Laporan Tugas Besar
Desain FPGA dan SoC 2024

Desain FPGA dan SoC

Kelompok : 5
Nama-NIM Anggota 1 : MUHAMMAD IRFAN ZIDNY / 1102223172
Nama-NIM Anggota 2 : FARHAN RUSYDAN ARIEF / 1102223164
Nama-NIM Anggota 3 : MOH ELDIVO ALSYAWAL OTOLUWA / 1102223205

Judul

FPGA-BASED FIR FILTER DESIGN

Deskripsi

Tugas besar ini merancang sistem pemrosesan sinyal digital berupa Filter FIR (Finite Impulse Response) 4-Tap pada FPGA. Sistem ini mengimplementasikan arsitektur hardware yang dikontrol oleh Finite State Machine (FSM) untuk mengatur aliran data dan operasi aritmatika. FSM bertugas memastikan proses pengambilan sampel data (sampling) dan pergeseran register (shifting) berjalan sinkron dan stabil setiap kali pengguna memberikan perintah step.

Fungsi

1. Penyaringan Sinyal: Meredam perubahan data drastis (noise) menggunakan algoritma Moving Average.
2. Kontrol Sekuensial: Mendemonstrasikan penggunaan State Machine untuk mengatur timing proses digital.
3. Visualisasi Data: Menampilkan hasil filter secara real-time ke 7-Segment Display dalam format desimal.

Fitur dan Spesifikasi

Fitur

Fitur	Deskripsi
4-Tap Moving Average Algorithm	Mampu meratakan fluktuasi sinyal dengan menghitung rata-rata dari 4 sampel data terakhir.
Eksekusi sekuensial per step	Mode operasi manual yang memungkinkan pengguna mengamati perubahan data per satu siklus clock
Visualisasi 7-Segment	Hasil filter dapat diamati melalui 7 segment
Edge detection	Mampu mendeteksi input tombol sebagai trigger input dan pemrosesan data

Spesifikasi

1. Platform Hardware: FPGA Altera Cyclone V (DE1-SoC Board).
2. Clock Frequency: 50 MHz (System Clock).
3. Controller: Finite State Machine (2-State: IDLE, SHIFT).
4. Arsitektur Data:
   a. Input 8-bit
   b. Akumulator 10-bit

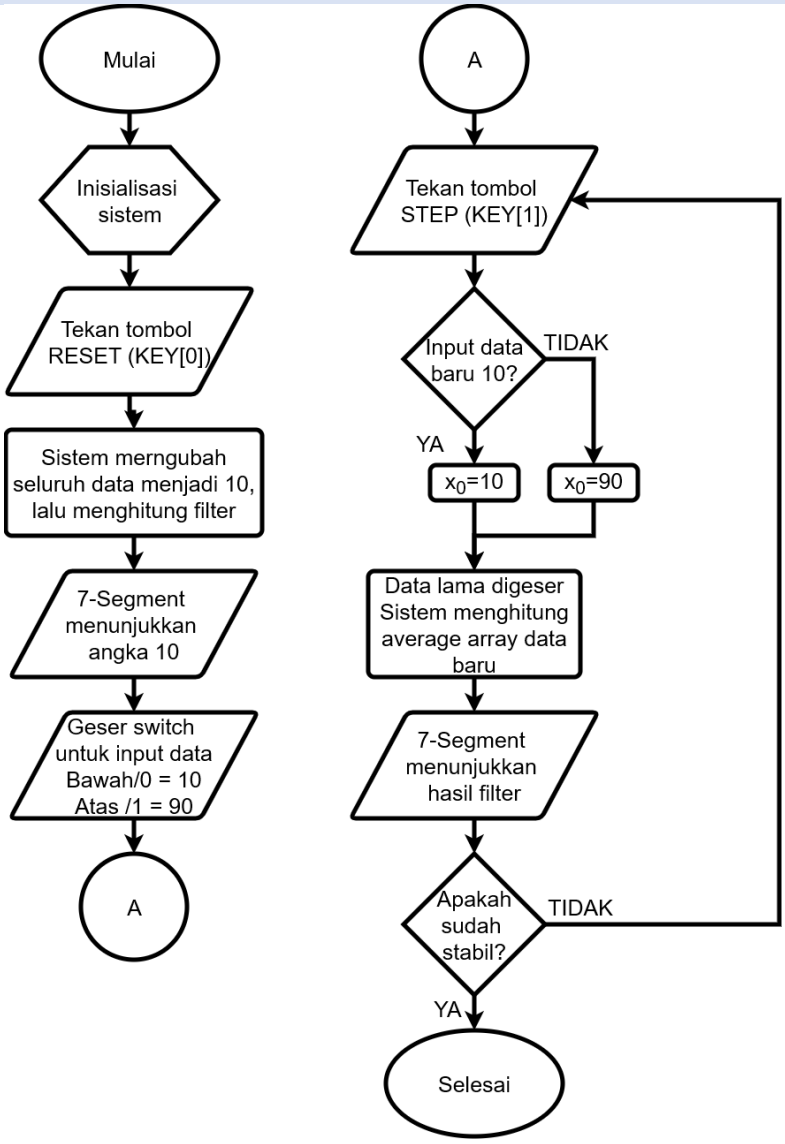
c. Output 8-bit.

