

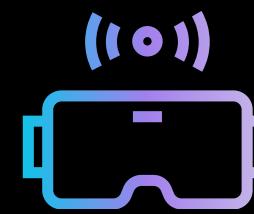
DSP FIR LOWPASS FILTER

page 01

Ahmad Fariz Khairi (2306211370)
Christover Angelo Lasut (2306220343)
Muhammad Raihan Mustofa (2306161946)
Ryan Adidaru Excel Barnabi 2306266994()

2025



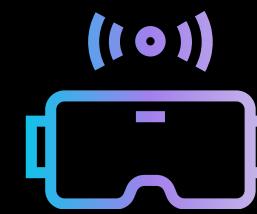


Latar Belakang



Dalam dunia modern yang semakin berkembang, teknologi pemrosesan sinyal digital atau Digital Signal Processing (DSP) menjadi salah satu fondasi penting dalam berbagai aplikasi, seperti komunikasi, audio, radar, dan sistem kendali. DSP memungkinkan manipulasi sinyal secara efisien untuk meningkatkan kualitas informasi yang disampaikan. Salah satu komponen utama dalam DSP adalah filter digital, yang berfungsi untuk memisahkan atau menghilangkan komponen sinyal yang tidak diinginkan.

Filter digital terbagi menjadi dua kategori utama, yaitu Infinite Impulse Response (IIR) dan Finite Impulse Response (FIR). Filter FIR, yang menjadi fokus dari proyek ini, memiliki sejumlah keunggulan, seperti stabilitas yang terjamin, respons fase linear, dan kemudahan implementasi menggunakan algoritma tertentu. FIR Lowpass Filter digunakan untuk menyaring sinyal frekuensi tinggi, sehingga hanya sinyal frekuensi rendah yang diizinkan melewati filter.



Latar Belakang

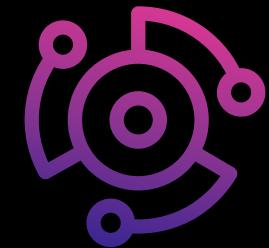


Dalam proyek ini, bahasa deskripsi perangkat keras VHDL (Very High-Speed Integrated Circuit Hardware Description Language) digunakan untuk mendesain FIR Lowpass Filter secara digital. Penggunaan VHDL memungkinkan pengembangan desain yang modular dan fleksibel, sehingga memudahkan proses implementasi pada perangkat keras seperti FPGA (Field Programmable Gate Array).

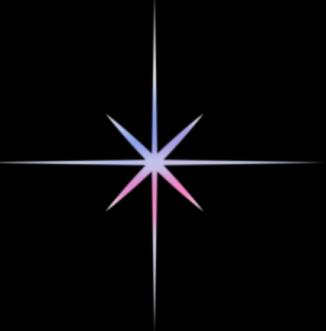
Penerapan FIR Lowpass Filter ini tidak hanya memberikan pengalaman teknis dalam mendesain dan mensimulasikan filter, tetapi juga memperkenalkan mahasiswa pada teknik pemrograman berbasis perangkat keras. Dengan pendekatan ini, proyek diharapkan dapat menjadi pembelajaran yang bermanfaat dan relevan bagi pengembangan keahlian di bidang pemrosesan sinyal.

Deskripsi PROYEK

Proyek kami bertujuan untuk merancang dan mengimplementasikan filter lowpass digital menggunakan VHDL, dengan fokus pada pembuatan solusi pemrosesan sinyal yang andal dan efisien yang menunjukkan penerapan praktis prinsip-prinsip desain sistem digital. Filter lowpass ini dirancang untuk meredam sinyal frekuensi tinggi sambil memungkinkan sinyal frekuensi rendah untuk melewati, dengan perhatian khusus pada parameter seperti frekuensi cutoff, karakteristik peredaman, dan efisiensi komputasi. Kami akan mengeksplorasi berbagai arsitektur filter, termasuk pendekatan finite impulse response (FIR) dan infinite impulse response (IIR), untuk menentukan implementasi yang paling sesuai dengan kebutuhan spesifik kami.



Tujuan



1. Merancang FIR Lowpass Filter berbasis VHDL yang dapat digunakan untuk menyaring sinyal digital.
2. Mengembangkan pemahaman tentang desain filter digital, mulai dari teori hingga implementasi praktis.
3. Mengaplikasikan konsep delay line dan operasi aritmatika digital untuk menghasilkan keluaran filter yang presisi.
4. Melakukan validasi desain menggunakan testbench untuk memastikan filter bekerja sesuai dengan spesifikasi yang diberikan.
5. Menyediakan solusi berbasis perangkat keras untuk aplikasi penyaringan sinyal digital dalam berbagai bidang.



