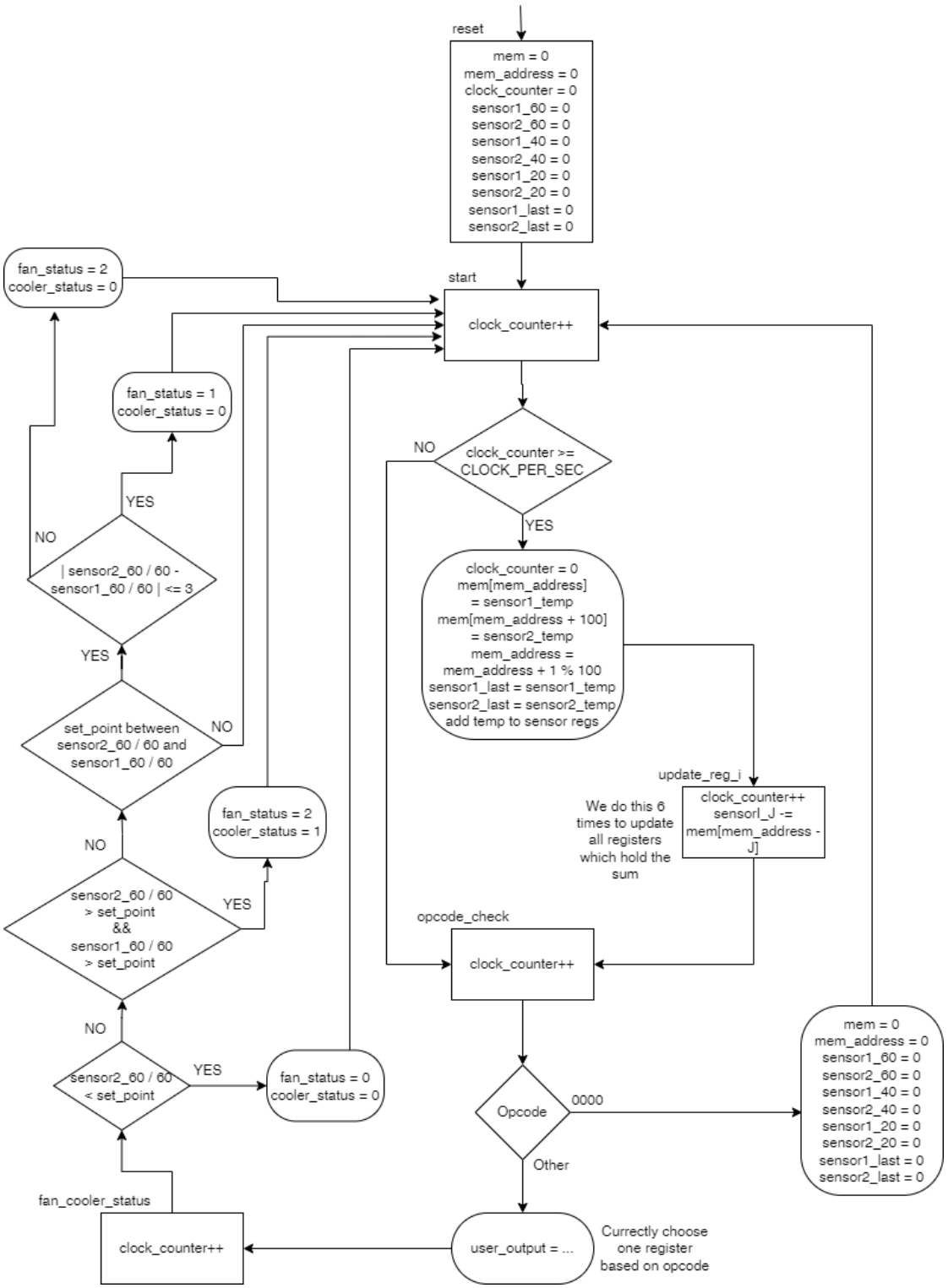
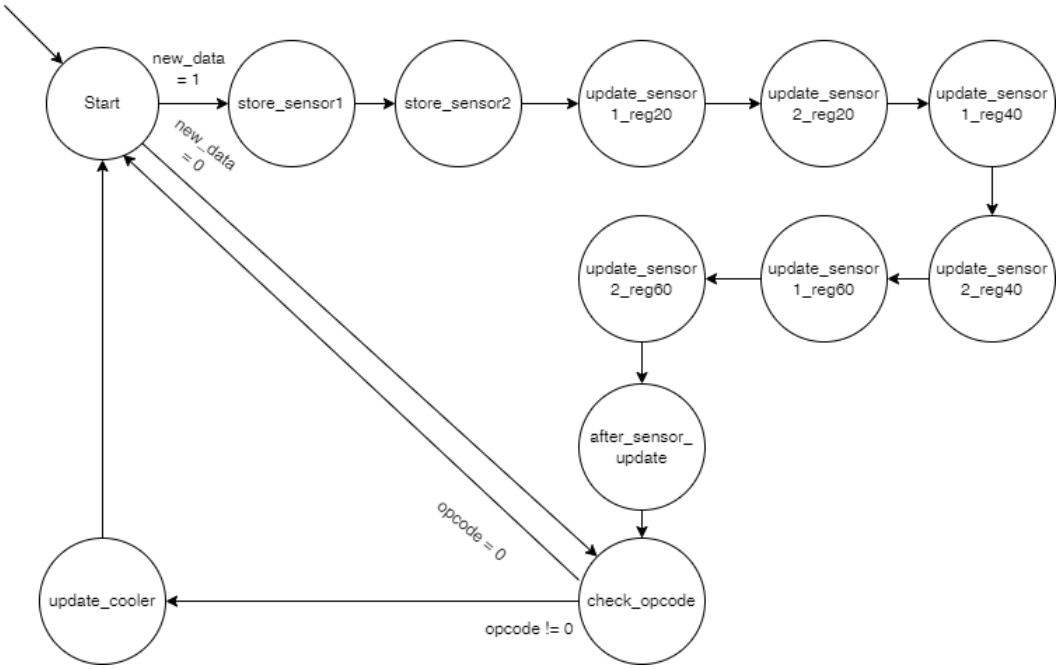
در ابتدا سعی می‌کنیم که ASM Chart مدار را طراحی کنیم. در طراحی من فرض شده است که هشت رجیستر داریم. برای هر دو سنسور چهار تا. یکی از آنها آخرین دمای ذخیره سازی شده را نگه داری می‌کنید. سه تای دیگر مجموع ۲۰، ۴۰ و ۶۰ ثانیه‌ی قبل را نگه داری می‌کنند. نکته‌ی مهمی که در این طراحی وجود دارد این است که فرض شده است در هر ثانیه مدار چندین کلاک می‌خورد. (با توجه به FSM حداقل ۱۰ کلاک.) دلیل نیاز داشتن به چندین کلاک این است که مموری ما یک خط آدرس و خروجی دارد. پس در هر کلاک فقط می‌توان یک عدد از آن خواند و نوشت و ما نیاز داریم که ۶ عدد از مموری بخوانیم (دمای ۲۰، ۴۰ و ۶۰ ثانیه‌ی گذشته) که بتوانیم آنها را از رجیسترها کم کنیم. همچنین برای ذخیره سازی دمای در حال حاضر نیز به دو کلاک زمان نیاز داریم.
حال مسئله‌ای که به وجود می‌آید این است که چطور می‌شود فهمید که کی یک ثانیه گذشته است. برای این موضوع از یک counter استفاده می‌کنیم. این counter در هر کلاک یکی روبه بالا می‌شمارد. در صورتی که عدد این counter از فرکانس مدار بیشتر شد، یک ثانیه گذشته است و باید که دمای جدید را خواند. دقت کنید که فرکانس مدار باید قبل از شروع مدار معلوم باشد. به همین منظور در طراحی نهایی من فرکانس را پارامتر کرده‌ام که به راحتی قابل تغییر باشد. همچنین دقت کنید که زمانی که این شمارنده بیشتر از فرکانس شد باید فرکانس را از آن کم کنیم.
من این سوال را به کمک یک مموری حل کردم. برای راحت تر شدن کار فرض کرده‌ام که ۱۰۰ خانه‌ی اول مموری برای سنسور اول هستند و ۱۰۰ خانه‌ی بعدی برای سنسور ۲. برای نگه داری آخرین آدرس نوشته شده در مموری از تنها یک شمارنده استفاده می‌کنیم که می‌تواند تا ۱۰۰ بشمارد و سپس ۰ می‌شود.
قسمت چک کردن opcode و تعیین حالت فن نیز به صورت روتین و نرمال انجام می‌شود.
می‌توانید ASM Chart مدار را در صفحه‌ی بعد مشاهده کنید.


