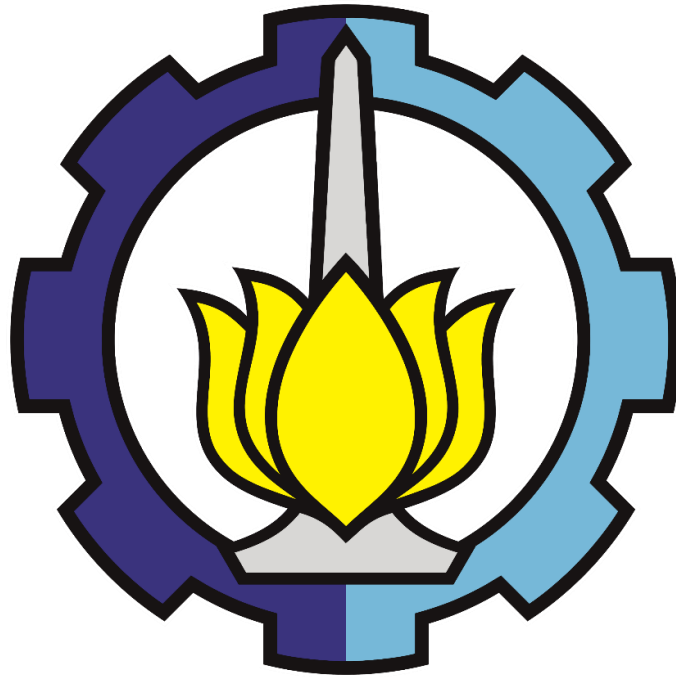


LAPORAN FINAL PROJECT PERANCANGAN KOMPONEN TERPROGRAM



Disusun Oleh :

Muhammad Hafidzh

5022221069

Dosen Mata Kuliah :

Dr. Rudy Dikairono, S.T., M.T.

Departemen Teknik Elektro

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

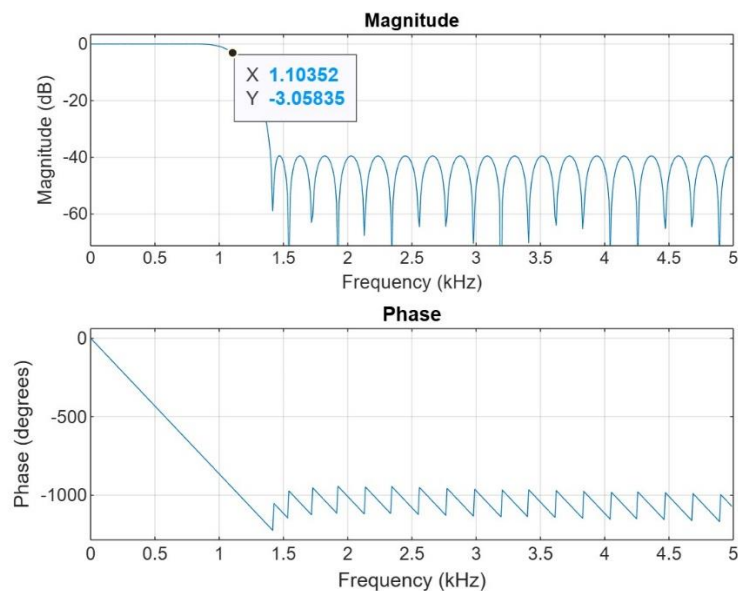
- **Langkah**

Langkah pertama dalam pengerjaan final project ini adalah melakukan sampling sinyal ADC dari function generator ke FPGA, menggunakan modul Qsys bernama hello_adc. Data hasil sampling berupa sinyal digital dengan resolusi 12-bit. Selanjutnya, data tersebut difilter menggunakan FIR low-pass filter, di mana koefisien filter telah dihitung sebelumnya menggunakan MATLAB. Setelah itu, data hasil filter kemudian dikirimkan secara paralel dalam format 12-bit ke Arduino Due. Pada Arduino Due, data digital tersebut dibaca dan diubah kembali menjadi sinyal analog melalui pin DAC untuk output.