5. Metode Filter: 4-Tap Moving Average

$$(y[n] = \frac{\sum_{i=1}^{n} x_i}{n})$$

6. Input:
   a. KEY[0]: Reset Asinkron (Active Low).
   b. KEY[1]: Step Trigger (Active Low) dengan Edge Detection.
   c. SW[0]: Selector Input Data (10 atau 90).
7. Output:
   a. 7-Segment (Nilai Filter)
   b. LEDR[7:0]: Monitor Data Input (Menampilkan nilai biner dari posisi Switch).
   c. LEDR[9] (Button Press Indicator).

Cara Penggunaan



1. Sistem diaktifkan dengan memberikan catu daya pada board FPGA dan mengunggah program (bitstream).
2. Sistem melakukan inisialisasi awal saat tombol Reset (KEY[0]) ditekan, mengembalikan seluruh data register dan tampilan ke nilai default (10).

3. Pengguna menentukan target input dengan menggeser Switch (SW 0) ke posisi Bawah (nilai 10) atau Atas (nilai 90).
4. Lampu indikator LEDR 0-7 menyala seketika untuk memvisualisasikan data biner dari input yang dipilih pengguna.
5. Tombol Step (KEY[1]) ditekan untuk memicu FSM mengambil satu sampel data baru ke dalam antrian filter.
6. Lampu status LEDR 9 menyala sesaat selama tombol Step ditekan sebagai tanda respons sistem terhadap input pengguna.
7. Sistem memproses data secara internal (geser, jumlah, dan bagi), lalu menampilkan hasil rata-rata pada 7-segment display.
8. Jika angka pada layar belum mencapai steady state (10 atau 90), pengguna kembali menekan tombol Step (KEY[1]) secara berulang.
9. Proses berulang terus menerus hingga output pada layar menjadi stabil (steady state) dan sama dengan nilai input yang dipilih.