reset
mem = 0
mem_address = 0
clock_counter = 0
sensor1_60 = 0
sensor2_60 = 0
sensor1_40 = 0
sensor2_40 = 0
sensor1_20 = 0
sensor2_20 = 0
sensor1_last = 0
sensor2_last = 0
start
clock_counter++
clock_counter >= CLOCK_PER_SEC
NO
YES
clock_counter = 0
mem[mem_address] = sensor1_temp
mem[mem_address + 100] = sensor2_temp
mem_address = mem_address + 1 % 100
sensor1_last = sensor1_temp
sensor2_last = sensor2_temp
add temp to sensor regs
update_reg_i
clock_counter++
sensorI_J -= mem[mem_address - J]
We do this 6 times to update all registers which hold the sum
opcode_check
clock_counter++
Opcode
0000
Other
mem = 0
mem_address = 0
sensor1_60 = 0
sensor2_60 = 0
sensor1_40 = 0
sensor2_40 = 0
sensor1_20 = 0
sensor2_20 = 0
sensor1_last = 0
sensor2_last = 0
user_output = ...
Currectly choose one register based on opcode
fan_cooler_status
clock_counter++
sensor2_60 / 60 < set_point
YES
fan_status = 0
cooler_status = 0
NO
sensor2_60 / 60 > set_point && sensor1_60 / 60 > set_point
YES
fan_status = 2
cooler_status = 1
NO
set_point between sensor2_60 / 60 and sensor1_60 / 60
NO
YES
| sensor2_60 / 60 - sensor1_60 / 60 | <= 3
NO
YES
fan_status = 1
cooler_status = 0
fan_status = 2
cooler_status = 0

حال با توجه به این ASM Chart می‌توان یک state diagram یا FSM طراحی کرد.