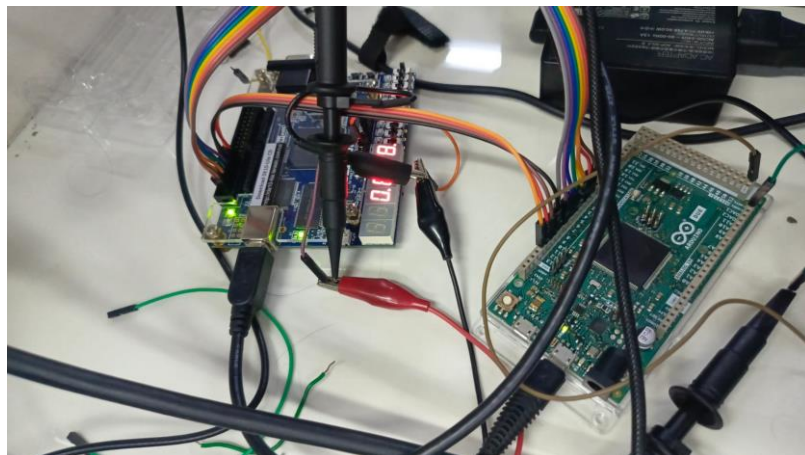
- **Hasil**

Plot Respon Frekuensi pada matlab

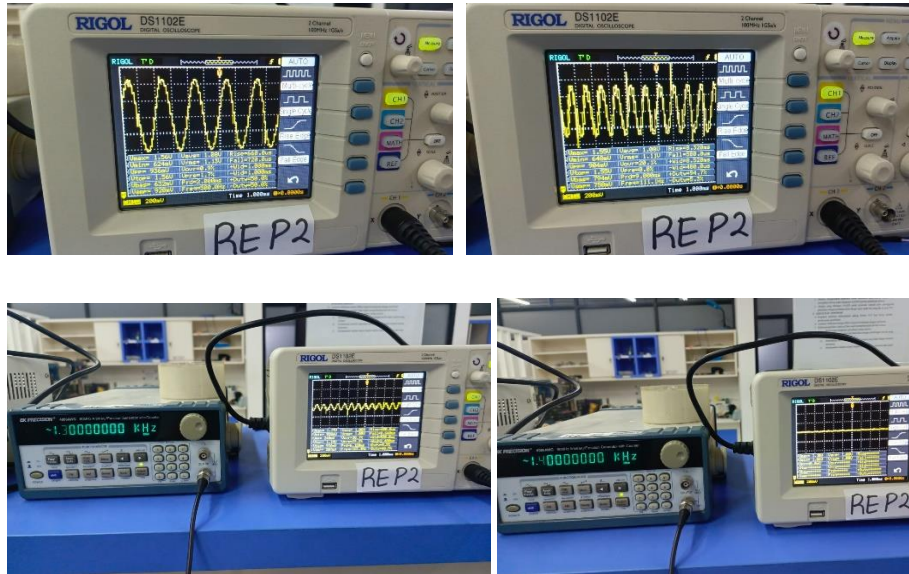
```
[n,fo,ao,w] = firpmord([860 1400],[1 0],[0.001 0.01],10000);  
b = firpm(n,fo,ao,w);  
freqz(b,1,[],10000)
```