Blok Diagram

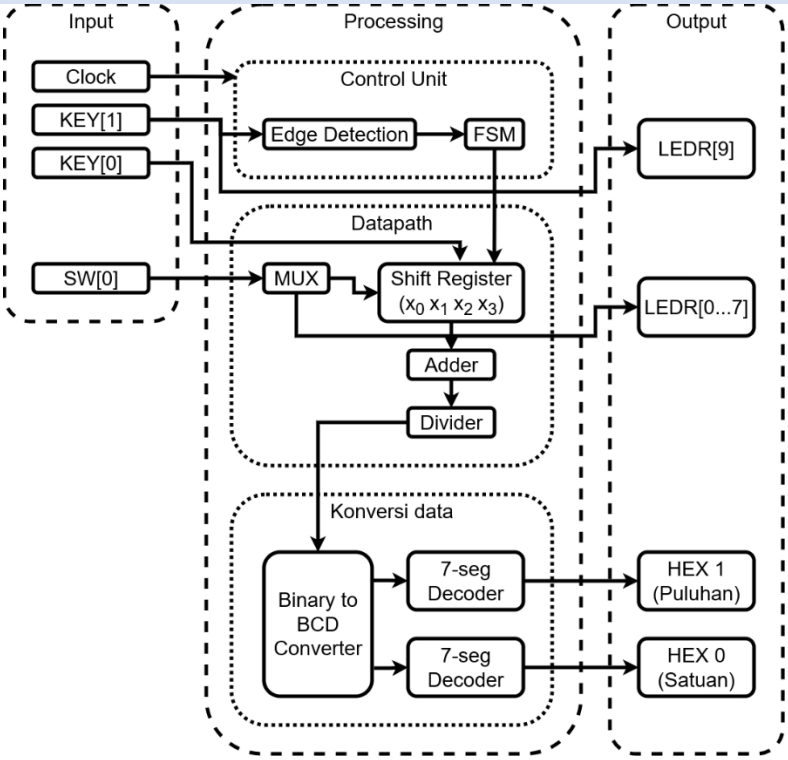


Diagram blok dibagi menjadi tiga kolom utama: Input, Processing (yang mencakup Control Unit, Datapath, dan Konversi Data), serta Output.

1. Blok Input (Masukan)

   Bagian ini menangani sinyal fisik dari pengguna dan clock sistem.

   a. Clock: Sinyal detak utama (50 MHz) yang masuk ke Control Unit untuk menyinkronkan kerja sistem.
   b. KEY[1] (Tombol Step): Tombol pemicu proses filter. Sinyal ini memiliki dua jalur:
      i. Masuk ke Control Unit (via Edge Detection) sebagai pemicu logika.
      ii. Masuk langsung ke Output LEDR[9] sebagai indikator fisik status tekanan tombol.

c. KEY[0] (Tombol Reset): Tombol untuk mereset sistem. Jalurnya bercabang ke dua blok vital untuk memastikan reset menyeluruh:
    i. Ke Control Unit (FSM): Mengembalikan status mesin ke kondisi IDLE.
    ii. Ke Datapath (Shift Register): Mengembalikan nilai data register ke nilai awal (10).
d. SW[0] (Switch Data): Sakelar untuk memilih nilai masukan, apakah 10 (Low) atau 90 (High).

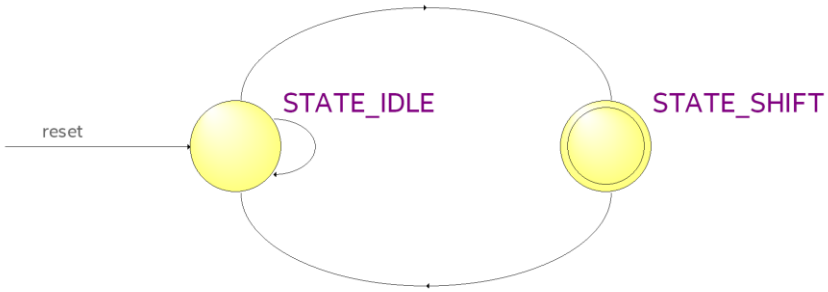
2. Blok Processing (Pemrosesan Utama)

Ini adalah inti dari FPGA yang terdiri dari tiga sub-blok fungsional:

a. Control Unit (Unit Kendali) Bertugas mengatur timing dan logika sistem.
    i. Edge Detection: Mengubah sinyal mekanik tombol KEY[1] menjadi satu pulsa digital (one-shot) agar stabil.
    ii. FSM (Finite State Machine): Otak sistem yang mengatur perpindahan state. FSM mengirimkan sinyal Enable ke Datapath hanya saat pulsa valid diterima dari Edge Detection.
b. Datapath (Jalur Data)Bertugas melakukan penyimpanan dan operasi matematika.
    i. MUX (Multiplexer): Memilih data input berdasarkan SW[0]. Outputnya dikirim ke Shift Register untuk diproses, dan juga dicabangkan ke LEDR[0...7] untuk monitoring.
    ii. Shift Register : Empat memori penyimpan data antrian. Data hanya bergeser masuk jika mendapat izin (enable) dari FSM.
    iii. Adder & Divider: Menjumlahkan keempat data register lalu membaginya dengan 4 (melalui bit-shifting) untuk mendapatkan nilai rata-rata.
c. Konversi Data (Data Conversion)Sub-blok ini (sebelumnya Output Interface) bertugas menerjemahkan hasil hitungan mesin ke format layar manusia.
    i. Binary to BCD Converter: Mengubah hasil rata-rata (format biner 8-bit) menjadi format Desimal yang terpisah antara digit Puluhan dan Satuan.
    ii. 7-seg Decoder: Mengubah kode angka desimal (BCD) menjadi sinyal listrik untuk menyalakan segmen lampu.