Alat

Visual Studio
Code

ModelSim

Github

MatLab

Implementasi Program



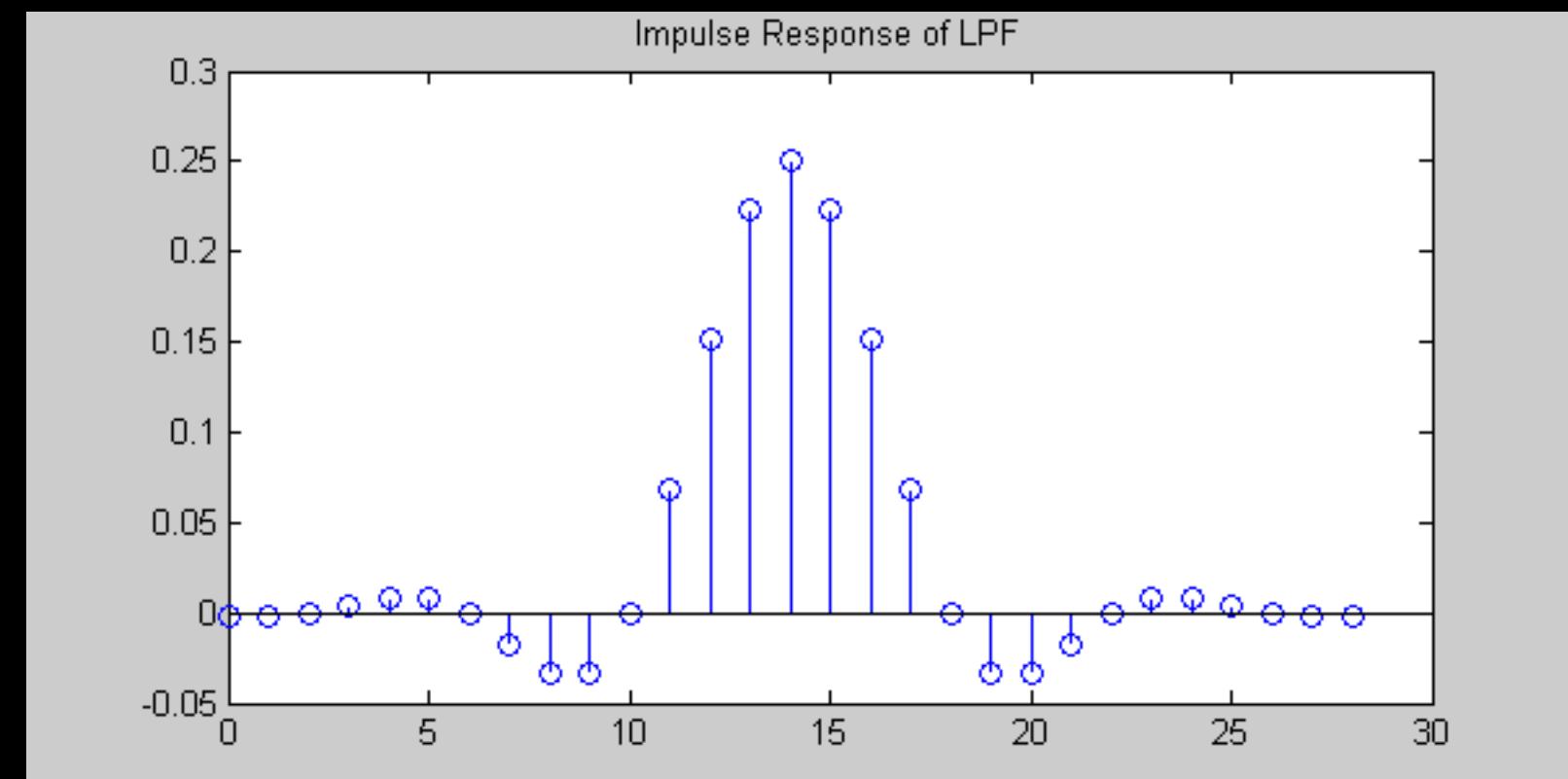
Kutipan Kode

FIR (Finite Impulse Response) Lowpass Filter merupakan filter digital yang dirancang untuk memproses sinyal dengan kemampuan meloloskan komponen frekuensi rendah dan meredam komponen frekuensi tinggi. Filter ini memiliki karakteristik unik dengan respons impuls terbatas, yang berarti outputnya bergantung pada input saat ini dan sejumlah input sebelumnya tanpa melibatkan feedback dari output sebelumnya. Prinsip kerjanya yang fundamental adalah mengalikan setiap sampel input (baik saat ini maupun sebelumnya) dengan serangkaian koefisien filter yang telah ditentukan, kemudian menjumlahkan seluruh hasil perkalian untuk menghasilkan satu sampel output yang representatif.

Dalam implementasinya, filter FIR menggunakan struktur delay line untuk menyimpan sampel-sampel input sebelumnya, dikombinasikan dengan serangkaian koefisien (tap) yang secara presisi menentukan karakteristik filter seperti frekuensi cutoff dan ripple. Desain spesifik filter ini menggunakan arsitektur Direct Form dengan 9 koefisien, dirancang untuk memfilter sinyal di atas 6 kHz dengan teknik scaling 2^{17} untuk meningkatkan presisi perhitungan fixed-point. Proses filtering dilakukan secara sekuensial, dipicu oleh clock, dengan perhitungan internal menggunakan lebar bit 35-36 bit untuk mencegah overflow, memastikan akurasi dan stabilitas pemrosesan sinyal digital yang optimal.