حال ماژول‌های مورد نیاز را طراحی می‌کنیم. در ابتدا برای رجیسترهای نگه دارنده‌ی آخرین دمای هر سنسور باید از DFFهایی استفاده کنیم که قابلیت enable یا load داشته باشد.

```
module DFFWithLoad #(parameter WIDTH = 8) (
    input wire clk,
    input wire clear,
    input wire load,
    input wire [WIDTH-1:0] in,
    output reg [WIDTH-1:0] out
);
    always @(posedge clk or posedge clear) begin
        if (clear)
            out <= 0;
        else if (load)
            out <= in;
    end
endmodule
```

حال برای رجیسترهای نگه دارنده‌ی جمع دما باید از رجیسترهایی استفاده کنیم که قابلیت جمع و تفریق یک عدد را نیز با عدد نگه داری شده در خودشان داشته باشند. همچنین امکان enable را به آن اضافه می‌کنیم که در صورتی که فعال نباشد عدد خروجی تغییری نمی‌کند. همچنین برای تعیین جمع یا تفریق از یک سیگنال استفاده می‌کنیم که زمانی که ۱ باشد تفریق و در غیر این صورت جمع انجام می‌شود.

```
module SensorSumHolder #(parameter WIDTH = 16) (
    input wire clk,
```

```
    input wire reset,
    input wire enable, // if enable is zero nothing will be changed
    input wire sub_sum_not, // if 0 out will be in + out otherwise out =
    out - in
    input wire signed [WIDTH-1:0] in,
    output reg signed [WIDTH-1:0] out
);
    always @(posedge clk or posedge reset) begin
        if (reset)
            out <= 0;
        else if (enable) begin
            if (sub_sum_not) begin
                out <= out - in;
            end else begin
                out <= out + in;
            end
        end
    end
endmodule
```

برای شمارنده‌ی کلاک و آدرس مموری یک شمارنده طراحی می‌کنیم که عددی به عنوان باقی‌مانده می‌گیرد و در زمان شمارش، عدد را بر پیمانه‌ی آن عدد حساب می‌کند. همچنین برای این این شمارنده یک سیگنال enable قرار می‌دهیم که مشخص می‌کند که آیا باید روبه بالا بشمارد یا عدد قبلی را خروجی دهد. همچنین یک سیگنال mod enable برای آن قرار می‌دهیم که مشخص می‌کند که آیا باید زمان شمردن بالا باقی مانده نیز بگیرد یا خیر. کد این قطعه به صورت زیر است.

```
module CounterWithMod #(parameter WIDTH = 8, parameter MOD = 100) (
    input wire clk,
    input wire clear,
    input wire enable,
    input wire mod_enable,
    output reg [WIDTH-1:0] out
);
    always @(posedge clk or posedge clear) begin
        if (clear)
            out <= 0;
        else if (enable) begin
            if (mod_enable)
                out <= (out + 1) % MOD;
            else
                out <= out + 1;
        end
    end
endmodule
```

حال برای خروجی دادن با توجه به opcode ورودی کاربر نیاز به یک mux ۸ به ۱ داریم. کد این مالتیپلکسر در زیر آمده است:

```
module Mux8To1 #(parameter WIDTH = 8) (
    input wire [WIDTH-1:0] a,
    input wire [WIDTH-1:0] b,
```

```
    input wire [WIDTH-1:0] c,
    input wire [WIDTH-1:0] d,
    input wire [WIDTH-1:0] e,
    input wire [WIDTH-1:0] f,
    input wire [WIDTH-1:0] g,
    input wire [WIDTH-1:0] h,
    input wire [2:0] select,
    output reg [WIDTH-1:0] out
);
    always @(*) begin
        case(select)
            3'b000: out = a;
            3'b001: out = b;
            3'b010: out = c;
            3'b011: out = d;
            3'b100: out = e;
            3'b101: out = f;
            3'b110: out = g;
            3'b111: out = h;
        endcase
    end
endmodule
```

در نهایت باید ماژولی طراحی کنیم که با توجه به دمای یک دقیقه‌ی اخیر و set point وضعیت فن و خنک کننده را مشخص کنید. در ابتدا برای تشخیص وضعیت به یک قدر مطلق گیر و تشخیص دهده‌ی بودن در یک بازه نیاز داریم. ماژول قدر مطلق‌گیر به صورت زیر است:

```
module Abs #(parameter WIDTH = 8) (
    input wire [WIDTH-1:0] in,
    output wire [WIDTH-1:0] out
);
    assign out = in[WIDTH-1] ? -in : in;
endmodule
```

حال ماژولی طراحی می‌کنیم که مشخص کنید که آیا عددی بین دو عدد دیگر است یا خیر. برای این کار ابتدا boundry بازه را سورت می‌کنیم و با دو مقایسه جواب را بدست می‌آوریم. دقت کنید که برای اینکه مقایسه در اعداد صحیح درست انجام بگیرد تمامی سیم‌ها را signed تعریف می‌کنیم.

```
module BetweenChecker #(parameter WIDTH = 8) (
    input wire signed [WIDTH-1:0] a,
    input wire signed [WIDTH-1:0] b,
    input wire signed [WIDTH-1:0] target,
    output wire result
);
    wire signed [WIDTH-1:0] lower = a > b ? b : a;
    wire signed [WIDTH-1:0] upper = a > b ? a : b;
    assign result = target > lower & target < upper;
endmodule
```

