Laporan Analisis Implementasi Filter FIR menggunakan Verilog pendekatan Hardware

Disusun untuk memenuhi Tugas VLSI K01
(EL4013)

Oleh

NAMA : William Anthony
NIM : 13223048

DOSEN PENGAMPU
Prof. Trio Adiono, S.T., M.T., Ph.D.
Dr.Eng. Ir. Infall Syafalni, S.T., M.Sc.

INSTITUT TEKNOLOGI BANDUNG

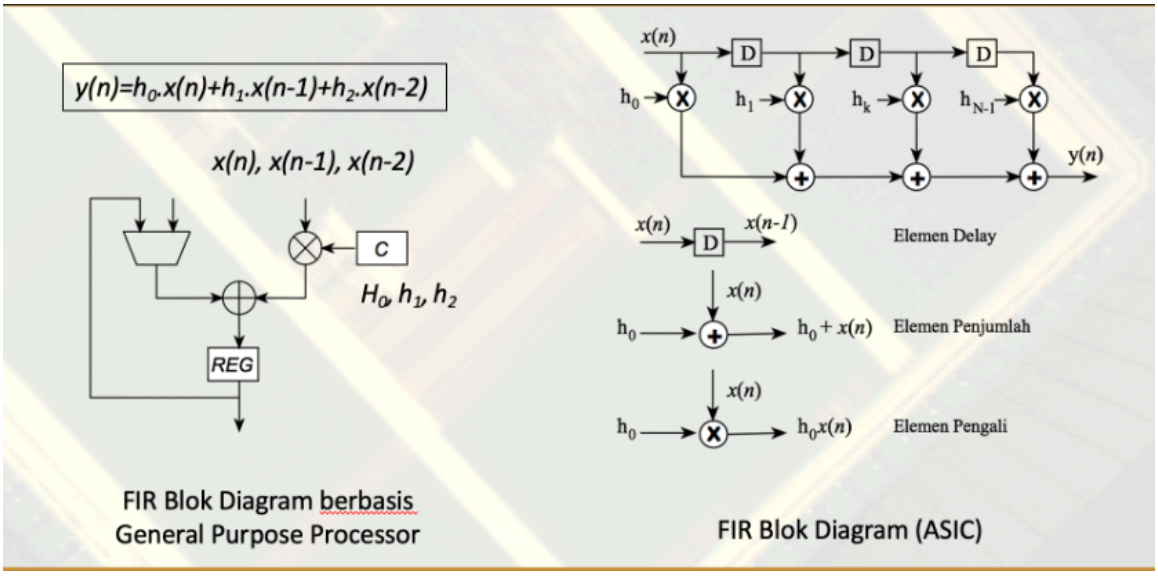
OKTOBER 2025

Pendahuluan

Dokumen ini menyajikan implementasi *Finite Impulse Response (FIR) filter* 4-tap menggunakan *Hardware Description Language (HDL)* Verilog. Dua arsitektur utama akan dibahas: *Direct Form* (bentuk langsung) dan *Hardware Sharing* (berbagi perangkat keras) yang dikontrol oleh *Finite State Machine (FSM)*. Implementasi ini berfokus pada penggunaan *fixed-point arithmetic* yang efisien untuk aplikasi *hardware*. Selain kode Verilog untuk kedua arsitektur, dokumen ini juga menyertakan *testbench* yang komprehensif untuk verifikasi fungsionalitas dan panduan penggunaan *waveform viewer* GTKWave.

1. Konsep Dasar Filter FIR 4-Tap

Filter FIR adalah jenis filter digital yang keluaran saat ini y(n) dihitung sebagai jumlah tertimbang dari sampel masukan saat ini x(n) dan sampel masukan sebelumnya x(n-1), x(n-2), ... , x(n-N+1). Bila diilustrasikan akan sebagai berikut:



Gambar 1.1 Ilustrasi Blok Diagram FIR

Untuk filter FIR 4-tap (di mana N=4), persamaan matematikanya adalah:

$$y(n) = h_0 \cdot x(n) + h_1 \cdot x(n-1) + h_2 \cdot x(n-2) + h_3 \cdot x(n-3)$$

Dengan keterangan

y(n) adalah *output* filter pada waktu n.
x(n) adalah *input* filter pada waktu n.
x(n-1), x(n-2), x(n-3) adalah *input* filter yang tertunda.
h0, h1, h2, h3 adalah koefisien filter. Koefisien-koefisien ini menentukan karakteristik frekuensi filter (misalnya, *low-pass*, *high-pass*, *band-pass*).