Kutipan Kode

```
-- Coefficients and delay line
type coeff_array is array(0 to 8) of signed(17 downto 0);
constant COEFFS : coeff_array := (
    to_signed(integer(-0.0024 * 2**17), 18),
    to_signed(integer(0.0073 * 2**17), 18),
    to_signed(integer(0.0606 * 2**17), 18),
    to_signed(integer(0.1691 * 2**17), 18),
    to_signed(integer(0.2654 * 2**17), 18),
    to_signed(integer(0.1691 * 2**17), 18),
    to_signed(integer(0.0606 * 2**17), 18),
    to_signed(integer(0.0073 * 2**17), 18),
    to_signed(integer(-0.0024 * 2**17), 18)
);
```



```
/MATLAB Drive/untitled.m
1           h = fir1(9, 6/24);
2           fliplr(h)

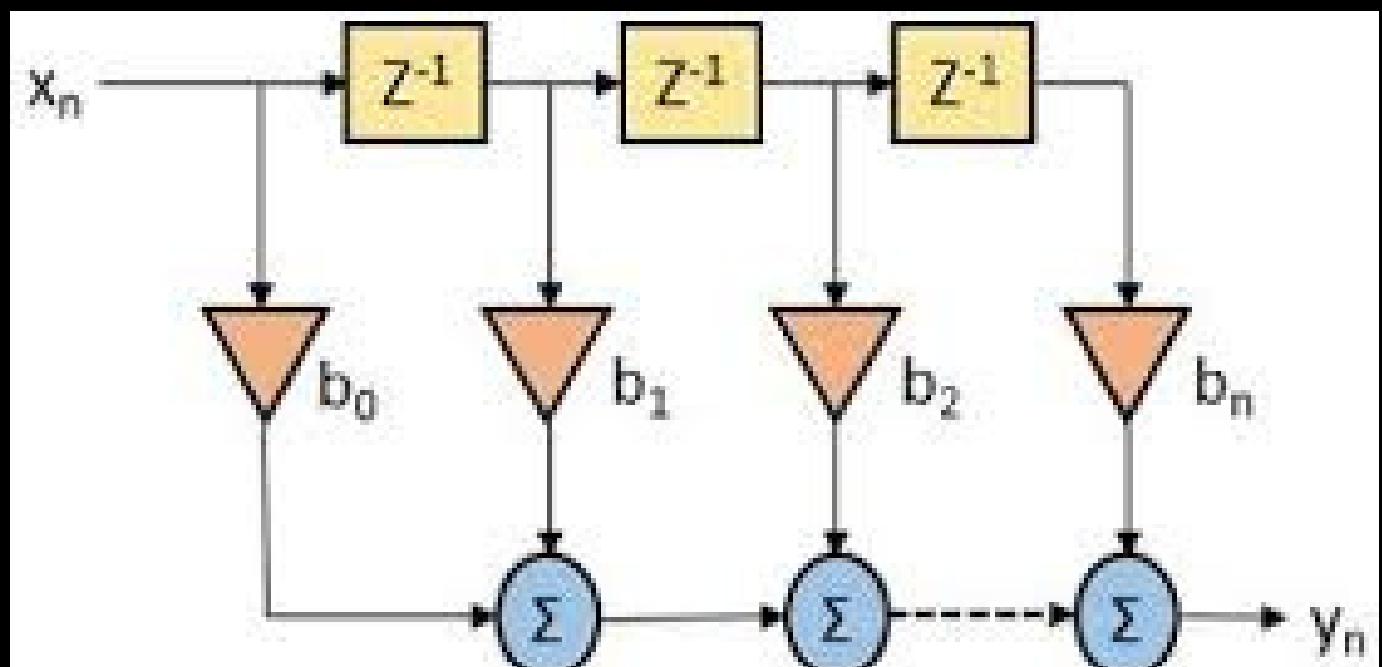
ans =
-0.0024    0.0073    0.0606    0.1691    0.2654    0.2654    0.1691    0.0606    0.0073   -0.0024

>>
```

Kutipan Kode

```
when compute =>
    if mac_count = 0 then
        -- First multiplication
        temp_mult := delay_line(0) * COEFFS(0);
        acc <= temp_mult;
        mac_count <= mac_count + 1;
    elsif mac_count < 8 then
        -- Subsequent multiply-accumulate
        temp_mult := delay_line(mac_count) * COEFFS(mac_count);
        acc <= acc + temp_mult;
        mac_count <= mac_count + 1;
    else
        -- Final multiply-accumulate
        temp_mult := delay_line(8) * COEFFS(8);
        acc <= acc + temp_mult;
        current_state <= output_result;
    end if;
```