حال ماژول مشخص کننده‌ی وضعیت خنک کننده و فن را طراحی می‌کنیم.

```
`include "Abs.v"
`include "BetweenChecker.v"

module FanStatusResultFinder (
    input wire signed [7:0] sensor1_60_average,
    input wire signed [7:0] sensor2_60_average,
    input wire signed [7:0] set_point,
    output reg [1:0] result
);
    // Get the abs of diffrence
    wire [7:0] abs_diff;
    Abs #(8) abs(sensor2_60_average - sensor1_60_average, abs_diff);

    // Check if set point is in between
    wire set_point_between_averages;
    BetweenChecker #(8) between_checker(sensor1_60_average,
    sensor2_60_average, set_point, set_point_between_averages);

    always @(*) begin
        if (sensor2_60_average < set_point)
            result = 0;
        else if (sensor2_60_average > set_point & sensor1_60_average >
    set_point)
            result = 1;
        else if (set_point_between_averages) begin
            if (abs_diff <= 3)
                result = 2;
            else
                result = 3;
        end
    end
endmodule
```

در نهایت به کمک این قطعات کل datapath را طراحی می‌کنیم. ورودی و خروجی‌های آن به صورت زیر هستند:

- reset_clock_counter: مود باقی مانده‌گیر شمارنده‌ی کلاک را فعال می‌کند.
- reset_sum_registers: رجیسترهایی که جمع را نگه داری می‌کند را صفر می‌کند. در زمانی استفاده می‌شود که آپکد برابر ۰ باشد.
- new_temp_ready: زمانی یک می‌شود که نیاز به گرفتن ورودی دمای جدید هستیم.
- sensor1_in: دمایی که باید با رجیسترهای سنسور یک جمع یا تفریق شود.
- sensor2_in: دمایی که باید با رجیسترهای سنسور دو جمع یا تفریق شود.
- sensor1_last_load: مشخص می‌کند که آیا باید ورودی در رجیستر آخرین دمای سنسور اول ذخیره شود یا خیر.
- sensor2_last_load: مشخص می‌کند که آیا باید ورودی در رجیستر آخرین دمای سنسور دوم ذخیره شود یا خیر.