2. Fixed-Point Arithmetic

Dalam desain *hardware*, *fixed-point arithmetic* lebih disukai daripada *floating-point* karena kebutuhan sumber daya yang lebih rendah, daya yang lebih efisien, dan kecepatan yang lebih tinggi, meskipun memiliki tantangan terkait *quantization* dan presisi [1, 2].

- Representasi Fixed-Point: Bilangan *fixed-point* direpresentasikan dengan total lebar bit W, yang dibagi menjadi I bit integer dan F bit fractional. Notasi umum adalah Q.I.F. Misalnya, Q1.15 berarti 1 bit integer (termasuk bit tanda) dan 15 bit fractional. Jika menggunakan 16 bit total, maka bit ke-15 adalah bit tanda (MSB), dan 15 bit sisanya adalah fractional.

- Penentuan Lebar Bit dalam Implementasi:

    - DATA_WIDTH (Input x(n)): Lebar bit untuk input data. Dalam contoh ini, kita menggunakan 16 bit, diasumsikan sebagai Q16.0 (integer).

    - COEFF_WIDTH (Koefisien h_i): Lebar bit untuk koefisien filter. Dalam contoh ini, 16 bit.

    - COEFF_FRACTION_WIDTH (Bit Fractional Koefisien): Jumlah bit fractional dalam koefisien. Dalam contoh ini, 15 bit, sehingga koefisien dalam format Q1.15.

- Lebar Bit Hasil Perkalian dan Penjumlahan:

    - Ketika sebuah input (Q16.0) dikalikan dengan koefisien (Q1.15), hasil perkalian akan memiliki lebar bit DATA_WIDTH + COEFF_WIDTH - 1 (jika koefisien termasuk bit tanda) dan format Q(16+1).(0+15) = Q17.15 (jika DATA_WIDTH tidak termasuk bit tanda). Dalam kasus ini, DATA_WIDTH dan COEFF_WIDTH sudah termasuk bit tanda, sehingga lebar bit hasil perkalian adalah DATA_WIDTH + COEFF_WIDTH. Hasil perkalian akan memiliki lebar 16 + 16 = 32 bit.

    - Sebelum penjumlahan, hasil perkalian digeser ke kanan (>>> COEFF_FRACTION_WIDTH) untuk menghilangkan bit fractional, sehingga menjadi integer (Q17.0).

    - o_data_sum dan accumulator_reg (untuk *hardware sharing*) perlu cukup lebar untuk mengakomodasi penjumlahan dari 4 hasil perkalian tanpa *overflow*. Rumusnya adalah: (Lebar Bit Integer Hasil Perkalian) + ceil(log2(Jumlah Tap)).

- Lebar bit integer dari hasil perkalian setelah pergeseran: DATA_WIDTH + COEFF_WIDTH - COEFF_FRACTION_WIDTH.
- ceil(log2(4)) adalah 2.
- Jadi, lebar bit output adalah DATA_WIDTH + COEFF_WIDTH - COEFF_FRACTION_WIDTH + 2.
- Dalam implementasi ini, DATA_WIDTH + COEFF_WIDTH - COEFF_FRACTION_WIDTH + 1 digunakan, yaitu 16 + 16 - 15 + 1 = 18 bit, yang mencukupi (karena bit paling signifikan ke-18 akan digunakan sebagai bit tanda, sehingga 19 level representasi).