Rangkaian



Hasil Filter dan DAC



- VHDL
Top Level

```

1 top_level.vhd
2
3 architecture t of top_level is
4     component hello_adc is
5         port (
6             adc_control_core_command_valid : in std_logic;
7             -- channel
8             adc_control_core_command_channel : in std_logic_vector(4 downto 0) := (others => '0');
9             -- startofpacket
10            adc_control_core_command_startofpacket : in std_logic := '0';
11            -- endofpacket
12            adc_control_core_command_endofpacket : in std_logic := '0';
13            -- ready
14            adc_control_core_command_ready : out std_logic;
15            -- adc_control_core_response_valid
16            adc_control_core_response_valid : out std_logic;
17            -- channel
18            adc_control_core_response_channel : out std_logic_vector(4 downto 0);
19            -- data
20            adc_control_core_response_data : out std_logic_vector(11 downto 0);
21            -- startofpacket
22            adc_control_core_response_startofpacket : out std_logic;
23            -- endofpacket
24            adc_control_core_response_endofpacket : out std_logic;
25            -- clk
26            clk_clk : in std_logic := '0';
27            clock_bridge_out_clk_clk : out std_logic;
28            clock_bridge_out_clk_clk : out std_logic;
29            reset_reset_n : in std_logic := '0';
30        );
31    end component hello_adc;
32
33    component FIR_filter is
34        port (
35            clk : in std_logic;
36            reset : in std_logic;
37            data_in : in std_logic_vector(11 downto 0);
38            data_out : out std_logic_vector(11 downto 0)
39        );
40    end component;
41
42    component clk_divider
43        port (
44            clk_in : in STD_100MHz;
45            reset : in STD_100MHz;
46            clk_out : out STD_100MHz
47        );
48    end component;
49
50    -- ADC signals
51    signal req_channel, cur_channel : std_logic_vector(4 downto 0);
52    signal sample_data : std_logic_vector(11 downto 0);
53    signal adc_cc_command_ready : std_logic;
54    signal adc_cc_response_valid : std_logic;
55    signal adc_cc_response_channel : std_logic_vector(4 downto 0);
56    signal adc_cc_response_data : std_logic_vector(11 downto 0);
57    -- System clock and reset
58    signal sys_clk, areset, reset : std_logic;
59
60    signal filtered_data : std_logic_vector(11 downto 0);
61    signal clk_div_10k : std_logic;
62
63 begin
64
65     reset <= not KEY(0);
66     reset <= not reset;
67
68     clk_div_10k <= clk_divider
69     port map (
70         clk_in => MAX10_CLK_50,
71         reset => reset,
72         clk_out => clk_div_10k
73     );
74
75     -- calculate channel used for sampling
76     -- Available channels on ADC12 are 1-6
77     -- use slide switches (SW) to select the channel
78     SW2 downto 0) down map to analog ADC_180
79     adc_command <= process(sys_clk, SW, adc_cc_command_ready)
80         variable temp : std_logic_vector(4 downto 0) := (others => '0');
81     begin
82         if rising_edge(sys_clk) then
83             if (adc_cc_command_ready = '1') then
84                 temp(2 downto 0) := std_logic_vector(unsigned(SW(2 downto 0)) + 1);
85             end if;
86             req_channel <= temp;
87         end process;
88
89         -- read the sampled value from the ADC
90         adc_read <= process(sys_clk, adc_cc_response_valid)
91             variable reading : std_logic_vector(11 downto 0) := (others => '0');