- سیگنال‌های کنترلی مربوط به میانگین: برای هر سنسور سه SensorSumHolder استفاده کردم. هر کدام از این سنسورها دو ورودی دارند که باید توسط CU مشخص شوند. یکی فعال بودن آنها هست و یکی مود بین تفریق و جمع. هر سنسور سه رجیستر دارد پس یک آرایه‌ی بیتی به طول ۳ برای هر سنسور در نظر گرفته شده است.
- average_select: عدد داده شده نشان می‌دهد که کدام یک از رجیسترهای مربوط به میانگین دما باید در average_output ظاهر شود.
- memory_address_count_up: در صورتی که سیگنال یک باشد شمارنده آدرس مموری یکی زیاد می‌شود.
- memory_select: در صورتی که یک باشد به محلی از حافظه دسترسی پیدا می‌کنیم که برای سنسور دو است. در غیر این صورت برای سنسور یک انتخاب می‌شود.
- set_point: مستقیم از خود قطعه می‌آید.
- fan_status_result: حالت‌های خروجی در زیر آمده است:
  ۱. فن و خنک کننده خاموش
  ۲. فن دور تند و خنک کننده روشن
  ۳. فن روشن با دور کم خنک کننده خاموش
  ۴. فن روشن با دور تند خنک کننده خاموش

```
`include "CounterWithMod.v"
`include "dff_load.v"
`include "SensorSumHolder.v"
`include "Mux8To1.v"
`include "FanStatusFinder.v"

module Datapath #(parameter CLOCK_FREQUENCY = 10000) (
    input wire clk,
    input wire reset,
    input wire reset_clock_counter,
    input wire reset_sum_registers,
    output wire new_temp_ready,
    // Sum saver registers
    input wire [7:0] sensor1_in,
    input wire [7:0] sensor2_in,
    input wire sensor1_last_load,
    input wire sensor2_last_load,
    // Each bit of these inputs will enable one of the chips (so sevral
   chips can be enabled together)
    input wire [2:0] sensor1_average_enable,
    input wire [2:0] sensor2_average_enable,
    input wire [2:0] sensor1_average_mode,
    input wire [2:0] sensor2_average_mode,
    // We need the raw ouput of these
    input wire [2:0] average_select,
    output wire signed [7:0] average_output,
    // Memory stuff
```

```
    input wire memory_address_count_up,
    input wire memory_select,
    input wire [1:0] memory_old_select, // 0: 0 / 1: -20 / 2: -40 / 3: -60
    output wire [7:0] memory_address,
    // Fan controller stuff
    input wire signed [7:0] set_point,
    output wire [1:0] fan_status_result
);
    // Sign extend inputs
    wire [15:0] sensor1_in_sign_extend = {{8{sensor1_in[7]}}, sensor1_in[7:0]};
    wire [15:0] sensor2_in_sign_extend = {{8{sensor2_in[7]}}, sensor2_in[7:0]};

    // Sum saver registers
    wire signed [7:0] sensor1_last_out, sensor2_last_out;
    DFFWithLoad #(.WIDTH(8)) sensor1_last(clk, reset | reset_sum_registers, sensor1_last_load, sensor1_in, sensor1_last_out);
    DFFWithLoad #(.WIDTH(8)) sensor2_last(clk, reset | reset_sum_registers, sensor2_last_load, sensor2_in, sensor2_last_out);
    wire signed [15:0] sensor1_20_out, sensor2_20_out, sensor1_40_out, sensor2_40_out, sensor1_60_out, sensor2_60_out;
    SensorSumHolder #(.WIDTH(16)) sensor1_20(.clk(clk), .reset(reset | reset_sum_registers), .enable(sensor1_average_enable[0]), .sub_sum_not(sensor1_average_mode[0]), .in(sensor1_in_sign_extend), .out(sensor1_20_out));
    SensorSumHolder #(.WIDTH(16)) sensor2_20(.clk(clk), .reset(reset | reset_sum_registers), .enable(sensor2_average_enable[0]), .sub_sum_not(sensor2_average_mode[0]), .in(sensor2_in_sign_extend), .out(sensor2_20_out));
    SensorSumHolder #(.WIDTH(16)) sensor1_40(.clk(clk), .reset(reset | reset_sum_registers), .enable(sensor1_average_enable[1]), .sub_sum_not(sensor1_average_mode[1]), .in(sensor1_in_sign_extend), .out(sensor1_40_out));
    SensorSumHolder #(.WIDTH(16)) sensor2_40(.clk(clk), .reset(reset | reset_sum_registers), .enable(sensor2_average_enable[1]), .sub_sum_not(sensor2_average_mode[1]), .in(sensor2_in_sign_extend), .out(sensor2_40_out));
    SensorSumHolder #(.WIDTH(16)) sensor1_60(.clk(clk), .reset(reset | reset_sum_registers), .enable(sensor1_average_enable[2]), .sub_sum_not(sensor1_average_mode[2]), .in(sensor1_in_sign_extend), .out(sensor1_60_out));
    SensorSumHolder #(.WIDTH(16)) sensor2_60(.clk(clk), .reset(reset | reset_sum_registers), .enable(sensor2_average_enable[2]), .sub_sum_not(sensor2_average_mode[2]), .in(sensor2_in_sign_extend), .
```

```
    out(sensor2_60_out));

    // Averages
    wire signed [7:0] sensor1_20_average = sensor1_20_out / 20;
    wire signed [7:0] sensor2_20_average = sensor2_20_out / 20;
    wire signed [7:0] sensor1_40_average = sensor1_40_out / 40;
    wire signed [7:0] sensor2_40_average = sensor2_40_out / 40;
    wire signed [7:0] sensor1_60_average = sensor1_60_out / 60;
    wire signed [7:0] sensor2_60_average = sensor2_60_out / 60;
    Mux8To1 average_result_mux(
        .a(sensor1_last_out),
        .b(sensor2_last_out),
        .c(sensor1_20_average),
        .d(sensor2_20_average),
        .e(sensor1_40_average),
        .f(sensor2_40_average),
        .g(sensor1_60_average),
        .h(sensor2_60_average),
        .select(average_select),
        .out(average_output)
    );

    // Cloak counter
    wire [31:0] clock_counter;
    CounterWithMod #(.WIDTH(32), .MOD(CLOCK_FREQUENCY))
    clock_counter_part(.clk(clk), .clear(reset), .enable(1'b1), .
    mod_enable(reset_clock_counter), .out(clock_counter));
    assign new_temp_ready = clock_counter >= CLOCK_FREQUENCY;

    // Mem address
    wire [7:0] mem_address;
    CounterWithMod #(.WIDTH(8), .MOD(100)) mem_address_counter(.clk(clk)
    , .clear(reset), .enable(memory_address_count_up), .mod_enable(1'b1)
    , .out(mem_address));
    wire [31:0] offset_memory_address = (mem_address + memory_old_select
     * 80) % 100;
    assign memory_address = memory_select ? offset_memory_address + 100
    : offset_memory_address;

    // Fan result
    FanStatusResultFinder fan_result_finder(.sensor1_60_average(
    sensor1_60_average), .sensor2_60_average(sensor2_60_average), .
    set_point(set_point), .result(fan_status_result));
endmodule
```