$$y[n] = \sum_{k=0}^M b_k x[n-k]$$



FIR Filter

Circuit Globe

Kutipan Kode

```
-- Generate test signals
for i in 0 to 2000 loop
    -- Calculate time
    t := real(i) / SAMPLE_RATE;

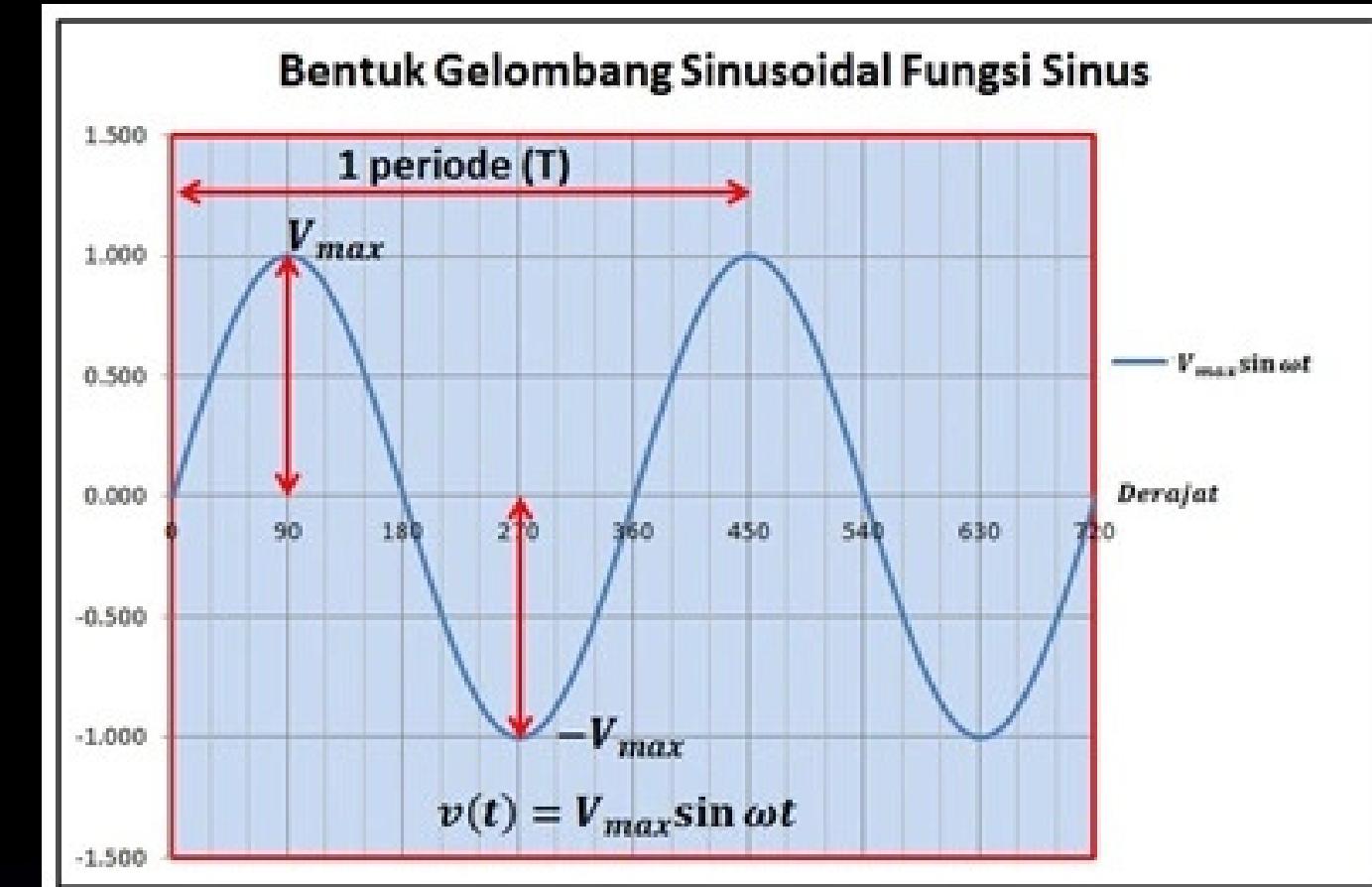
    -- Create input signal: 1 kHz + 15 kHz sine waves
    input_signal <= std_logic_vector(to_signed(
        integer(
            low_freq_amp * sin(2.0 * MATH_PI * 1000.0 * t) + -- 1 kHz sine wave
            high_freq_amp * sin(2.0 * MATH_PI * 15000.0 * t) -- 15 kHz sine wave
        ),
        16
    ));

    -- Generate individual sine waves for analysis
    sine_1kHz <= std_logic_vector(to_signed(
        integer(low_freq_amp * sin(2.0 * MATH_PI * 1000.0 * t)),
        16
    ));

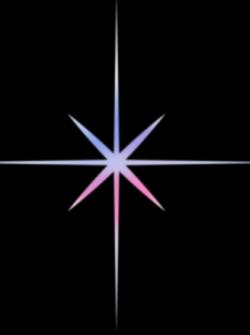
    sine_15kHz <= std_logic_vector(to_signed(
        integer(high_freq_amp * sin(2.0 * MATH_PI * 15000.0 * t)),
        16
    ));

    -- Wait for next clock cycle
    wait for CLK_PERIOD;
end loop;
```

```
stimulus: process
variable low_freq_amp : real := 32767.0 * 0.8; -- 1 kHz sine wave amplitude
variable high_freq_amp : real := 32767.0 * 0.2; -- 15 kHz sine wave amplitude
variable t : real := 0.0;
begin
```



Kutipan Kode



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

package state_mapping_pkg is
    -- State enumeration type
    type state_type is (idle, load, compute, output_result);

    -- Function to convert state to string
    function get_state_name(state : state_type) return string;
end package state_mapping_pkg;

package body state_mapping_pkg is
    -- Implementation of the state-to-string function
    function get_state_name(state : state_type) return string is
    begin
        case state is
            when idle      => return "idle";
            when load      => return "load";
            when compute   => return "compute";
            when output_result => return "output_result";
            when others    => return "unknown"; -- Optional for safety
        end case;
    end function;
end package body state_mapping_pkg;
```

```
type state_type is (idle, load, compute, output_result);
signal current_state : state_type := idle;
```

```
when idle =>
    acc <= (others => '0');
    mac_count <= 0;
    current_state <= load;

when load =>
    -- Shift delay line
    for i in 8 downto 1 loop
        delay_line(i) <= delay_line(i-1);
    end loop;
    delay_line(0) <= resize(signed(input_signal), 18);
    current_state <= compute;
```

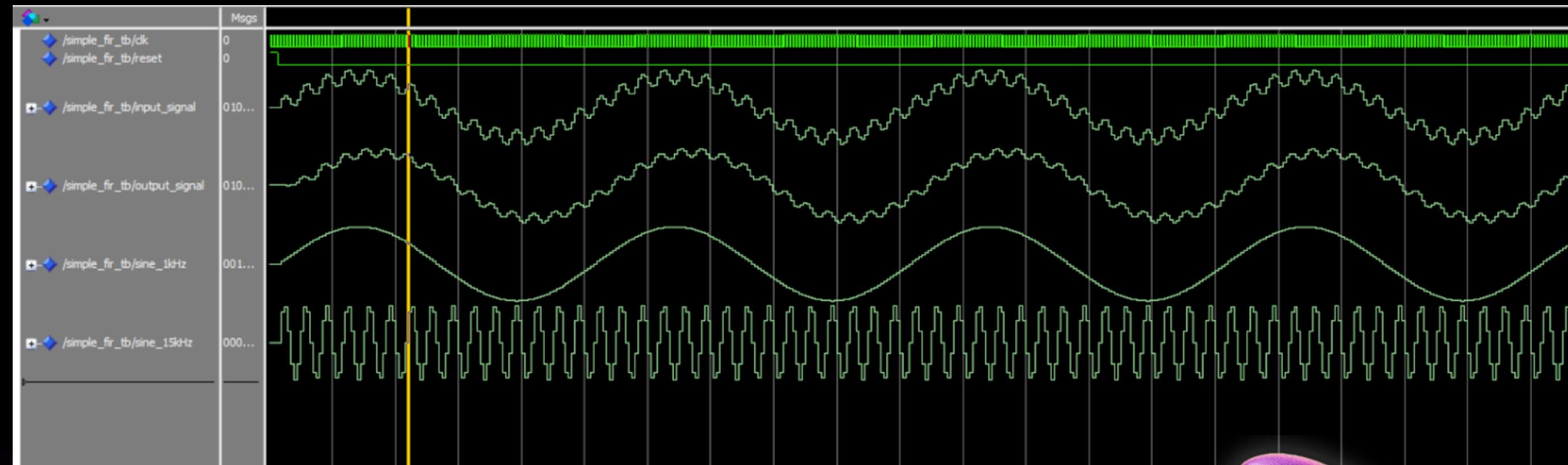
```
-- Monitoring FSM state and output signals
monitor: process
begin
    wait for CLK_PERIOD;
    report "Simulation started";
    while true loop
        wait for CLK_PERIOD;

