAT_FFT_REAL x N k p) - FFT_REAL x N k p,
     Val (FLOAT_FFT_IMAGE x N k p) - FFT_IMAGE x N k p)

```

- Akumulasi dari error round-off dalam floating-point FFT:

```

Lemma 5:
  ∀x N k p.
  (FLOAT_FFT_ERROR x N 0 p = complex (0,0)) ∧
  ∃f.
  (FLOAT_FFT_ERROR x N (SUC k) p =
   (if DIG k p = 0 then
    FLOAT_FFT_ERROR x N k p +
    FLOAT_FFT_ERROR x N k (p + 2 ** (Log_2 N - 1 - k))
  f x N k p
   else
    (FLOAT_FFT_ERROR x N k (p - 2 ** (Log_2 N - 1 - k))
     FLOAT_FFT_ERROR x N k p) * OMEGA k p N + f x N k
  f x N k p))
  ∃e.
  ∀i.
  1 ≤ i ∧ i ≤ 12 ⇒
  e i ≤ 1 / 2 pow 24 ∧
  (f x N k p =
   (if DIG k p = 0 then
    complex
    (e 1 *
     (Val (FLOAT_FFT_REAL x N k p) *
      Val
      (FLOAT_FFT_REAL x N k
       (p + 2 ** (Log_2 N - 1 - k))))),
     e 2 *
     (Val (FLOAT_FFT_IMAGE x N k p) +
      Val
      (FLOAT_FFT_IMAGE x N k
       (p + 2 ** (Log_2 N - 1 - k))))))
   else
  )

```

```

complex
  (((1 + e 3) * (1 + e 4) * (1 + e 5) - 1) *
   (Val
     (FLOAT_FFT_REAL x N k
      (p - 2 ** (Log_2 N - 1 - k))) -
    Val (FLOAT_FFT_REAL x N k p)) * OMEGA_REAL k p N -
   ((1 + e 6) * (1 + e 7) * (1 + e 5) - 1) *
   (Val
     (FLOAT_FFT_IMAGE x N k
      (p - 2 ** (Log_2 N - 1 - k))) -
     FFT_IMAGE x N k p) * OMEGA_IMAGE k p N,
   ((1 + e 8) * (1 + e 9) * (1 + e 10) - 1) *
   (Val
     (FLOAT_FFT_IMAGE x N k
      (p - 2 ** (Log_2 N - 1 - k))) -
    Val (FLOAT_FFT_IMAGE x N k p)) * OMEGA_REAL k p N -
   ((1 + e 11) * (1 + e 12) * (1 + e 10) - 1) *
   (Val
     (FLOAT_FFT_REAL x N k
      (p - 2 ** (Log_2 N - 1 - k))) -
    Val (FLOAT_FFT_REAL x N k p)) *
   OMEGA_IMAGE k p N)))

```

### 3. Analisis error FFT rela to fixed-point:

- Konstruksi bilangan complex pada bilangan fixed-point:

```

fxp_complex_tybij =
  ⊢def (∀ a. fxp_complex (fxp_coords a) = a) ∧
  ∀ r. K T r ⇒ (fxp_coords (fxp_complex r) = r)

```

- Bagian real dan imajiner dari bilangan kompleks fixed-point:

```

fxp_Re = ⊢def ∀ z. fxp_Re z = FST (fxp_coords z)

```

```

fxp_Im = ⊢def ∀ z. fxp_Im z = SND (fxp_coords z)

```

- Operasi aritmatika pada bilangan kompleks fixed-point:

```

FxpAdd =
  ⊢def ∀ X a b. FxpAdd X a b = FST (fxpAdd X Convergent Clip a b)

FxpSub =
  ⊢def ∀ X a b. FxpSub X a b = FST (fxpSub X Convergent Clip a b)

FxpMul =
  ⊢def ∀ X a b. FxpMul X a b = FST (fxpMul X Convergent Clip a b)

```

```

fxp_compadd =
  ⊢def ∀ X a b.
    fxp_compadd X a b =
      (FxpAdd X (FST a) (FST b), FxpAdd X (SND a) (SND b))

fxp_compsub =
  ⊢def ∀ X a b.
    fxp_compsub X a b =
      (FxpSub X (FST a) (FST b), FxpSub X (SND a) (SND b))

fxp_compmul =
  ⊢def ∀ X a b.
    fxp_compmul X a b =
      (FxpSub X (FxpMul X (FST a) (FST b)) (FxpMul X (SND a) (SND b)),
       FxpSub X (FxpMul X (FST a) (SND b)) (FxpMul X (SND a) (FST b)))

fxp_complex_add =
  ⊢def ∀ X w z.
    fxp_complex_add X w z =
      fxp_complex (fxp_compadd X (fxp_coords w) (fxp_coords z))

fxp_complex_sub =
  ⊢def ∀ X w z.
    fxp_complex_sub X w z =
      fxp_complex (fxp_compsub X (fxp_coords w) (fxp_coords z))

float_complex_mul =
  ⊢def ∀ X w z.
    fxp_complex_mul X w z =
      fxp_complex (fxp_compmul X (fxp_coords w) (fxp_coords z))

```

- Definisi dari jumlah finite pada bilangan kompleks fixed-point:

```

fxp_complex_sumc =
  ⊢def (∀ n X f.
    fxp_complex_sumc n 0 X f =
    fxp_complex
      (fxp (WORD (REPLICATE (streamlength X) F),X),
       fxp (WORD (REPLICATE (streamlength X) F),X)) ∧
    ∀ n m X f.
    fxp_complex_sumc n (SUC m) X f =
    fxp_complex_add X (fxp_complex_sumc n m X f) (f (n + m))

FXP_COMPLEX_SUM_DEF =
  ⊢def ∀ m n X f. fxp_complex_sum (m,n) X f = fxp_complex_sumc m n X f

fxp_complex_sum =
  ⊢def ∀ X f n m.
    (fxp_complex_sum (n,0) X f =
    fxp_complex
      (fxp (WORD (REPLICATE (streamlength X) F),X),
       fxp (WORD (REPLICATE (streamlength X) F),X)) ∧
    (fxp_complex_sum (n,SUC m) X f =
    fxp_complex_add X (fxp_complex_sumc n m X f) (f (n + m))))

```

- Rounding pada format kompleks fixed-point:

```

Fxp_round =
  ⊢def ∀ X x. Fxp_round X x = FST (fxp_round X Convergent x Clip)

fxp_complex_round =
  ⊢def ∀ X z.
    fxp_complex_round X z =
    fxp_complex (Fxp_round X (Re z), Fxp_round X (Im z))

```

- Unit imajiner Fixed-point:

```

fxp_ii = ⊢def ∀ X. fxp_ii X = fxp_complex_round X ii

```

- Penilaian bilangan kompleks fixed-point:

```

fxp_complex_value =
  ⊢def ∀ z. fxp_complex_value z =
    complex (value (fxp_Re z), value (fxp_Im z))

```

- Pangkat kompleks fixed-point dari principal akar-N pad unity:

```
FXP_OMEGA =
  ⊢def ∀ X k p N. FXP_OMEGA X k p N =
    fxp_complex_round X (OMEGA k p N)
```

- N urutan algoritma fast Fourier transform dalam domain nyata fixed-point:

```
FXP_FFT =
  ⊢def (∀ X x N. FXP_FFT X x N 0 = (λ p. fxp_complex_round X (x p))) ∧
  ∀ X x N k.
    FXP_FFT X x N (SUC k) =
      (λ p.
        (if DIG k p = 0 then
          fxp_complex_add X (FXP_FFT X x N k p)
            (FXP_FFT X x N k (p + 2 ** (Log_2 N - 1 - k)))
        else
          fxp_complex_sub X
            (fxp_complex_mul X
              (FXP_FFT X x N k (p - 2 ** (Log_2 N - 1 - k)))
              (FXP_OMEGA X k p N))
            (fxp_complex_mul X (FXP_FFT X x N k p)
              (FXP_OMEGA X k p N))))
```

- Perhitungan error dari elemen ke-p dari fixed-point pada langkah k:

```
FXP_FFT_ERROR =
  ⊢def ∀ X x N k p.
    FXP_FFT_ERROR X x N k p =
      fxp_complex_value (FXP_FFT X x N k p) - FFT x N k p
```

- Error dalam perhitungan dari koefisien Fourier menggunakan fixed-point FFT:

```
FXP_FFT_FINAL_ERROR =
  ⊢def ∀ X x N p.
    FXP_FFT_FINAL_ERROR X x N p =
      FXP_FFT_ERROR X x N (Log_2 N) (p_star p (Log_2 N))
```

- Kalkulasi dari bagian real dan imajiner dari algoritma fixed-point FFT:

```

FXP_FFT_REAL =
  ⊢def ∀ X x N k p. FXP_FFT_REAL X x N k p = fxp_Re (FXP_FFT X x N k p)

FXP_FFT_IMAGE =
  ⊢def ∀ X x N k p. FXP_FFT_IMAGE X x N k p = fxp_Im (FXP_FFT X x N k p)

```

- Bagian real dan imajiner dari pangkat kompleks fixed-point dari principal akar-N pada unity:

```

FXP_OMEGA_REAL =
  ⊢def ∀ X k p N. FXP_OMEGA_REAL X k p N = fxp_Re (FXP_OMEGA X k p N)

FXP_OMEGA_IMAGE =
  ⊢def ∀ X k p N. FXP_OMEGA_IMAGE X k p N = fxp_Im (FXP_OMEGA X k p N)

```

- Penulisan kembali algoritma fixed-point FFT menggunakan bagian real dan imajiner:

```

Lemma 6:
  ∀ X x N k p.
    (if DIG k p = 0 then
      (FXP_FFT_REAL X x N (SUC k) p =
        FxpAdd X (FXP_FFT_REAL X x N k p)
          (FXP_FFT_REAL X x N k (p + 2 ** (Log_2 N - 1 - k)))) ^
      (FXP_FFT_IMAGE X x N (SUC k) p =
        FxpAdd X (FXP_FFT_IMAGE X x N k p)
          (FXP_FFT_IMAGE X x N k (p + 2 ** (Log_2 N - 1 - k))))
    else
      (FXP_FFT_REAL X x N (SUC k) p =
        FxpSub X
          (FxpSub X
            (FxpMul X
              (FXP_FFT_REAL X x N k (p - 2 ** (Log_2 N - 1 - k)))
              (FXP_OMEGA_REAL X k p N))
            (FxpMul X
              (FXP_FFT_IMAGE X x N k (p - 2 ** (Log_2 N - 1 - k)))
              (FXP_OMEGA_IMAGE X k p N)))
          (FxpSub X
            (FxpMul X (FXP_FFT_REAL X x N k p)
              (FXP_OMEGA_REAL X k p N))
            (FxpMul X (FXP_FFT_IMAGE X x N k p)
              (FXP_OMEGA_IMAGE X k p N)))) ^

```

```

(FXP_FFT_IMAGE X x N (SUC k) p =
  FxpSub X
    (FxpAdd X
      (FxpMul X
        (FXP_FFT_IMAGE X x N k (p - 2 ** (Log_2 N - 1 - k)))
        (FXP_OMEGA_REAL X k p N))
      (FxpMul X
        (FXP_FFT_REAL X x N k (p - 2 ** (Log_2 N - 1 - k)))
        (FXP_OMEGA_IMAGE X k p N)))
    (FxpAdd X
      (FxpMul X (FXP_FFT_IMAGE X x N k p)
        (FXP_OMEGA_REAL X k p N))
      (FxpMul X (FXP_FFT_REAL X x N k p)
        (FXP_OMEGA_IMAGE X k p N))))

```

- Penjelasan suatu error untuk setiap dari langkah aritmatika dalam bagian real dan imajiner dari algoritma fixed-point FFT;

Lemma 7:

```

∀ X x N k p.
  ∃ e'.
    ∀ i.
      1 ≤ i ∧ i ≤ 12 ⇒
        e' i ≤ 1 / 2 pow frachbits X ∧
        (if DIG k p = 0 then
          (value (FXP_FFT_REAL X x N (SUC k) p) =
            value (FXP_FFT_REAL X x N k p) +
            value
              (FXP_FFT_REAL X x N k (p + 2 ** (Log_2 N - 1 - k))) +
            e' 1) ∧
          (value (FXP_FFT_IMAGE X x N (SUC k) p) =
            value (FXP_FFT_IMAGE X x N k p) +
            value
              (FXP_FFT_IMAGE X x N k (p + 2 ** (Log_2 N - 1 - k))) +
            e' 2)
        else
          (value (FXP_FFT_REAL X x N (SUC k) p) =
            (value
              (FXP_FFT_REAL X x N k (p - 2 ** (Log_2 N - 1 - k))) -
            value (FXP_FFT_REAL X x N k p) + e' 3) *
            value (FXP_OMEGA_REAL X k p N) + e' 4 -
            ((value
              (FXP_FFT_IMAGE X x N k
                (p - 2 ** (Log_2 N - 1 - k))) -
            value (FXP_FFT_IMAGE X x N k p) + e' 5) *
            value (FXP_OMEGA_IMAGE X k p N) + e' 6) + e' 7) ∧

```

```

(value (FXP_FFT_IMAGE X x N (SUC k) p) =
  (value
    (FXP_FFT_IMAGE X x N k (p - 2 ** (Log_2 N - 1 - k))) -
    value (FXP_FFT_IMAGE X x N k p) + e' 8) *
    value (FXP_OMEGA_REAL X k p N) + e' 9 +
    ((value
      (FXP_FFT_REAL X x N k (p - 2 ** (Log_2 N - 1 - k))) -
      value (FXP_FFT_REAL X x N k p) + e' 10) *
      value (FXP_OMEGA_IMAGE X k p N) + e' 11) + e' 12))

```

- Penulisan kembali error dari elemen ke-p dari fixed-point FFT pada langkah k sebagai bilangan kompleks:

Lemma 8:

```

∀ X x N k p.
FXP_FFT_ERROR X x N k p =
complex
  (value (FXP_FFT_REAL X x N k p) - FFT_REAL x N k p,
   value (FXP_FFT_IMAGE X x N k p) - FFT_IMAGE x N k p)

```

- Akumulasi dari error round-off dalam fixed-point FFT:

Lemma 9:

```

∀ X x N k p.
(FXP_FFT_ERROR X x N 0 p ⇒ complex (0,0)) ∧
∃ f'.
(FXP_FFT_ERROR X x N (SUC k) p =
  (if DIG k p = 0 then
    FXP_FFT_ERROR X x N k p +
    FXP_FFT_ERROR X x N k (p + 2 ** (LOG_2 N - 1 - k)) +
    f' X x N k p
  else
    (FXP_FFT_ERROR X x N k (p - 2 ** (LOG_2 N - 1 - k)) -
     FXP_FFT_ERROR X x N k p) * OMEGA k p N + f' X x N k p)) ∧
∃ e'.
∀ i.
1 ≤ i ∧ i ≤ 12 ⇒
e' i ≤ 1 / 2 pow fracbits X ∧
(f' X x N k p =
  (if DIG k p = 0 then
    complex (e' 1, e' 2)
  else
    complex
      (e' 3 * OMEGA_REAL k p N + e' 4 -
       e' 5 * OMEGA_IMAGE k p N - e' 6 + e' 7,
       e' 8 * OMEGA_REAL k p N + e' 9 +
       e' 10 * OMEGA_IMAGE k p N + e' 11 + e' 12)))

```

- Floating-point ke error fixed-point FFT dari elemen ke-p pada langkah k:

```

FLOAT_TO_FXP_FFT_ERROR =
  ⊢def ∀ X x N k p.
    FLOAT_TO_FFT_ERROR X x N k p =
      fxp_complex_value (FXP_FFT X x N k p) -
      float_complex_Val (FLOAT_FFT x N k p)

```

- Akumulasi dari error round-off dalam transisi dari floating-point ke fixed-point FFT:

```

Lemma 10:
  ∀ X x N k p.
    (FLOAT_TO_FXP_FFT_ERROR X x N 0 p = complex (0,0)) ∧
    ∃ f f'.
      FLOAT_TO_FXP_FFT_ERROR X x N (SUC k) p =
        (if DIG k p = 0 then
          FLOAT_TO_FXP_FFT_ERROR X x N k p +
          FLOAT_TO_FXP_FFT_ERROR X x N k
            (p + 2 ** (LOG_2 N - 1 - k)) + f' X x N k p - f x N k p
        else
          (FLOAT_TO_FXP_FFT_ERROR X x N k
            (p - 2 ** (LOG_2 N - 1 - k)) -
          FLOAT_TO_FXP_FFT_ERROR X x N k p) * OMEGA k p N +
          f' X x N k p - f x N k p)

```

#### 4. Verifikasi FFT RTL ke fixed-point (16 point radix-4 DIF FFT):

- Deskripsi RTL dari operasi aritmatika:
- N-bit adder 2'complement dalam level RTL:

```

xor =  $\vdash_{def} \forall a b. a \text{ xor } b = \neg a \wedge b \vee a \wedge \neg b$ 

carry =
 $\vdash_{def} (\forall a b. \text{carry } 0 \ a \ b = F) \wedge$ 
 $\forall i \ a \ b.$ 
 $\text{carry } (\text{SUC } i) \ a \ b =$ 
 $\text{BIT } i \ a \wedge \text{BIT } i \ b \vee \text{BIT } i \ a \wedge \text{carry } i \ a \ b \vee$ 
 $\text{BIT } i \ b \wedge \text{carry } i \ a \ b$ 

N_add_RTL =
 $\vdash_{def} \forall N \ a \ b \ q.$ 
 $\text{N\_add\_RTL } N \ a \ b \ q =$ 
 $(\forall i.$ 
 $i < N \wedge 0 \leq i \implies \text{BIT } i \ q =$ 
 $\text{BIT } i \ a \ \text{xor} \ (\text{BIT } i \ b \ \text{xor} \ \text{carry } i \ a \ b)) \wedge$ 
 $(\text{BIT } N \ q = \text{carry } N \ a \ b)$ 

```

- Verifikasi dari RTL ke level fixed-point dari penambahan N-bit:

```

FXP =  $\vdash_{def} \forall N \ a. \text{FXP } N \ a = \text{fxp } (a, N, N - 1, 1)$ 

FXP_VECT =  $\vdash_{def} \forall N \ a. \text{FXP\_VECT } N \ a = @ \ b. \forall i. b \ i = \text{FXP } N \ (a \ i)$ 

FXP_VECT_COMPLEX =
 $\vdash_{def} \forall N \ a \ b.$ 
 $\text{FXP\_VECT\_COMPLEX } N \ a \ b =$ 
 $@ \ c. \forall i. c \ i = \text{fxp\_complex } (\text{FXP } N \ (a \ i), \text{FXP } N \ (b \ i))$ 

FXP_VECT_COMPLEX_NUM =
 $\vdash_{def} \forall N \ a \ b.$ 
 $\text{FXP\_VECT\_COMPLEX\_NUM } N \ a \ b =$ 
 $@ \ c. \forall i. c \ i = \text{FXP\_VECT\_COMPLEX } N \ (a \ i) \ (b \ i)$ 