حال با توجه به نمودار حالت control unit را طراحی می‌کنیم. سعی کرده‌ام که با کامنت توضحاتی درباره‌ی قسمت‌های مختلف بدهم.

```
module ControlUnit(
```

```
    input wire clk,
    input wire reset,
    ////////////////////
    // Main Controller //
    ////////////////////
    input wire [3:0] microupcode,
    input wire signed [7:0] sensor1_temp,
    input wire signed [7:0] sensor2_temp,
    output reg [1:0] fan_status,
    output reg cooler_status,
    //////////////
    // Datapath //
    //////////////
    output reg reset_clock_counter,
    output reg reset_sum_registers,
    input wire new_temp_ready,
    // Sum saver registers
    output reg [7:0] sensor1_in,
    output reg [7:0] sensor2_in,
    output reg sensor1_last_load,
    output reg sensor2_last_load,
    // Each bit of these inputs will enable one of the chips (so sevral chips can be enabled together)
    output reg [2:0] sensor1_average_enable,
    output reg [2:0] sensor2_average_enable,
    output reg [2:0] sensor1_average_mode,
    output reg [2:0] sensor2_average_mode,
    // We need the raw ouput of these
    output reg [2:0] average_select,
    // Memory stuff
    output reg datapath_memory_address_count_up,
    output reg datapath_memory_select,
    output reg [1:0] memory_old_select,
    // Fan controller stuff
    input wire [1:0] fan_status_result,
    /////////
    // RAM //
    /////////
    output reg ram_write_enable,
    inout [7:0] ram_inout,
    output reg ram_reset
);
    // State stuff
    localparam [3:0] STATE_START = 0, STATE_STORE_SENSOR1 = 1, STATE_STORE_SENSOR2 = 2,
        STATE_UPDATE_SENSOR1_REG20 = 3, STATE_UPDATE_SENSOR2_REG20 = 4,
        STATE_UPDATE_SENSOR1_REG40 = 5, STATE_UPDATE_SENSOR2_REG40 = 6,
        STATE_UPDATE_SENSOR1_REG60 = 7, STATE_UPDATE_SENSOR2_REG60 = 8,
```

```
48          STATE_CHECK_UPCODE = 9, STATE_UPDATE_COOLER = 10,
    STATE_AFTER_SENSOR_UPDATE = 11;
49      reg [3:0] state, next_state;
50
51      // Ram
52      reg [7:0] ram_output;
53      assign ram_inout = ram_write_enable ? ram_output : 8'bz;
54
55      // Combinational part
56      always @(*) begin
57          // Reset status registers
58          reset_clock_counter = 0;
59          reset_sum_registers = 0;
60          ram_write_enable = 0;
61          ram_reset = 0;
62          sensor1_last_load = 0;
63          sensor2_last_load = 0;
64          sensor1_average_enable = 0;
65          sensor2_average_enable = 0;
66          datapath_memory_address_count_up = 0;
67          // Check state
68          case(state)
69              STATE_START: begin
70                  if (new_temp_ready) begin
71                      next_state = STATE_STORE_SENSOR1;
72                      reset_clock_counter = 1;
73                  end else begin
74                      next_state = STATE_CHECK_UPCODE;
75                  end
76              end
77              STATE_STORE_SENSOR1: begin
78                  datapath_memory_select = 0; // first sensor
79                  memory_old_select = 0; // current index
80                  ram_output = sensor1_temp; // save temp 1
81                  sensor1_in = sensor1_temp;
82                  ram_write_enable = 1; // write to ram
83                  sensor1_last_load = 1;
84                  sensor1_average_enable = 3'b111; // add to all
85                  sensor1_average_mode = 3'b000; // mode add
86                  next_state = STATE_STORE_SENSOR2;
87              end
88              STATE_STORE_SENSOR2: begin
89                  datapath_memory_select = 1; // second sensor
90                  memory_old_select = 0; // current index
91                  ram_output = sensor2_temp; // save temp 2
92                  sensor2_in = sensor2_temp;
93                  ram_write_enable = 1; // write to ram
94                  sensor2_last_load = 1;
```

```
                sensor2_average_enable = 3'b111; // add to all
                sensor2_average_mode = 3'b000; // mode add
                next_state = STATE_UPDATE_SENSOR1_REG20;
            end
            STATE_UPDATE_SENSOR1_REG20: begin
                datapath_memory_select = 0; // first sensor
                memory_old_select = 1; // -20
                // We edit the reg in next state
                next_state = STATE_UPDATE_SENSOR2_REG20;
            end
            STATE_UPDATE_SENSOR2_REG20: begin
                // Update the sensor from previous state
                sensor1_average_enable = 3'b001; // sub from first
                sensor1_average_mode = 3'b001; // mode sub
                sensor1_in = ram_inout;
                // Update ram
                datapath_memory_select = 1; // second sensor
                memory_old_select = 1; // -20
                next_state = STATE_UPDATE_SENSOR1_REG40;
            end
            STATE_UPDATE_SENSOR1_REG40: begin
                // Update the sensor
                sensor2_average_enable = 3'b001; // sub from first
                sensor2_average_mode = 3'b001; // mode sub
                sensor2_in = ram_inout;
                // Update ram
                datapath_memory_select = 0; // first sensor
                memory_old_select = 2; // -40
                next_state = STATE_UPDATE_SENSOR2_REG40;
            end
            STATE_UPDATE_SENSOR2_REG40: begin
                // Update the sensor
                sensor1_average_enable = 3'b010;
                sensor1_average_mode = 3'b010; // mode sub
                sensor1_in = ram_inout;
                // Update ram
                datapath_memory_select = 1; // second sensor
                memory_old_select = 2; // -40
                next_state = STATE_UPDATE_SENSOR1_REG60;
            end
            STATE_UPDATE_SENSOR1_REG60: begin
                // Update the sensor
                sensor2_average_enable = 3'b010;
                sensor2_average_mode = 3'b010; // mode sub
                sensor2_in = ram_inout;
                // Update ram
                datapath_memory_select = 0; // first sensor
                memory_old_select = 3; // -60
```

```
                next_state = STATE_UPDATE_SENSOR2_REG60;
            end
            STATE_UPDATE_SENSOR2_REG60: begin
                // Update the sensor
                sensor1_average_enable = 3'b100;
                sensor1_average_mode = 3'b100; // mode sub
                sensor1_in = ram_inout;
                // Update ram
                datapath_memory_select = 1; // second sensor
                memory_old_select = 3; // -60
                next_state = STATE_AFTER_SENSOR_UPDATE;
            end
            STATE_AFTER_SENSOR_UPDATE: begin
                // Update the sensor
                sensor2_average_enable = 3'b100;
                sensor2_average_mode = 3'b100; // mode sub
                sensor2_in = ram_inout;
                datapath_memory_address_count_up = 1; // increase memory
    address
                // Done
                next_state = STATE_CHECK_UPCODE;
            end
            STATE_CHECK_UPCODE: begin
                case (microupcode)
                4'b0000: begin
                    reset_sum_registers = 1;
                    ram_reset = 1;
                end
                4'b1000: average_select = 0;
                4'b1100: average_select = 1;
                4'b1001: average_select = 2;
                4'b1010: average_select = 4;
                4'b1011: average_select = 6;
                4'b1101: average_select = 3;
                4'b1110: average_select = 5;
                4'b1111: average_select = 7;
                endcase
                next_state = microupcode == 0 ? STATE_START :
    STATE_UPDATE_COOLER;
            end
            STATE_UPDATE_COOLER: begin
                // This is combinational so we can do this
                case (fan_status_result)
                2'b00: begin
                    fan_status = 0;
                    cooler_status = 0;
                end
                2'b01: begin
```

```
                    fan_status = 2;
                    cooler_status = 1;
                end
                2'b10: begin
                    fan_status = 1;
                    cooler_status = 0;
                end
                2'b11: begin
                    fan_status = 2;
                    cooler_status = 0;
                end
                endcase
                next_state = STATE_START;
            end
        endcase
    end

    // Sequential part
    always @(posedge clk or posedge reset) begin
        if (reset)
            state <= STATE_START;
        else
            state <= next_state;
    end
endmodule
```