3. Blok Output (Keluaran Fisik)Bagian ini adalah antarmuka visual hasil keluaran sistem.
    a. LEDR[9]: Indikator status tombol (Menyala saat KEY[1] ditekan).
    b. LEDR[0...7]: Indikator data masukan (Menampilkan nilai biner dari data yang dipilih Switch secara real-time).
    c. HEX 1 (Puluhan) & HEX 0 (Satuan): Layar 7-Segment yang menampilkan hasil akhir rata-rata filter dalam angka desimal yang mudah dibaca.

FSM



sistem dirancang menggunakan konsep Finite State Machine (FSM) tipe Moore, di mana output kendali (sinyal enable) hanya bergantung pada current state. FSM ini berfungsi untuk menyinkronkan proses pengambilan data (sampling) agar sesuai dengan input tombol pengguna. Mesin ini memiliki dua state utama, yaitu:

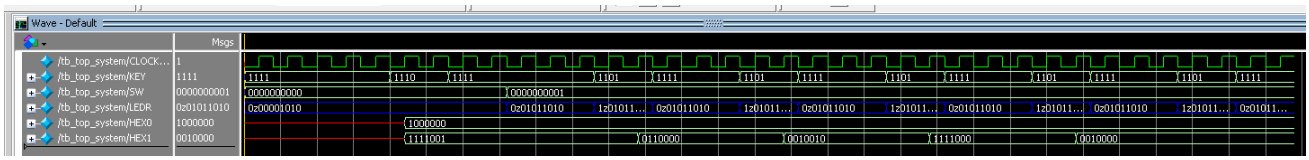
1. STATE_IDLE (Kondisi Tunggu): Merupakan kondisi default saat sistem baru dinyalakan atau setelah di-reset. Pada fase ini, sistem bersifat pasif (standby). FSM menonaktifkan sinyal enable ke jalur data (datapath), sehingga Shift Register mempertahankan nilai data sebelumnya (latching) dan tidak terjadi perubahan output pada sistem aritmatika.
2. STATE_SHIFT (Kondisi Proses): Merupakan fase eksekusi aktif. Pada state ini, FSM mengirimkan sinyal logika '1' ke Shift Register. Hal ini memicu register untuk mengambil data input baru dari MUX dan menggeser data secara simultan dalam satu siklus clock.

Mekanisme Transisi:

Perpindahan dari IDLE ke SHIFT dipicu oleh sinyal step_trigger yang dihasilkan oleh modul Edge Detection saat tombol KEY[1] ditekan. Transisi ini memastikan sistem merespons input pengguna. Setelah berada di STATE_SHIFT selama satu periode clock (waktu yang cukup untuk melakukan satu kali operasi pergeseran data), FSM dirancang untuk melakukan transisi otomatis (unconditional transition) kembali ke STATE_IDLE. Mekanisme "satu siklus" ini sangat krusial untuk menjamin bahwa satu kali penekanan tombol fisik hanya akan diproses sebagai satu kali pengambilan data, mencegah terjadinya multiple-input error atau pembacaan ganda.

Hasil simulasi dan Analisis

Simulasi awal dilakukan dengan menjalankan kode test bench dan melakukan simulasi modelsim. Berikut Adalah hasil waveformmodelsim



Berdasarkan waveform di atas, berikut adalah analisis langkah demi langkah sesuai urutan waktu dari kiri ke kanan:

1. Kondisi Awal (Reset):

a. Terlihat sinyal KEY bernilai 1110 (Bit 0 low), yang menandakan tombol Reset ditekan.
b. Output HEX1 bernilai 1111001 (Angka 1) dan HEX0 bernilai 1000000 (Angka 0).
c. Hasil: Sistem terinisialisasi dengan nilai 10.