92             variable ch : std_logic_vector(4 downto 0) := (others => '0');
93         begin
94             if rising_edge(sys_clk) then
95                 if (adc_cc_response_valid = '1') then
96                     reading <= adc_cc_response_data;
97                     ch <= adc_cc_response_channel;
98                 end if;
99                 cur_channel <= ch;
100                sample_data <= reading;
101            end process;
102
103            -- instantiate QSYS subsystem with ADC and PLL
104            qsys_u0 : hello_adc
105            port map (
106                -- command always valid
107                adc_control_core_command_valid => '1',
108                adc_control_core_command_channel => req_channel,
109                startofpacket and endofpacket are ignored in adc_control_core
110                adc_control_core_command_startofpacket => '1',
111                adc_control_core_command_endofpacket => '1',
112                adc_control_core_command_ready => adc_cc_command_ready,
113                adc_control_core_response_valid => adc_cc_response_valid,
114                adc_control_core_response_channel => adc_cc_response_channel,
115                adc_control_core_response_data => adc_cc_response_data,
116                adc_control_core_response_startofpacket => '1',
117                adc_control_core_response_endofpacket => '1',
118                clk_clk => MAX10_CLK_50,
119                clock_bridge_out_clk_clk => sys_clk,
120                reset_reset_n => reset
121            );
122
123            -- instantiate FIR filter

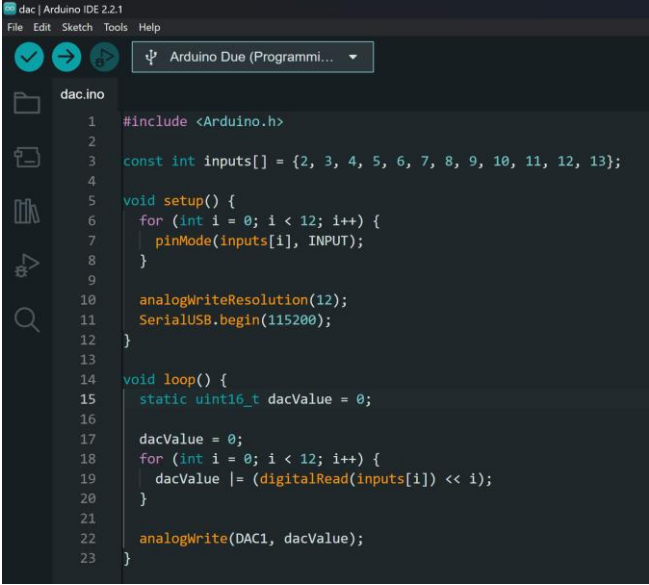
```

```

179 -- Instantiate FIR Filter
180 FIR_Filter_Instance : FIR_Filter
181 port map (
182     clk     => clk_div_10k,
183     reset   => reset,
184     data_in  => sample_data,
185     data_out => filtered_data
186 );
187 GPD(0) <= filtered_data(0);
188 GPD(1) <= filtered_data(1);
189 GPD(2) <= filtered_data(2);
190 GPD(3) <= filtered_data(3);
191 GPD(4) <= filtered_data(4);
192 GPD(5) <= filtered_data(5);
193 GPD(6) <= filtered_data(6);
194 GPD(7) <= filtered_data(7);
195 GPD(8) <= filtered_data(8);
196 GPD(9) <= filtered_data(9);
197 GPD(10) <= filtered_data(10);
198 GPD(11) <= filtered_data(11);
199 -- GPD(0) <= sample_data(0);
200 -- GPD(1) <= sample_data(1);
201 -- GPD(2) <= sample_data(2);
202 -- GPD(3) <= sample_data(3);
203 -- GPD(4) <= sample_data(4);
204 -- GPD(5) <= sample_data(5);
205 -- GPD(6) <= sample_data(6);
206 -- GPD(7) <= sample_data(7);
207 -- GPD(8) <= sample_data(8);
208 -- GPD(9) <= sample_data(9);
209 -- GPD(10) <= sample_data(10);
210 -- GPD(11) <= sample_data(11);
211 end architecture A;
212
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity FIR_Filter is
7     Generic (
8         N          : integer := 8;
9         DATA_WIDTH : integer := 12
10    );
11    Port (
12        clk      : in  STD_LOGIC;
13        reset    : in  STD_LOGIC;
14        data_in  : in  signed(DATA_WIDTH-1 downto 0);
15        data_out : out signed(DATA_WIDTH-1 downto 0)
16    );
17 end FIR_Filter;
18
19 architecture Behavioral of FIR_Filter is
20     type coeff_array is array (0 to 48) of signed(DATA_WIDTH-1 downto 0);
21     type coeff_array1 is array (0 to 48) of signed(DATA_WIDTH*2-1 downto 0);
22     constant C : coeff_array := (
23         to_signed(2, DATA_WIDTH), to_signed(-5, DATA_WIDTH), to_signed(-2, DATA_WIDTH), to_signed(1, DATA_WIDTH),
24         to_signed(4, DATA_WIDTH), to_signed(6, DATA_WIDTH), to_signed(5, DATA_WIDTH), to_signed(0, DATA_WIDTH),