        -- Report FSM state and output signal
        report "FSM State: " & get_state_name(current_state);
        report "Output Signal: " & integer'image(to_integer(signed(output_signal)));
    end loop;
end process;
```

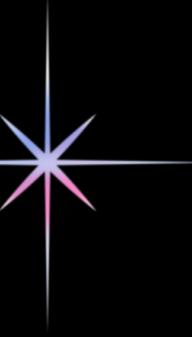
Percobaan dan Analisis



Hasil Percobaan



Percobaan



Grafik di slide berikutnya menunjukkan bagaimana filter lowpass mampu meredam sinyal frekuensi tinggi (15 kHz) sementara mempertahankan sinyal frekuensi rendah (1 kHz). Hal ini terlihat dari penurunan amplitudo pada sinyal 15 kHz di keluaran filter, sementara sinyal 1 kHz masih dipertahankan. Ini menunjukkan bahwa filter lowpass dapat bekerja sesuai dengan rancangan, yaitu melewatkannya frekuensi rendah dan meredam frekuensi tinggi.

Gambar tersebut menunjukkan sinyal input yang terdiri dari kombinasi dua gelombang sine, yaitu gelombang 1 kHz dengan amplitudo 80% dan gelombang 15 kHz dengan amplitudo 20%. Sinyal ini digunakan untuk menguji kinerja filter lowpass yang dirancang.



Analisis