2. Perubahan Input (Switch):
   a. Sinyal SW berubah dari 0 menjadi 1 (Input 90).
   b. Sinyal LEDR langsung berubah dari ...01010 (10) menjadi ...1011010 (90).
   c. HEX1 dan HEX0 belum berubah (masih 10). FSM menahan data sampai ada trigger.
3. Proses Step: Setiap kali sinyal KEY berubah menjadi 1101 (Bit 1 low / Tombol Step ditekan), output berubah sesuai logika filter rata-rata:

Step Ke-	**Kode HEX1 (Puluhan)**	**Kode HEX0 (Satuan)**	**Nilai Desimal**	**Analisis Perhitungan**
1	0110000 (Angka 3)	1000000 (Angka 0)	**30**	$\frac{90+10+10+10}{4} = 30$
2	0010010 (Angka 5)	1000000 (Angka 0)	**50**	$\frac{90+90+10+10}{4} = 50$
3	1111000 (Angka 7)	1000000 (Angka 0)	**70**	$\frac{90+90+90+10}{4} = 70$
4	0010000 (Angka 9)	1000000 (Angka 0)	**90**	$\frac{90+90+90+90}{4} = 90$

Berdasarkan analisis terhadap waveform simulasi, sistem terbukti mampu menjalankan fungsi Moving Average Filter 4-Tap dengan presisi tinggi sesuai spesifikasi perancangan. Pada fase awal, sinyal reset berhasil menginisialisasi seluruh register sehingga output HEX berada pada nilai stabil 10. Ketika terjadi perubahan input mendadak dari logika '0' (nilai 10) ke logika '1' (nilai 90) pada sinyal SW, sistem menunjukkan respons yang terbagi dua: indikator LEDR merespons seketika secara real-time, sedangkan output hasil filter (HEX) tetap bertahan pada nilai lama. Hal ini membuktikan bahwa Finite State Machine (FSM) berhasil menahan pembaruan data hingga sinyal pemicu yang valid diterima. Selanjutnya, saat tombol Step (KEY[1]) ditekan secara berurutan (terlihat pada transisi sinyal KEY ke logika low), output 7-Segment mengalami perubahan transien yang linear, yaitu bergerak dari 10 menjadi 30, 50, 70, hingga akhirnya mencapai kondisi steady state di angka 90. Respons bertahap ini memverifikasi bahwa unit aritmatika (penjumlah dan pembagi) berfungsi akurat, sekaligus memvalidasi karakteristik sistem sebagai Low Pass Filter yang efektif meredam lonjakan data input melalui mekanisme sampling sekuensia

Skenario pelaksanaan simulasi hardware dimulai dengan melakukan transfer sistem dari **Quartus** ke **FPGA**. Setelah sistem aktif dan berada pada kondisi **steady state**, seven segment menampilkan nilai 10. Selanjutnya, **SW[0]** dinyalakan (bernilai logika 1) untuk mengubah nilai input menjadi 90.

Kemudian **Button[1]** ditekan untuk memberikan input ke sistem sesuai nilai yang ditentukan oleh SW[0]. Setiap penekanan tombol menyebabkan satu data baru masuk ke dalam memori **Filter FIR**, di mana empat register akan bergeser dengan membuang data lama dan menggantinya dengan data baru. Oleh karena itu, nilai output tidak langsung menjadi 90, melainkan meningkat secara bertahap sesuai mekanisme moving average.

Pada hasil pengujian menunjukkan bahwa sistem Moving Average 4-Tap bekerja sesuai perancangan. Pada kondisi awal dengan input konstan 10, output berada pada keadaan stabil di nilai 10. Saat input dinaikkan menjadi 90, output tidak berubah secara instan, melainkan meningkat bertahap menjadi

30, 50, 70, hingga 90 setiap penekanan tombol step. Hal ini menunjukkan bahwa sistem bekerja secara sinkron dan memanfaatkan memori historis.

Pada saat input diturunkan kembali ke 10, output juga mengalami penurunan bertahap akibat adanya sisa data yang tersimpan di dalam shift register. Dengan demikian, dapat disimpulkan bahwa fungsi shift register dan perhitungan rata-rata berjalan dengan benar sesuai prinsip filter FIR Moving Average.