حال مموری را طراحی می‌کنیم:

```
module Memory (
    input wire clk,
    input wire reset,
    input wire [7:0] addr, // 8 bit address to access at last 256
    input wire write_enable,
    inout [7:0] data
);
    reg [7:0] mem[0:199];
    reg [7:0] read_data;
    integer i;

    assign data = write_enable ? 8'bz : read_data;

    always @(posedge clk or posedge reset) begin
        if (reset)
            for (i = 0; i < 200; i = i + 1)
                mem[i] <= 0;
        else if (write_enable)
            mem[addr] <= data;
        else
            read_data <= mem[addr];
```

```
    end
endmodule
```

در نهایت نیز تمامی قطعات را به هم می‌چسبانیم.

```
`include "Memory.v"
`include "Datapath.v"
`include "ControlUnit.v"

module FanController #(parameter CLOCK_FREQUENCY = 10000) (
    input wire clk,
    input wire reset,
    input wire [3:0] microupcode,
    input wire signed [7:0] set_point,
    input wire signed [7:0] sensor1_temp,
    input wire signed [7:0] sensor2_temp,
    output wire [1:0] fan_status,
    output wire cooler_status,
    output wire [7:0] user_output
);
    wire [7:0] mem_address, mem_inout;
    wire mem_wire_enable, mem_reset;
    Memory ram (.clk(clk), .reset(reset | mem_reset), .addr(mem_address)
    , .write_enable(mem_wire_enable), .data(mem_inout));

    wire reset_clock_counter, reset_sum_registers, new_temp_ready,
    sensor1_last_load, sensor2_last_load, memory_address_count_up,
    memory_select;
    wire [7:0] sensor1_in, sensor2_in;
    wire [2:0] sensor1_average_enable, sensor2_average_enable,
    sensor1_average_mode, sensor2_average_mode, average_select;
    wire [1:0] fan_status_result, memory_old_select;
    Datapath #(CLOCK_FREQUENCY) datapath(
        .clk(clk),
        .reset(reset),
        .reset_clock_counter(reset_clock_counter),
        .reset_sum_registers(reset_sum_registers),
        .new_temp_ready(new_temp_ready),
        .sensor1_in(sensor1_in),
        .sensor2_in(sensor2_in),
        .sensor1_last_load(sensor1_last_load),
        .sensor2_last_load(sensor2_last_load),
        .sensor1_average_enable(sensor1_average_enable),
        .sensor2_average_enable(sensor2_average_enable),
        .sensor1_average_mode(sensor1_average_mode),
        .sensor2_average_mode(sensor2_average_mode),
        .average_select(average_select),
        .average_output(user_output),
        .memory_address_count_up(memory_address_count_up),
```

```
        .memory_select(memory_select),
        .memory_old_select(memory_old_select),
        .memory_address(mem_address),
        .set_point(set_point),
        .fan_status_result(fan_status_result)
    );
    ControlUnit cu(
        .clk(clk),
        .reset(reset),
        .microupcode(microupcode),
        .sensor1_temp(sensor1_temp),
        .sensor2_temp(sensor2_temp),
        .fan_status(fan_status),
        .cooler_status(cooler_status),
        .reset_clock_counter(reset_clock_counter),
        .reset_sum_registers(reset_sum_registers),
        .new_temp_ready(new_temp_ready),
        .sensor1_in(sensor1_in),
        .sensor2_in(sensor2_in),
        .sensor1_last_load(sensor1_last_load),
        .sensor2_last_load(sensor2_last_load),
        .sensor1_average_enable(sensor1_average_enable),
        .sensor2_average_enable(sensor2_average_enable),
        .sensor1_average_mode(sensor1_average_mode),
        .sensor2_average_mode(sensor2_average_mode),
        .average_select(average_select),
        .datapath_memory_address_count_up(memory_address_count_up),
        .datapath_memory_select(memory_select),
        .memory_old_select(memory_old_select),
        .fan_status_result(fan_status_result),
        .ram_write_enable(mem_wire_enable),
        .ram_inout(mem_inout),
        .ram_reset(mem_reset)
    );
endmodule
```