Proyek ini bertujuan untuk mengimplementasikan filter FIR sederhana menggunakan VHDL, di mana pengolahan sinyal dilakukan melalui finite state machine (FSM). Filter dirancang dengan empat state utama: idle, load, compute, dan output_result. Dalam simulasi, ditemukan bahwa FSM selalu berada di state idle, yang menunjukkan adanya kesalahan logika atau implementasi pada desain sistem. Hal ini menjadi fokus utama untuk analisis lebih lanjut guna menemukan penyebab dan memberikan solusi perbaikan.

Pada implementasi FSM, state idle berfungsi sebagai titik awal sebelum sistem melanjutkan proses ke state berikutnya. Namun, selama simulasi, FSM tidak bertransisi ke state load seperti yang diharapkan. Salah satu penyebab potensial adalah sinyal reset yang tidak dilepas dengan benar. Jika sinyal reset tetap aktif, sistem akan terus kembali ke state idle pada setiap siklus clock. Selain itu, mekanisme transisi dalam case current_state is harus diperiksa lebih lanjut. Kemungkinan besar, kondisi yang mengontrol transisi dari idle ke load tidak terpenuhi atau tidak diimplementasikan secara benar.

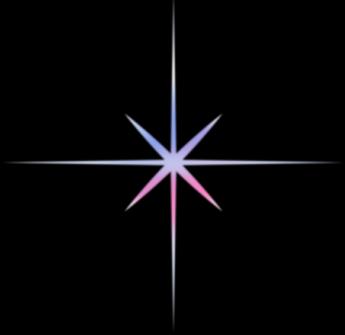
Filter ini menggunakan struktur delay line dan operasi multiply-accumulate (MAC) untuk mengolah sinyal input. Struktur delay line bertugas menyimpan sampel sebelumnya yang digunakan dalam perhitungan MAC. Simulasi