Lampiran (Kode Verilog)

top_system.v

```
module top_system (
    input CLOCK_50,
    input [3:0] KEY,
    input [9:0] SW,
    output [9:0] LEDR,
    output [6:0] HEX0,
    output [6:0] HEX1
);

    reg key1_prev;
    wire step_trigger;

    always @(posedge CLOCK_50) begin
        key1_prev <= KEY[1];
    end
    assign step_trigger = (key1_prev == 1'b1 && KEY[1] == 1'b0);

    reg [7:0] input_data;
    always @(*) begin
        if (SW[0] == 1'b0) input_data = 8'd10;
        else input_data = 8'd90;
    end

    localparam STATE_IDLE  = 2'd0;
    localparam STATE_SHIFT = 2'd1;

    reg [1:0] current_state, next_state;

    reg [7:0] x0, x1, x2, x3;
```

```
wire [9:0] sum;
wire [7:0] filter_out;

always @(posedge CLOCK_50) begin
   if (KEY[0] == 1'b0) begin
      current_state <= STATE_IDLE;
   end else begin
      current_state <= next_state;
   end
end

always @(*) begin
   case (current_state)
      STATE_IDLE: begin
         if (step_trigger)
            next_state = STATE_SHIFT;
         else
            next_state = STATE_IDLE;
      end

      STATE_SHIFT: begin
         next_state = STATE_IDLE;
      end

      default: next_state = STATE_IDLE;
   endcase
end

always @(posedge CLOCK_50) begin
   if (KEY[0] == 1'b0) begin
      x0 <= 8'd10;
      x1 <= 8'd10;
      x2 <= 8'd10;
      x3 <= 8'd10;
   end else begin
      if (current_state == STATE_SHIFT) begin
         x0 <= input_data;
         x1 <= x0;
         x2 <= x1;
         x3 <= x2;
      end
```

```
    end
  end

  assign sum = x0 + x1 + x2 + x3;
  assign filter_out = sum[9:2];

  reg [3:0] digit_satuan;
  reg [3:0] digit_puluhan;

  always @(*) begin
    digit_satuan  = filter_out % 10;
    digit_puluhan = filter_out / 10;
  end

  assign LEDR[7:0] = input_data;

  assign LEDR[9] = ~KEY[1];

  seven_segment hex_low (
    .in(digit_satuan),
    .out(HEX0)
  );

  seven_segment hex_high (
    .in(digit_puluhan),
    .out(HEX1)
  );

endmodule
```

seven_segment.v

```
module seven_segment(
  input [3:0] in,
  output reg [6:0] out
);
  always @(*) begin
    case(in)
      4'h0: out = 7'b1000000;
      4'h1: out = 7'b1111001;
      4'h2: out = 7'b0100100;
      4'h3: out = 7'b0110000;
      4'h4: out = 7'b0011001;
```

```
            4'h5: out = 7'b0010010;
            4'h6: out = 7'b0000010;
            4'h7: out = 7'b1111000;
            4'h8: out = 7'b0000000;
            4'h9: out = 7'b0010000;
            4'hA: out = 7'b0001000;
            4'hB: out = 7'b0000011;
            4'hC: out = 7'b1000110;
            4'hD: out = 7'b0100001;
            4'hE: out = 7'b0000110;
            4'hF: out = 7'b0001110;
            default: out = 7'b1111111;
        endcase
    end
endmodule
```

tb_top_system.v

```
`timescale 1ns / 1ps

module tb_top_system();

    reg CLOCK_50;
    reg [3:0] KEY;
    reg [9:0] SW;

    wire [9:0] LEDR;
    wire [6:0] HEX0;
    wire [6:0] HEX1;

    top_system uut (
        .CLOCK_50(CLOCK_50),
        .KEY(KEY),
        .SW(SW),
        .LEDR(LEDR),
        .HEX0(HEX0),
        .HEX1(HEX1)
    );

    initial begin
        CLOCK_50 = 0;
        forever #10 CLOCK_50 = ~CLOCK_50;
    end
```

```
    initial begin
        KEY = 4'b1111;
        SW = 10'b0;
        #100;

        KEY[0] = 0;
        #40;
        KEY[0] = 1;
        #40;

        SW[0] = 1;
        #40;

        press_step_button();
        press_step_button();
        press_step_button();
        press_step_button();
        press_step_button();

        $stop;
    end

    task press_step_button;
        begin
            #20;
            KEY[1] = 0;
            #40;
            KEY[1] = 1;
            #40;
        end
    endtask

endmodule
```

Link Video Implementasi

LINK VIDEO

Atau

https://drive.google.com/drive/folders/13rEqHvps9pyjTTlEMXvs_CQ0o9x_8JPL?usp=sharing