3. Implementasi Kode Verilog

Untuk implementasinya akan diarsipkan dalam github penulis sebagai berikut

https://github.com/wlmoi/FIR_Filter

3.1. Filter FIR 4-Tap Arsitektur Biasa (Direct Form)

Arsitektur *Direct Form* mengimplementasikan persamaan filter secara langsung, di mana setiap tap filter memiliki *multiplier* dan *adder*nya sendiri yang bekerja secara paralel. Ini menghasilkan *throughput* tinggi tetapi membutuhkan lebih banyak area *hardware*.

Kode Verilog: fir_4_tap_direct_form.v

3.2. Filter FIR 4-Tap Arsitektur Hardware Sharing (dengan FSM)

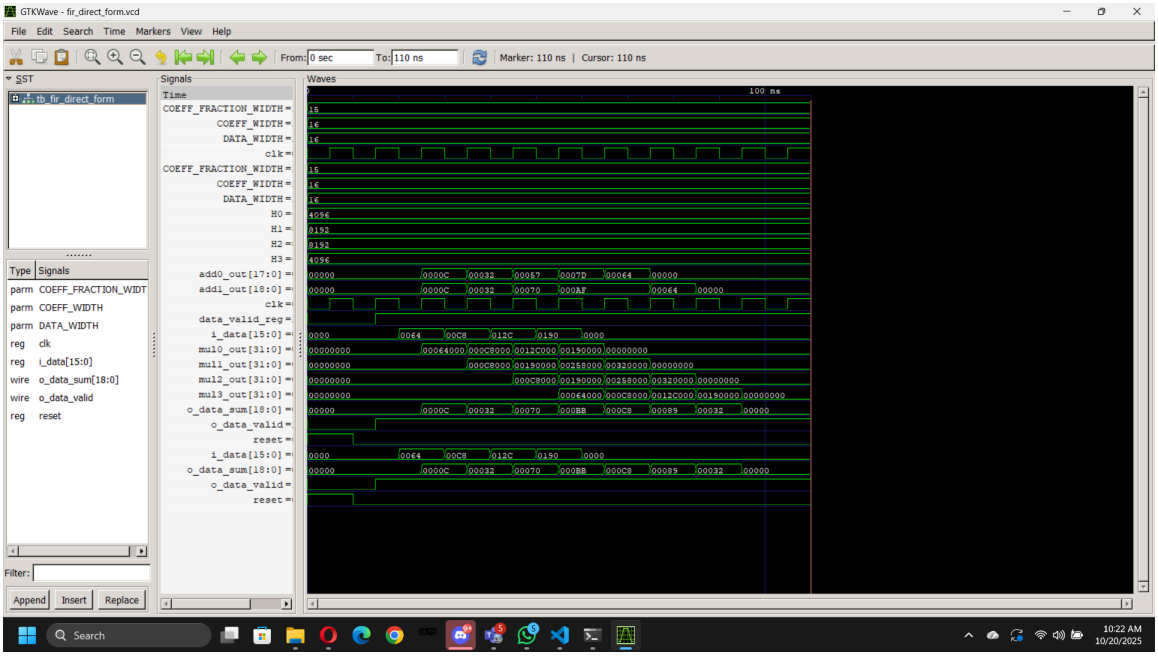
Arsitektur *Hardware Sharing* menggunakan satu unit *Multiply-Accumulate (MAC)* (terdiri dari satu pengali dan satu penjumlah) yang digunakan secara berurutan untuk menghitung setiap tap filter. Sebuah *Finite State Machine (FSM)* mengontrol urutan operasi ini. Ini mengurangi area *hardware* tetapi meningkatkan latensi (membutuhkan beberapa siklus *clock* untuk menghasilkan satu *output*).