Lemma 11:
 $\forall N \ a \ b \ q.$ 
 $\text{N\_add\_RTL } N \ a \ b \ q \implies$ 
 $(\text{FXP } N \ q = \text{FxpAdd } (N, N - 1, 1) \ (\text{FXP } N \ a) \ (\text{FXP } N \ b))$ 

```

- Penambahan kompleks dalam level RTL:

```

N_complex_add_RTL =
  ⊢def ∀ N ar ai br bi qr qi.
    N_complex_add_RTL N ar ai br bi qr qi =
    N_add_RTL N ar br qr ∧ N_add_RTL N ai bi qi

```

- Verifikasi dari RTL ke level fixed-point dari adder kompleks N-bit:

Lemma 12:

```

∀ N ar ai br bi qr qi.
  N_complex_add_RTL N ar ai br bi qr qi ⇒
  (fxp_complex (FXP N qr, FXP N qi) =
   fxp_complex_add (N, N - 1, 1) (fxp_complex (FXP N ar, FXP N ai))
   (fxp_complex (FXP N br, FXP N bi)))

```

- N-bit subtractor 2's complement dalam level RTL:

```

borrow =
  ⊢def (∀ a b. borrow 0 a b = T) ∧
  ∀ i a b.
    borrow (SUC i) a b =
    BIT i a ∧ ¬ BIT i b ∨ BIT i a ∧ borrow i a b ∨
    ¬ BIT i b ∧ borrow i a b

N_sub_RTL =
  ⊢def ∀ N a b q.
    N_sub_RTL N a b q =
    (∀ i.
     i < N ∧ 0 ≤ i ⇒ BIT i q =
     BIT i a xor ¬ BIT i b xor borrow i a b) ∧
    (BIT N q = borrow N a b)

```

- Verifikasi dari RTL ke level fixed-point dari subtractor N-bit:

Lemma 13:

```

∀ N a b q.
  N_sub_RTL N a b q ⇒
  (FXP N q = FxpSub (N, N - 1, 1) (FXP N a) (FXP N b))

```

- Complex subtraction dalam level RTL:

```

N_complex_sub_RTL =
  ⊢def ∀ N ar ai br bi qr qi.
    N_complex_sub_RTL N ar ai br bi qr qi =
      N_sub_RTL N ar br qr ∧ N_sub_RTL N ai bi qi

```

- Verifikasi dari RTL ke level fixed-point dari subtracter komplek N-bit:

Lemma 14:

```

∀ N ar ai br bi qr qi.
  N_complex_sub_RTL N ar ai br bi qr qi ⇒
  (fxp_complex (FXP N qr, FXP N qi) =
   fxp_complex_sub (N, N - 1, 1) (fxp_complex (FXP N ar, FXP N ai))
   (fxp_complex (FXP N br, FXP N bi)))

```

- Multiplier N-bit 2' complement dalam level RTL:

```

N_mul_unsigned =
  ⊢def ∀ N a b q.
    N_mul_unsigned N a b q = (q = BWORD (BINVAL a * BINVAL b) (2 * N))

TWO_COMP = ⊢def ∀ N a. TWO_COMP N a = BWORD N (SUC (BINVAL (WNOT a)))

UN_SIGNED = ⊢def ∀ N a. UN_SIGNED N a = WSEG (N - 1) 0 a

CONVERT =
  ⊢def ∀ N a.
    CONVERT N a =
      (if MSB a then WSEG (N - 1) 0 (TWO_COMP N a) else UN_SIGNED N a)

N_mul_two_comp_RTL =
  ⊢def ∀ N a b q.
    N_mul_two_comp_RTL N a b q =
      (q =
       WCAT
        (WORD MSB a xor MSB b],
        BWORD (2 * N - 2)
        (BINVAL (CONVERT N a) * BINVAL (CONVERT N b) +
         BV (MSB a xor MSB b)))

```

- Verifikasi dari RTL ke level fixed-point dari multiplier N-bit:

Lemma 15:

$$\forall N \ a \ b \ q.$$

$$N\_mul\_two\_comp\_RTL \ N \ a \ b \ q \implies$$

$$(FXP \ N \ q = FxpMul \ (N, N - 1, 1) \ (FXP \ N \ a) \ (FXP \ N \ b))$$

- Perkalian kompleks dalam level RTL:

$N\_complex\_mul\_two\_comp\_RTL =$

$\vdash_{def} \forall N \ ar \ ai \ br \ bi \ qr \ qi.$

$N\_complex\_mul\_two\_comp\_RTL \ N \ ar \ ai \ br \ bi \ qr \ qi =$

$\exists s1 \ s2 \ s3 \ s4 \ s5 \ s6.$

$N\_sub\_RTL \ N \ ar \ bi \ s1 \wedge N\_sub\_RTL \ N \ br \ bi \ s2 \wedge$

$N\_add\_RTL \ N \ br \ bi \ s3 \wedge N\_mul\_two\_comp\_RTL \ N \ s1 \ bi \ s4 \wedge$

$N\_mul\_two\_comp\_RTL \ N \ s2 \ ar \ s5 \wedge N\_mul\_two\_comp\_RTL \ N \ s3 \ ai \ s6 \wedge$

$N\_add\_RTL \ N \ s5 \ s4 \ qr \wedge N\_add\_RTL \ N \ s6 \ s4 \ qi$

- Verifikasi dari RTL ke level perkalian kompleks N-bit:

Lemma 16:

$\forall N \ ar \ ai \ br \ bi \ qr \ qi.$

$N\_complex\_mul\_two\_comp\_RTL \ N \ ar \ ai \ br \ bi \ qr \ qi \implies$

$(fxp\_complex \ (FXP \ N \ qr, FXP \ N \ qi) =$

$fxp\_complex\_mul \ (N, N - 1, 1) \ (fxp\_complex \ (FXP \ N \ ar, FXP \ N \ ai))$

$(fxp\_complex \ (FXP \ N \ br, FXP \ N \ bi)))$

- Radix-4 butterfly dalam level fixed-point:

```
radix_4_butterfly_fxp =
  ⊢def ∀ X a b c d e f g h.
    radix_4_butterfly_fxp X a b c d e f g h =
      (e =
        fxp_complex_add X (fxp_complex_add X a b)
          (fxp_complex_add X c d)) ∧
      (f =
        fxp_complex_add X
          (fxp_complex_sub X a (fxp_complex_mul X b (fxp_ii X)))
          (fxp_complex_sub X (fxp_complex_mul X d (fxp_ii X)) c)) ∧
      (g =
        fxp_complex_add X (fxp_complex_sub X a b)
          (fxp_complex_sub X c d)) ∧
      (h =
        fxp_complex_sub X
          (fxp_complex_sub X a (fxp_complex_mul X b (fxp_ii X)))
          (fxp_complex_add X c (fxp_complex_mul X d (fxp_ii X))))
```

- Radix-4 butterfly dalam level RTL:

```
radix_4_butterfly_RTL =
  ⊢def ∀ N ar ai br bi cr ci dr di q1r q1i q2r q2i q3r q3i q4r q4i.
    radix_4_butterfly_RTL N ar ai br bi cr ci dr di q1r q1i q2r q2i q3r
      q3i q4r q4i =
      (∃ y1r y1i y2r y2i.
        N_complex_add_RTL N ar ai br bi y1r y1i ∧
        N_complex_add_RTL N cr ci dr di y2r y2i ∧
        N_complex_add_RTL N y1r y1i y2r y2i q1r q1i) ∧
```

```

(∃ y3r y3i y4r y4i y5r y5i y6r y6i.
N_complex_mul_two_comp_RTL N br bi (NEWWORD N 0) (NEWWORD N 1) y3r y3i ∧
N_complex_sub_RTL N ar ai y3r y3i y4r y4i ∧
N_complex_mul_two_comp_RTL N dr di (NEWWORD N 0) (NEWWORD N 1) y5r y5i ∧
N_complex_sub_RTL N y5r y5i cr ci y6r y6i ∧
N_complex_add_RTL N y4r y4i y6r y6i q2r q2i) ∧
(∃ y7r y7i y8r y8i.
N_complex_sub_RTL N ar ai br bi y7r y7i ∧
N_complex_sub_RTL N cr ci dr di y8r y8i ∧
N_complex_add_RTL N y7r y7i y8r y8i q3r q3i) ∧
∃ y9r y9i y10r y10i y11r y11i y12r y12i.
N_complex_mul_two_comp_RTL N br bi (NEWWORD N 0) (NEWWORD N 1) y9r y9i ∧
N_complex_add_RTL N ar ai y9r y9i y10r y10i ∧
N_complex_mul_two_comp_RTL N dr di (NEWWORD N 0) (NEWWORD N 1) y11r y11i ∧
N_complex_add_RTL N y11r y11i cr ci y12r y12i ∧
N_complex_sub_RTL N y10r y10i y12r y12i q4r q4i

```

- Verifikasi dari RTL ke level fixed-point dari radix-4 butterfly:

Lemma 17:

```

∀ N ar ai br bi cr ci dr di q1r q1i q2r q2i q3r q3i q4r q4i.
radix_4_butterfly_RTL N ar ai br bi cr ci dr di q1r q1i q2r q2i
q3r q3i q4r q4i ⇒
radix_4_butterfly_fxp (N,N - 1,1)
(fxp_complex (FXP N ar,FXP N ai))
(fxp_complex (FXP N br,FXP N bi))
(fxp_complex (FXP N cr,FXP N ci))
(fxp_complex (FXP N dr,FXP N di))
(fxp_complex (FXP N q1r,FXP N q1i))
(fxp_complex (FXP N q2r,FXP N q2i))
(fxp_complex (FXP N q3r,FXP N q3i))
(fxp_complex (FXP N q4r,FXP N q4i))

```

- Radix-4 dragonfly dalam level fixed-point:

```

radix_4_dragonfly_fxp =
  ⊢def ∀ X a b c d w e f g h.
radix_4_dragonfly_fxp X a b c d w e f g h =
∃ y1 y2 y3 y4.
radix_4_butterfly_fxp X a b c d y1 y2 y3 y4 ∧
(e = fxp_complex_mul X (w 1) y1) ∧
(f = fxp_complex_mul X (w 2) y2) ∧
(g = fxp_complex_mul X (w 3) y3) ∧
(h = fxp_complex_mul X (w 4) y4)

```

- Radix-4 dragonfly dalam level RTL:

```
radix_4_dragonfly RTL =
   $\Gamma_{def} \forall N \text{ ar ai br bi cr ci dr di wr wi q1r q1i q2r q2i q3r q3i q4r q4i.}$ 
    radix_4_dragonfly RTL N ar ai br bi cr ci dr di wr wi
    q1r q1i q2r q2i q3r q3i q4r q4i =
       $\exists s1 s2 s3 s4 s5 s6 s7 s8.$ 
        radix_4_butterfly RTL N ar ai br bi cr ci dr di
        s1 s2 s3 s4 s5 s6 s7 s8  $\wedge$ 
        N_complex_mul_two_comp RTL N s1 s2 (wr 1) (wi 1) q1r q1i  $\wedge$ 
        N_complex_mul_two_comp RTL N s3 s4 (wr 2) (wi 2) q2r q2i  $\wedge$ 
        N_complex_mul_two_comp RTL N s5 s6 (wr 3) (wi 3) q3r q3i  $\wedge$ 
        N_complex_mul_two_comp RTL N s7 s8 (wr 4) (wi 4) q4r q4i
```

- Verifikasi dari RTL ke level fixed-point dari radix-4 dragonfly:

```
Lemma 18:
   $\forall N \text{ ar ai br bi cr ci dr di q1r q1i q2r q2i q3r q3i q4r q4i wr wi.}$ 
    radix_4_dragonfly RTL N ar ai br bi cr ci dr di wr wi q1r q1i q2r
    q2i q3r q3i q4r q4i  $\implies$ 
    radix_4_dragonfly_fix (N,N - 1,1)
    (fxp_complex (FXP N ar,FXP N ai))
    (fxp_complex (FXP N br,FXP N bi))
    (fxp_complex (FXP N cr,FXP N ci))
    (fxp_complex (FXP N dr,FXP N di)) (FXP_VECT_COMPLEX N wr wi)
    (fxp_complex (FXP N q1r,FXP N q1i))
    (fxp_complex (FXP N q2r,FXP N q2i))
    (fxp_complex (FXP N q3r,FXP N q3i))
    (fxp_complex (FXP N q4r,FXP N q4i))
```

- Radix-4 16-point decimation-in-frequency dalam level fixed-point:

```

radix_4_16_point_DIF_FFT_fxp =
  ⊢def ∀ N x A w.
    radix_4_16_point_DIF_FFT_fxp N x A w =
      ∃ A1.
        radix_4_dragonfly_fxp (N,N - 1,1) (x 0) (x 4) (x 8) (x 12) (w 0)
          (A1 0) (A1 4) (A1 8) (A1 12) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (x 1) (x 5) (x 9) (x 13) (w 1)
          (A1 1) (A1 5) (A1 9) (A1 13) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (x 2) (x 6) (x 10) (x 14) (w 2)
          (A1 2) (A1 6) (A1 10) (A1 14) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (x 3) (x 7) (x 11) (x 15) (w 3)
          (A1 3) (A1 7) (A1 11) (A1 15) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (A1 0) (A1 1) (A1 2) (A1 3)
          (w 4) (A 0) (A 4) (A 8) (A 12) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (A1 4) (A1 5) (A1 6) (A1 7)
          (w 5) (A 1) (A 5) (A 9) (A 13) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (A1 8) (A1 9) (A1 10) (A1 11)
          (w 6) (A 2) (A 6) (A 10) (A 14) ∧
        radix_4_dragonfly_fxp (N,N - 1,1) (A1 12) (A1 13) (A1 14) (A1 15)
          (w 7) (A 3) (A 7) (A 11) (A 15)
  
```

ZEON PDF  
www.zeon.com

- Radix-4 16-point decimation-in-frequency dalam level

RTL:

```
radix_4_16_point_DIF_FFT_RTL =
  ⊢def ∀ N xr xi ar ai wr wi.
    radix_4_16_point_DIF_FFT_RTL N xr xi ar ai wr wi =
    ∃ alr ali.
      radix_4_dragonfly_RTL N (xr 0) (xi 0) (xr 4) (xi 4) (xr 8) (xi 8)
        (xr 12) (xi 12) (wr 0) (wi 0) (alr 0) (ali 0) (alr 4) (ali 4)
        (alr 8) (ali 8) (alr 12) (ali 12) ∧
      radix_4_dragonfly_RTL N (xr 1) (xi 1) (xr 5) (xi 5) (xr 9) (xi 9)
        (xr 13) (xi 13) (wr 1) (wi 1) (alr 1) (ali 1) (alr 5) (ali 5)
        (alr 9) (ali 9) (alr 13) (ali 13) ∧
      radix_4_dragonfly_RTL N (xr 2) (xi 2) (xr 6) (xi 6) (xr 10)
        (xi 10) (xr 14) (xi 14) (wr 2) (wi 2) (alr 2) (ali 2) (alr 6)
        (ali 6) (alr 10) (ali 10) (alr 14) (ali 14) ∧
      radix_4_dragonfly_RTL N (xr 3) (xi 3) (xr 7) (xi 7) (xr 11)
        (xi 11) (xr 15) (xi 15) (wr 3) (wi 3) (alr 3) (ali 3) (alr 7)
        (ali 7) (alr 11) (ali 11) (alr 15) (ali 15) ∧
      radix_4_dragonfly_RTL N (alr 0) (ali 0) (alr 1) (ali 1) (alr 2)
        (ali 2) (alr 3) (ali 3) (wr 4) (wi 4) (ar 0) (ai 0) (ar 4)
        (ai 4) (ar 8) (ai 8) (ar 12) (ai 12) ∧
      radix_4_dragonfly_RTL N (alr 4) (ali 4) (alr 5) (ali 5) (alr 6)
        (ali 6) (alr 7) (ali 7) (wr 5) (wi 5) (ar 1) (ai 1) (ar 5)
        (ai 5) (ar 9) (ai 9) (ar 13) (ai 13) ∧
      radix_4_dragonfly_RTL N (alr 8) (ali 8) (alr 9) (ali 9) (alr 10)
        (ali 10) (alr 11) (ali 11) (wr 6) (wi 6) (ar 2) (ai 2) (ar 6)
        (ai 6) (ar 10) (ai 10) (ar 14) (ai 14) ∧
      radix_4_dragonfly_RTL N (alr 12) (ali 12) (alr 13) (ali 13)
        (alr 14) (ali 14) (alr 15) (ali 15) (wr 7) (wi 7) (ar 3) (ai 3)
        (ar 7) (ai 7) (ar 11) (ai 11) (ar 15) (ai 15)
```

- Verifikasi dari RTL ke level fixed-point dari radix-4 16-point decimation-in-frequency FFT:

Lemma 19:

```
∀ N xr xi ar ai wr wi.
  radix_4_16_point_DIF_FFT_RTL N xr xi ar ai wr wi ⇒
  radix_4_16_point_DIF_FFT_fxp N (FXP_VECT_COMPLEX N xr xi)
    (FXP_VECT_COMPLEX N ar ai) (FXP_VECT_COMPLEX_NUM N wr wi)
```

- Algoritma radix-4 16-point decimation-in-frequency dalam domain real ideal:

```
radix_4_16_point_FFT_real_algorithm =
   $\vdash_{def} \forall x w.$ 
  radix_4_16_point_FFT_real_algorithm x w =
  ( $\lambda p.$ 
    complex_sum (0,4)
      ( $\lambda i.$ 
        complex_sum (0,4)
          ( $\lambda j.$  x (4 * j + i) * w (4 * p MOD 4 * j)) * w (p * i)))
```

- Algoritma radix-4 16-point decimation-in-frequency dalam domain floating-point:

```
radix_4_16_point_FFT_floating_point_algorithm =
   $\vdash_{def} \forall x' w'.$ 
  radix_4_16_point_FFT_floating_point_algorithm x' w' =
  ( $\lambda p.$ 
    float_complex_sum (0,4)
      ( $\lambda i.$ 
        float_complex_sum (0,4)
          ( $\lambda j.$  x' (4 * j + i) * w' (4 * p MOD 4 * j)) *
            w' (p * i)))
```

- Algoritma radix-4 16-point decimation-in-frequency dalam domain fixed-point:

```
radix_4_16_point_FFT_fixed_point_algorithm =
   $\vdash_{def} \forall X x'' w''.$ 
  radix_4_16_point_FFT_fixed_point_algorithm X x'' w'' =
  ( $\lambda p.$ 
    fxp_complex_sum (0,4) X
      ( $\lambda i.$ 
        fxp_complex_mul X
          (fxp_complex_sum (0,4) X
            ( $\lambda j.$ 
              fxp_complex_mul X (x'' (4 * j + i))
                (w'' (4 * p MOD 4 * j)))) (w'' (p * i))))
```

- Verifikasi dari perluasan fixed-point ke bentuk penutup algoritma fixed-point:

Lemma 20:

$\forall N \times w \text{ A.}$

$\text{radix\_4\_16\_point\_DIF\_FFT\_fxp } N \times A \text{ } w \implies$

$\forall p.$

$A \text{ } p =$

$\text{radix\_4\_16\_point\_FFT\_fixed\_point\_algorithm } (N, N - 1, 1) \times$

$(w \text{ (} p \text{ DIV } 4)) \text{ } p$

ZEON PDF Driver  
www.zeon.com.tw