mengindikasikan bahwa operasi MAC mungkin tidak berjalan dengan benar, terutama pada variabel mac_count yang digunakan untuk mengontrol iterasi pengolahan koefisien. Jika variabel ini tidak diinisialisasi atau diperbarui dengan benar, maka akumulasi hasil perhitungan tidak akan berjalan sesuai ekspektasi, sehingga FSM tidak dapat melanjutkan ke state compute atau output_result.



Kesimpulan

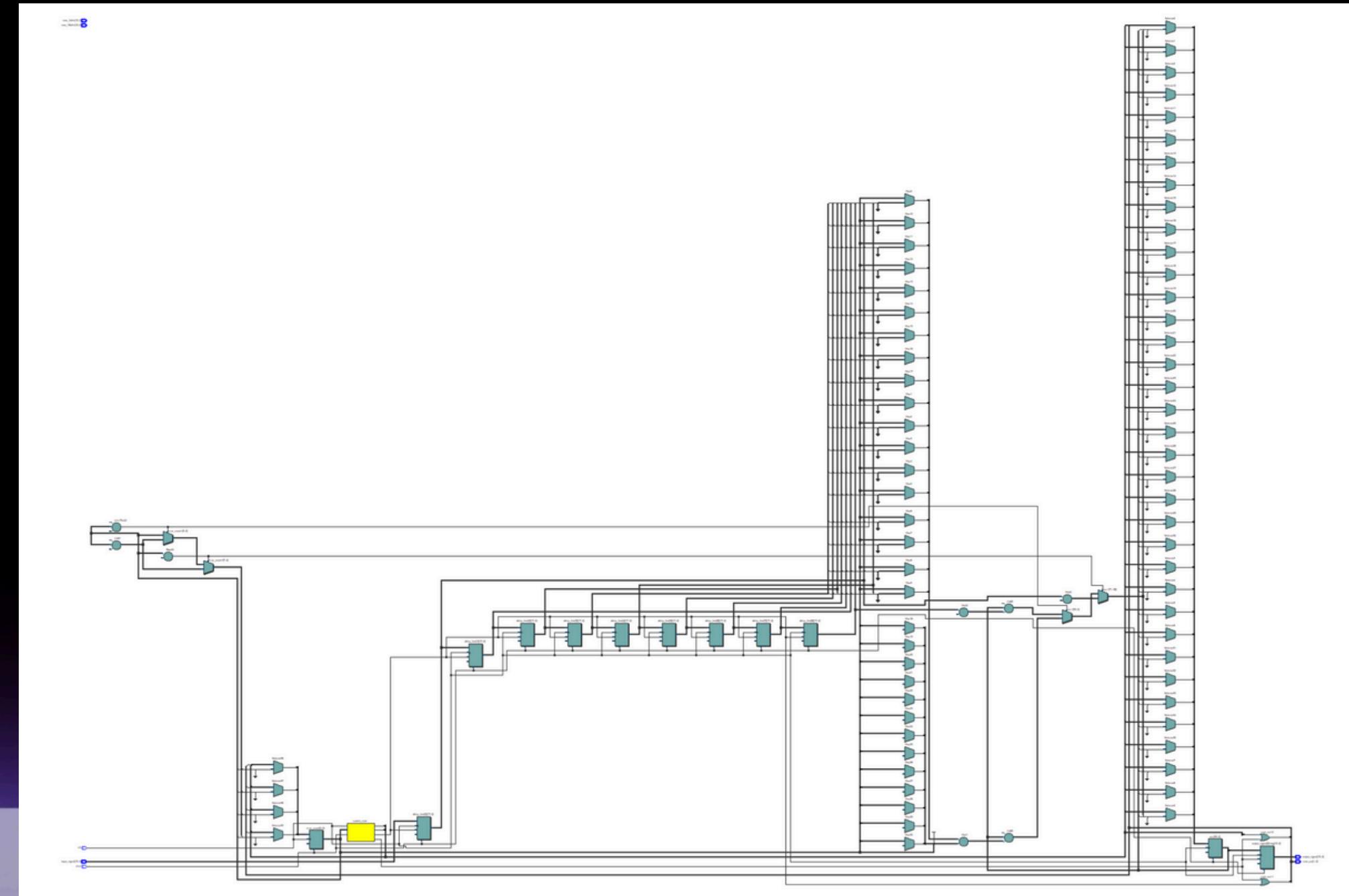
Implementasi FIR Lowpass Filter berbasis VHDL berhasil meredam sinyal frekuensi tinggi (15 kHz) dan mempertahankan sinyal frekuensi rendah (1 kHz), menunjukkan kemampuan filter frekuensi yang baik. Namun, hal-hal seperti logika transisi state pada FSM, mekanisme reset, dan operasi Multiply-Accumulate (MAC) memerlukan optimasi lebih lanjut untuk meningkatkan kinerja filter.

Referensi

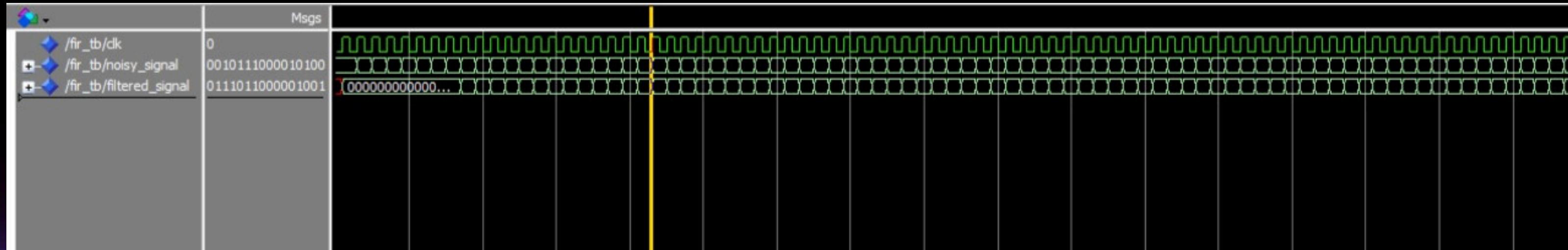
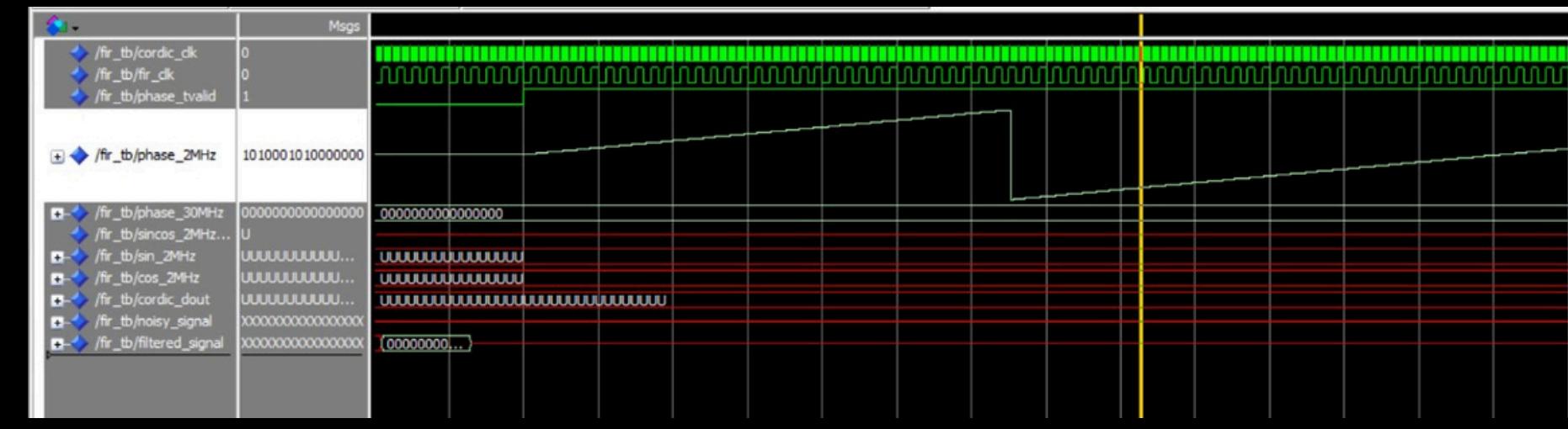
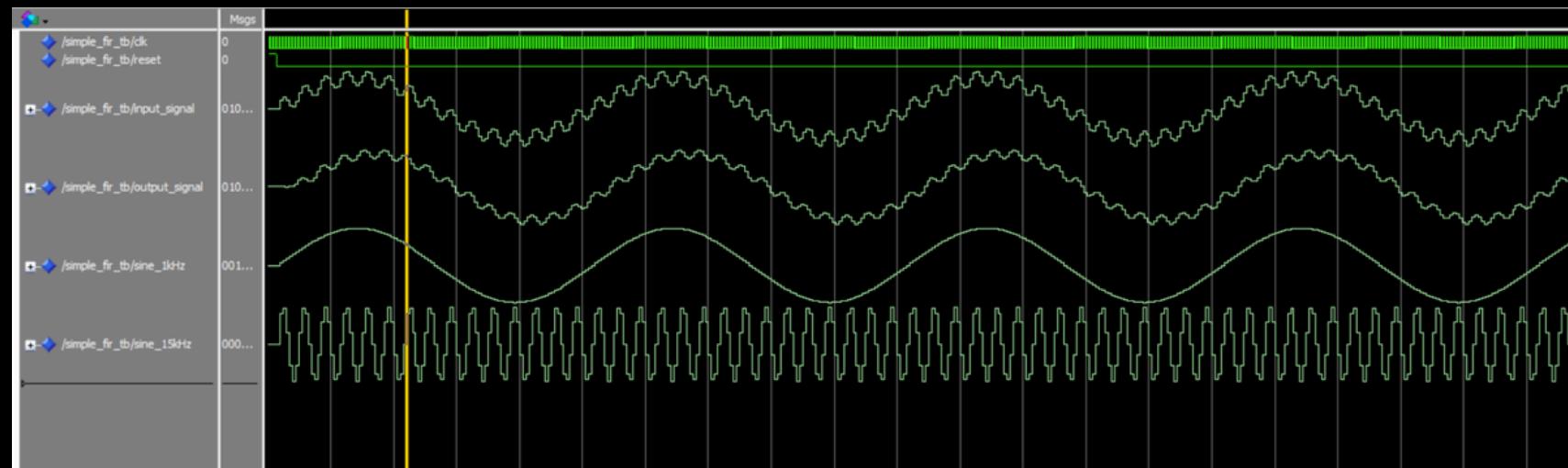


- Circuit Globe, “Difference Between FIR Filter and IIR Filter,” Circuit Globe, Mar. 24, 2020. <https://circuitglobe.com/difference-between-fir-filter-and-iir-filter.html> (accessed Dec. 08, 2024).