25         to_signed(-6, DATA_WIDTH), to_signed(-11, DATA_WIDTH), to_signed(-10, DATA_WIDTH), to_signed(-2, DATA_WIDTH),
26         to_signed(10, DATA_WIDTH), to_signed(20, DATA_WIDTH), to_signed(21, DATA_WIDTH), to_signed(9, DATA_WIDTH),
27         to_signed(-13, DATA_WIDTH), to_signed(-36, DATA_WIDTH), to_signed(-46, DATA_WIDTH), to_signed(-30, DATA_WIDTH),
28         to_signed(15, DATA_WIDTH), to_signed(83, DATA_WIDTH), to_signed(157, DATA_WIDTH), to_signed(213, DATA_WIDTH),
29         to_signed(234, DATA_WIDTH), to_signed(213, DATA_WIDTH), to_signed(157, DATA_WIDTH), to_signed(83, DATA_WIDTH),
30         to_signed(15, DATA_WIDTH), to_signed(-30, DATA_WIDTH), to_signed(-46, DATA_WIDTH), to_signed(-36, DATA_WIDTH),
31         to_signed(-13, DATA_WIDTH), to_signed(9, DATA_WIDTH), to_signed(21, DATA_WIDTH), to_signed(20, DATA_WIDTH),
32         to_signed(10, DATA_WIDTH), to_signed(-2, DATA_WIDTH), to_signed(-10, DATA_WIDTH), to_signed(-11, DATA_WIDTH),
33         to_signed(-6, DATA_WIDTH), to_signed(0, DATA_WIDTH), to_signed(5, DATA_WIDTH), to_signed(5, DATA_WIDTH),
34         to_signed(4, DATA_WIDTH), to_signed(1, DATA_WIDTH), to_signed(-2, DATA_WIDTH), to_signed(-5, DATA_WIDTH),
35         to_signed(2, DATA_WIDTH)
36     );
37     signal shift_reg : coeff_array1 := (others => (others => '0'));
38     signal mult : coeff_array1;
39     signal acc : signed(DATA_WIDTH*2 downto 0) := (others => '0');
40
41 begin
42     process(clk, reset)
43     begin
44         if reset = '1' then
45             for i in 1 to 48 loop
46                 shift_reg(i) <= (others => '0');
47             end loop;
48             data_out <= (others => '0');
49         elsif rising_edge(clk) then
50             for i in 48 downto 1 loop
51                 shift_reg(i) <= shift_reg(i-1);
52             end loop;
53             shift_reg(0) <= data_in;
54             for i in 0 to 48 loop
55                 mult(i) <= shift_reg(i) * C(i);
56             end loop;
57             acc <= (resize(mult(0), 25) + resize(mult(1), 25) + resize(mult(2), 25) + resize(mult(3), 25) +
58                 resize(mult(4), 25) + resize(mult(5), 25) + resize(mult(6), 25) + resize(mult(7), 25) +
59                 resize(mult(8), 25) + resize(mult(9), 25) + resize(mult(10), 25) + resize(mult(11), 25) +
60                 resize(mult(12), 25) + resize(mult(13), 25) + resize(mult(14), 25) + resize(mult(15), 25) +
61                 resize(mult(16), 25) + resize(mult(17), 25) + resize(mult(18), 25) + resize(mult(19), 25) +
62                 resize(mult(20), 25) + resize(mult(21), 25) + resize(mult(22), 25) + resize(mult(23), 25) +
63                 resize(mult(24), 25) + resize(mult(25), 25) + resize(mult(26), 25) + resize(mult(27), 25) +
64                 resize(mult(28), 25) + resize(mult(29), 25) + resize(mult(30), 25) + resize(mult(31), 25) +
65                 resize(mult(32), 25) + resize(mult(33), 25) + resize(mult(34), 25) + resize(mult(35), 25) +
66                 resize(mult(36), 25) + resize(mult(37), 25) + resize(mult(38), 25) + resize(mult(39), 25) +
67                 resize(mult(40), 25) + resize(mult(41), 25) + resize(mult(42), 25) + resize(mult(43), 25) +
68                 resize(mult(44), 25) + resize(mult(45), 25) + resize(mult(46), 25) + resize(mult(47), 25) +
69                 resize(mult(48), 25)) / 1000;
70             data_out <= acc(DATA_WIDTH-1 downto 0);
71         end if;
72     end process;
73 end Behavioral;