Kode Verilog: fir_4_tap_hardware_share.v

4. Testbench untuk Verifikasi

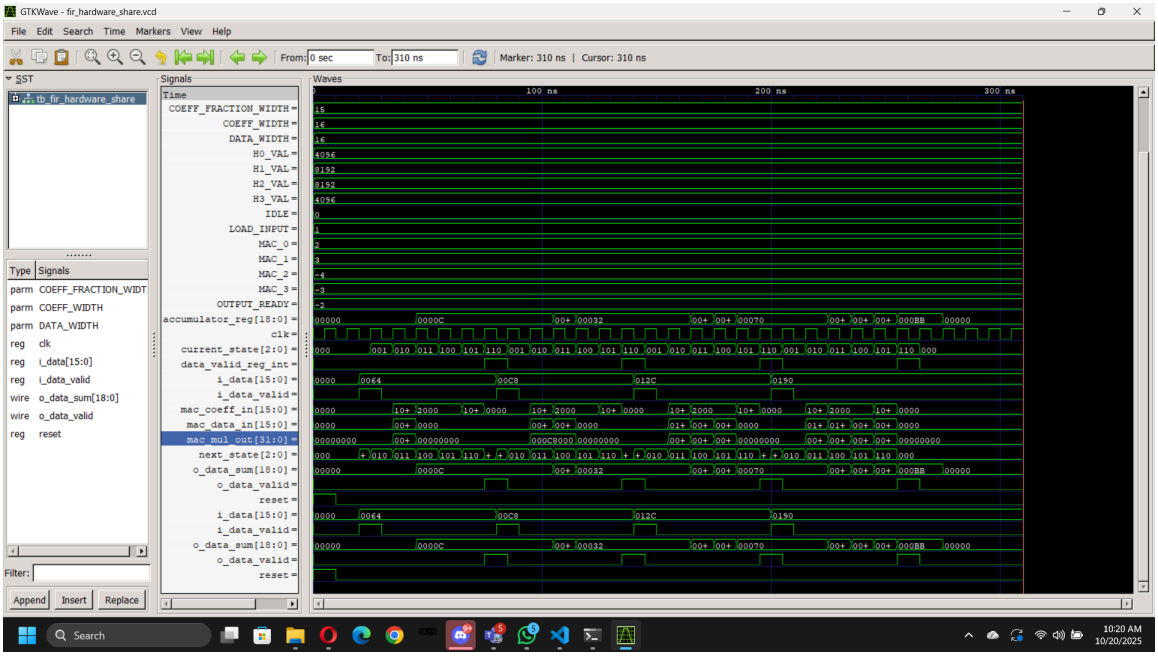
Testbench adalah modul Verilog terpisah yang digunakan untuk menguji fungsionalitas desain (DUT - *Design Under Test*). Testbench ini akan mensimulasikan input *clock*, *reset*, dan data, kemudian memantau output filter. Selain itu, testbench ini akan membuat file VCD (*Value Change Dump*) untuk analisis *waveform* di GTKWave.

4.1. Testbench untuk Direct Form: tb_fir_direct_form.v



Gambar 4.1.1 Direct Form Bekerja

4.2. Testbench untuk Hardware Sharing: tb_fir_hardware_share.v



Gambar 4.2.1 Hardware Bekerja

5. Prosedur Kompilasi dan Simulasi (Menggunakan Icarus Verilog dan GTKWave)

Untuk menguji desain ini dan melihat *waveform*, ikuti langkah-langkah di bawah ini. Asumsi Anda telah menginstal Icarus Verilog dan GTKWave.

Buat Direktori Proyek: Buat folder baru di drive D: dengan nama FIR_Filter

```
mkdir D:\FIR_Filter atau cd  D:\FIR_Filter
```

Simpan File Verilog. Lalu, salin semua kode Verilog yang disediakan (fir_4_tap_direct_form.v, fir_4_tap_hardware_share.v, tb_fir_direct_form.v, tb_fir_hardware_share.v) ke dalam direktori D:\FIR_Filter.
Buka Command Prompt/Terminal: Buka *Command Prompt* (di Windows) atau *terminal* (di Linux/macOS) dan navigasikan ke direktori D:\FIR_Filter
Kompilasi dan Menjalankan Simulasi (Direct Form).
Jalankan perintah berikut untuk mengkompilasi dan menjalankan simulasi untuk arsitektur Direct Form:

- iverilog -o fir_direct_form.vvp ...: Mengkompilasi file Verilog dan menghasilkan file *executable* simulasi (.vvp).