- “FIR Lowpass Filter Example,” Github.io, 2024. https://arm-software.github.io/CMSIS_5/DSP/html/group__FIRLPF.html (accessed Dec. 08, 2024).
- “Memisahkan Sinyal Yang Tercampur dengan FIR Filter,” Mon Honneur Est Nommée Fidèle, Feb. 27, 2012. <https://stahlvormund.wordpress.com/2012/02/27/memisahkan-sinyal-yang-tercampur-dengan-fir-filter/> (accessed Dec. 08, 2024).
- “FIR Low-Pass Filter Component,” Qsc.com, 2024. https://qsyshelp.qsc.com/Content/Schematic_Library/filter_lowpass_fir.htm?TocPath=%7CSystem%20Link%7CUSB%7CUSB%20Input%7CFIR%20Low-pass%C2%A0Filter%7C___0 (accessed Dec. 08, 2024).
- Surf-VHDL, “How to Implement FIR Filter in VHDL,” Surf-VHDL, Nov. 14, 2015. <https://surf-vhdl.com/how-to-implement-fir-filter-in-vhdl/> (accessed Dec. 08, 2024).
- D. Marinov, “Part 2: Finite impulse response (FIR) filters,” VHDLwhiz, Feb. 10, 2022. <https://vhdlwhiz.com/part-2-finite-impulse-response-fir-filters/> (accessed Dec. 08, 2024).

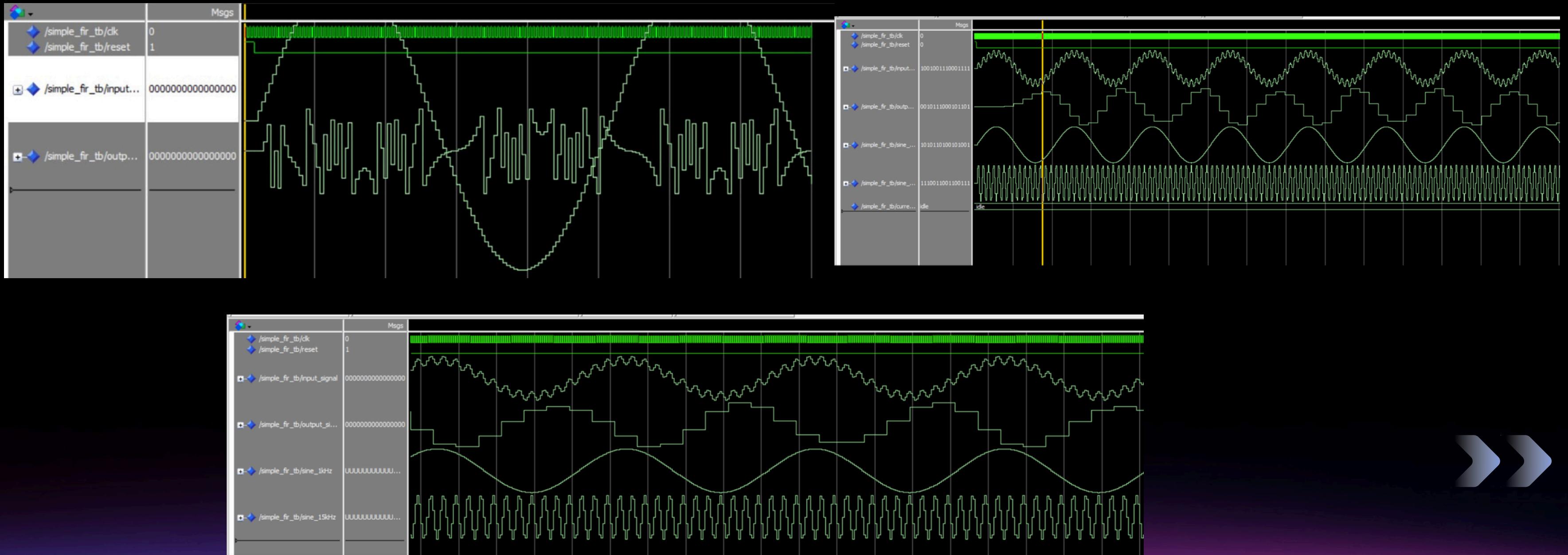
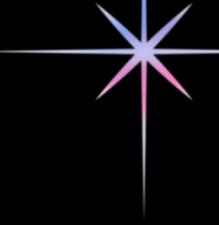
Appendix A : Schematic



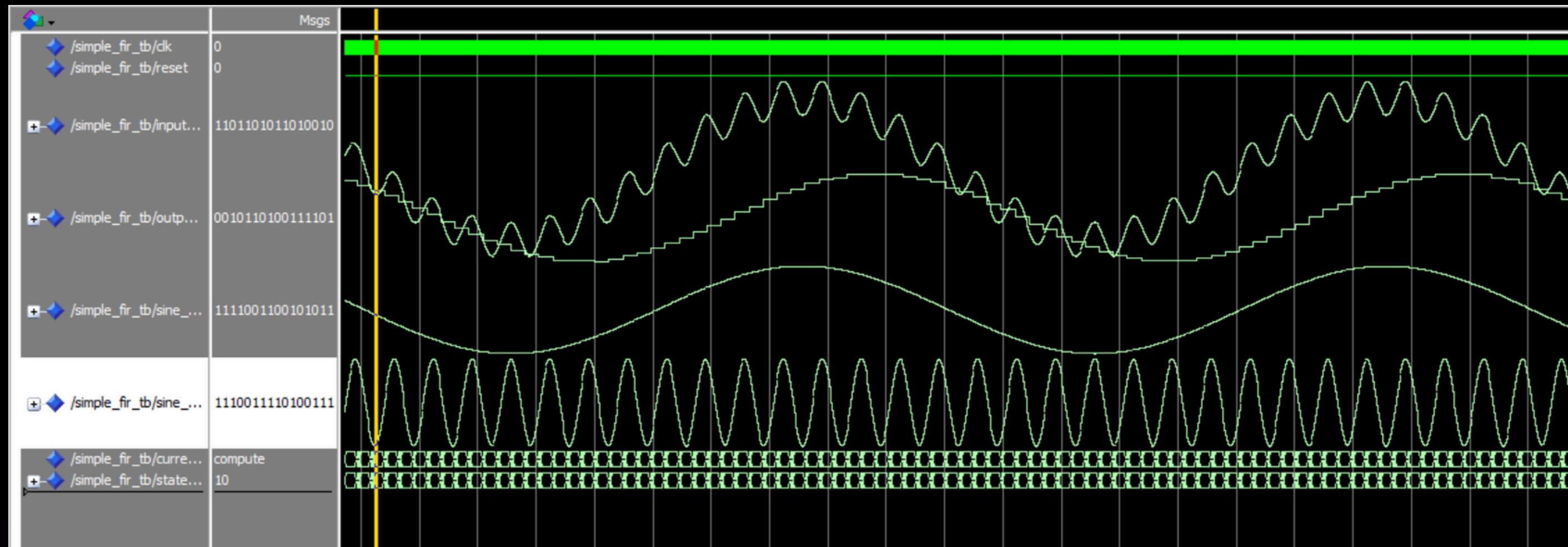
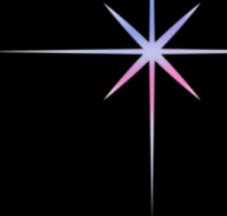
Appendix B: Documentation



Appendix B: Documentation



Appendix B: Documentation



THANK YOU!