```

- Arduino



```

1  #include <Arduino.h>
2
3  const int inputs[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
4
5  void setup() {
6      for (int i = 0; i < 12; i++) {
7          pinMode(inputs[i], INPUT);
8      }
9
10     analogWriteResolution(12);
11     SerialUSB.begin(115200);
12 }
13
14 void loop() {
15     static uint16_t dacValue = 0;
16
17     dacValue = 0;
18     for (int i = 0; i < 12; i++) {
19         dacValue |= (digitalRead(inputs[i]) << i);
20     }
21
22     analogWrite(DAC1, dacValue);
23 }

```

- Analisa

VHDL di atas merupakan implementasi dari sebuah Finite Impulse Response (FIR) lowpass filter digital dengan arsitektur berbasis shift register. Filter ini menerima masukan berupa data digital 12-bit dan menghasilkan keluaran yang juga berupa data 12-bit yang telah difilter. Koefisien filter yang telah didapatkan dari perhitungan matlab terdapat 49 orde. Setiap koefisien dalam array C merepresentasikan bobot yang digunakan untuk mengalikan data input yang tersimpan dalam shift register. Pada bagian top_level, clock yang digunakan untuk filter ialah clock yang telah dibagi yakni menjadi 10KHz, dimana harus sesuai dengan frekuensi sampling saat menghitung koefisien filter pada matlab.

Proses filter dimulai dengan reset awal, di mana seluruh register geser (shift_reg) diinisialisasi dengan nilai nol. Selanjutnya, saat sinyal clk mengalami rising edge, data input terbaru akan dimasukkan ke posisi paling awal dari register geser, sementara data lainnya digeser ke posisi berikutnya. Dengan cara ini, data input yang lebih lama secara bertahap bergerak ke ujung register. Proses ini menciptakan sebuah buffer yang menyimpan data input terbaru hingga data input yang telah diterima pada 48 siklus sebelumnya. Setelah data terbaru dimasukkan ke register, setiap nilai dalam register dikalikan dengan koefisien filter masing-masing menggunakan operasi perkalian. Hasil dari semua perkalian ini disimpan dalam array mult. Untuk mendapatkan hasil filter, semua nilai hasil perkalian dijumlahkan dalam sinyal akumulator (acc) setelah di-resize ke panjang data yang lebih besar untuk mencegah overflow. Akumulator ini membagi hasil penjumlahan dengan konstanta 1000 untuk mengatur skala keluaran. Hasil akhir dari proses ini adalah nilai keluaran filter (data_out), yang diambil dari bit paling signifikan hingga bit paling tidak signifikan sesuai dengan lebar data yang didefinisikan.

Pada sisi Arduino, data digital hasil filter diterima dalam format paralel 12-bit dari FPGA melalui pin digital dan dikonversi menjadi sinyal analog menggunakan DAC internal. Proses ini dimulai dengan mengatur setiap pin sebagai input dan menginisialisasi resolusi DAC sebesar 12-bit. Data paralel dibaca bit-per-bit, disusun menjadi nilai 12-bit, dan dikeluarkan sebagai sinyal analog pada pin DAC1.