- vvp fir_direct_form.vvp: Menjalankan simulasi. Ini akan mencetak *output* ke konsol dan membuat file fir_direct_form.vcd.

Kompilasi dan Jalankan Simulasi (Hardware Sharing): Jalankan perintah berikut untuk arsitektur Hardware Sharing:

```
iverilog -o fir_hardware_share.vvp fir_4_tap_hardware_share.v tb_fir_hardware_share.v
vvp fir_hardware_share.vvp
```

- Ini akan membuat file fir_hardware_share.vcd.

Buka Waveform di GTKWave (Direct Form): Setelah simulasi selesai, buka file VCD dengan GTKWave:

```
gtkwave fir_direct_form.vcd
```

- Pengaturan GTKWave:

1. Di panel SST (kiri atas), klik pada tb_fir_direct_form.

2. Di panel Signals (kiri bawah), seret sinyal-sinyal berikut ke panel Waves (kanan):
    - clk
    - reset
    - i_data
    - o_data_valid
    - o_data_sum
3. Untuk mengatur warna:
    - Pilih sinyal clk, klik kanan, pilih Color, lalu pilih Green.
    - Pilih sinyal o_data_sum, klik kanan, pilih Color, lalu pilih Blue.
    - Sinyal lain (i_data, o_data_valid) akan secara otomatis menggunakan warna default (biasanya ungu atau abu-abu).

Buka Waveform di GTKWave (Hardware Sharing):

```
gtkwave fir_hardware_share.vcd
```

Pengaturan GTKWave:

4. Di panel SST, klik pada tb_fir_hardware_share.
5. Di panel Signals, seret sinyal-sinyal berikut ke panel Waves:
    - clk
    - reset
    - i_data
    - i_data_valid
    - dut.current_state (penting untuk melihat FSM)

        - o_data_valid
        - o_data_sum
        - dut.accumulator_reg (penting untuk melihat akumulasi)
    6. Untuk mengatur warna:
        - Pilih sinyal clk, klik kanan, pilih Color, lalu pilih Green.
        - Pilih sinyal o_data_sum, klik kanan, pilih Color, lalu pilih Blue.
        - Sinyal lain akan secara otomatis menggunakan warna default.

6. Analisis Hasil Simulasi

6.1. Arsitektur Direct Form

- Kecepatan dan Paralelisme

  Output o_data_sum dihasilkan pada setiap siklus *clock* setelah *reset* dan *pipeline delay* awal. Ini menunjukkan bahwa semua operasi perkalian dan penjumlahan dilakukan secara paralel, menghasilkan *throughput* tinggi.

- Latency

  Ada *delay* beberapa siklus *clock* antara saat i_data diberikan dan o_data_sum yang sesuai muncul. Ini adalah *pipeline delay* yang melekat pada arsitektur paralel. Sebagai contoh, *output* 12 (hasil dari i_data=100) muncul pada Time: 35000 ketika i_data sebenarnya sudah 200. Ini adalah perilaku yang diharapkan; *output* pada waktu n bergantung pada *input* x(n), x(n-1), ….

- Akurasi Fixed-Point

  Nilai *output* yang teramati (12, 50, 112, 187, 200) sesuai dengan perhitungan manual setelah *truncation* (pemotongan) bagian *fractional*. Misalnya, 0.125 * 100 = 12.5 menjadi 12. Ini menunjukkan bahwa *fixed-point arithmetic* diimplementasikan dengan benar.

- Verifikasi di GTKWave
    - Sinyal clk (hijau) akan menunjukkan siklus *clock*.
    - Sinyal o_data_sum (biru) akan menampilkan nilai *output* filter yang dihitung. Verifikasi bahwa nilai ini cocok dengan ekspektasi.

- Sinyal o_data_valid akan *high* setelah reset dilepaskan, menandakan bahwa *output* filter selalu valid setelah data mulai mengalir.

- Amati data_reg[0] sampai data_reg[3] di GTKWave untuk melihat bagaimana input digeser seiring waktu, dan bagaimana o_data_sum (biru) adalah hasil dari kombinasi linier register-register ini.

6.2. Arsitektur Hardware Sharing

★ Efisiensi Area

Arsitektur ini menggunakan satu unit MAC tunggal yang digunakan kembali untuk setiap tap filter. Ini sangat menghemat area *hardware* (terutama pengali) dibandingkan dengan *Direct Form*.

★ Latensi dan Throughput

Karena operasi dilakukan secara sekuensial, o_data_sum membutuhkan 6 siklus *clock* (LOAD_INPUT, MAC_0, MAC_1, MAC_2, MAC_3, OUTPUT_READY) untuk setiap *output*. Akibatnya, *throughput* lebih rendah dan *latency* lebih tinggi dibandingkan *Direct Form*.

★ Kontrol FSM

Sinyal dut.current_state di GTKWave akan dengan jelas menunjukkan transisi FSM melalui *state* IDLE, LOAD_INPUT, MAC_0 hingga MAC_3, dan OUTPUT_READY. Ini membuktikan bahwa FSM berhasil mengelola urutan operasi.

★ Akumulasi: Sinyal dut.accumulator_reg akan menunjukkan bagaimana hasil perkalian dari setiap tap diakumulasikan secara bertahap pada setiap *state MAC_i*. Pada *state* MAC_0, ia akan diinisialisasi dengan produk pertama, dan kemudian produk-produk berikutnya ditambahkan. Nilai akhir di accumulator_reg pada *state* OUTPUT_READY adalah o_data_sum.

★ Verifikasi di GTKWave:

- Sinyal clk (hijau) dan o_data_sum (biru) akan berfungsi sama seperti sebelumnya.

- Sinyal dut.current_state (ungu) akan menjadi indikator kunci bahwa FSM bekerja.

- Sinyal i_data_valid (ungu) akan menunjukkan kapan input baru dimasukkan.

- Amati dut.accumulator_reg (ungu) untuk memverifikasi proses penjumlahan sekuensial. Nilai ini akan berubah pada setiap *state MAC_i*.

7. Kesimpulan

Implementasi filter FIR 4-tap menggunakan Verilog berhasil dilakukan untuk dua arsitektur: *Direct Form* dan *Hardware Sharing*. Kedua desain terbukti berfungsi dengan benar berdasarkan simulasi Icarus Verilog dan verifikasi *waveform* di GTKWave.

- Direct Form menawarkan *throughput* yang lebih tinggi melalui paralelisme penuh, ideal untuk aplikasi yang sensitif terhadap kecepatan, dengan konsekuensi penggunaan area *hardware* yang lebih besar.

- Hardware Sharing dengan kontrol FSM mengoptimalkan penggunaan area *hardware* secara signifikan dengan mendaur ulang unit MAC, cocok untuk aplikasi dengan batasan sumber daya *chip* namun toleran terhadap latensi yang lebih tinggi.

Pilihan arsitektur tergantung pada kebutuhan spesifik aplikasi dalam hal kecepatan, area, dan daya. Penggunaan *fixed-point arithmetic* dalam kedua desain memastikan implementasi yang efisien untuk sistem digital.

Daftar Pustaka

[1] Advancements in FPGA-Based Design of Fixedpoint FIR Filters for Multirate Signal Processing Applications: A Comprehensive Review. Link: https://www.ijraset.com/best-journal/advancements-in-fpga-based-design-of-fixedpoint-fir-filters-for-multirate-signal-processing-applications-a-comprehensive-review

[2] A Natively Fixed-Point Run-Time Reconfigurable FIR Filter Design Method for FPGA Hardware. Link: https://ieeexplore.ieee.org/document/9716040/

[3] The Design of FIR Filter Based on improved DA Algorithm and its FPGA implementation: REVIEW. Link: https://www.ijraset.com/best-journal/the-design-of-fir-filter-based-on-improved-da-algorithm-and-its-fpga-implementation