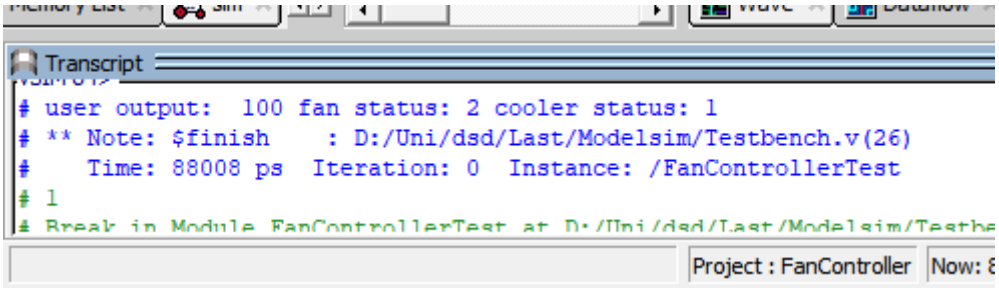
تست مدار

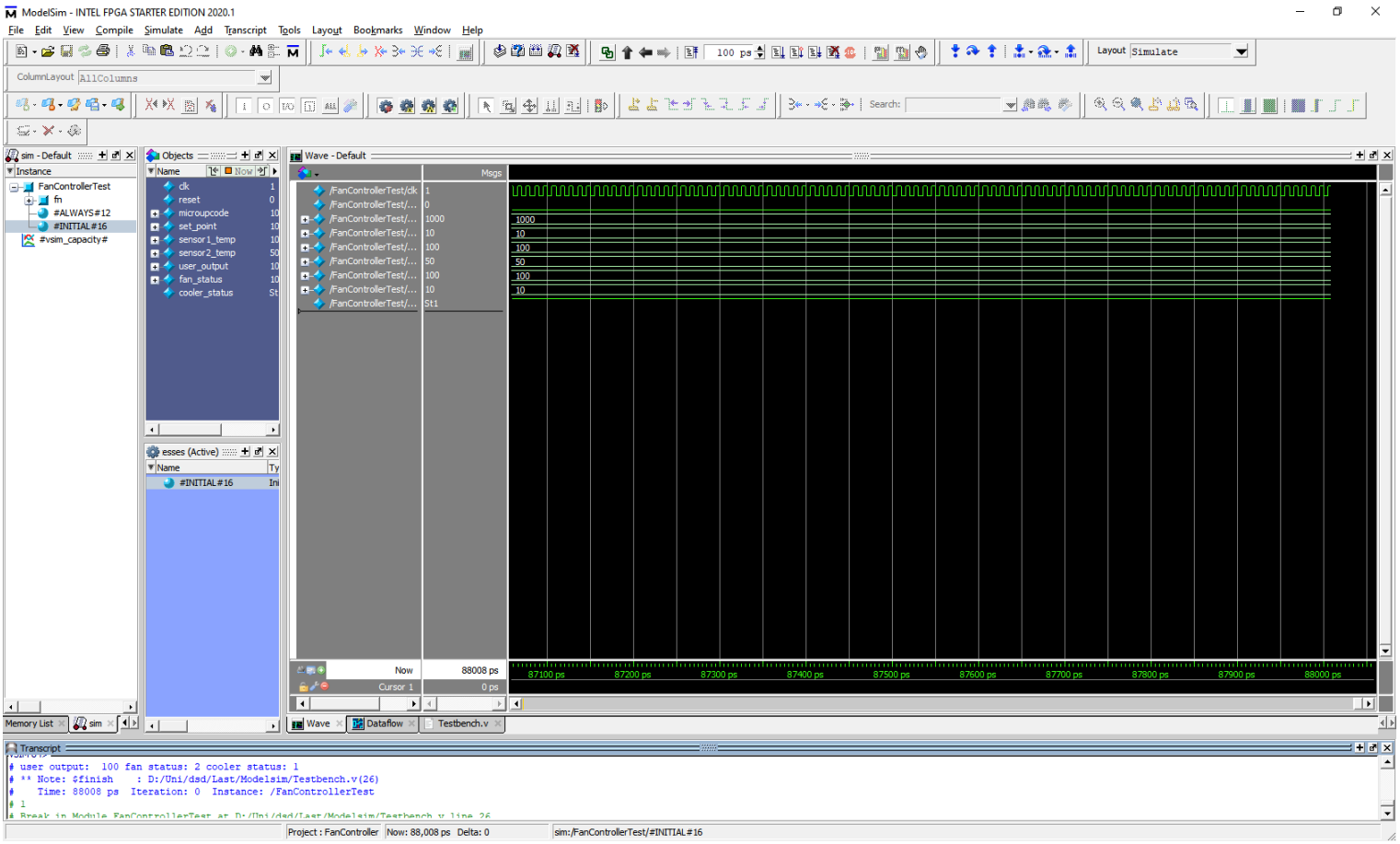
حال مدار را تست می‌کنیم. برای تست دمای ورودی را ثابت نگه می‌داریم و صبر می‌کنیم که یک دقیقه از کار مدار بگذرد. سپس خروجی مدار را بررسی می‌کنیم. تمامی رجیسترها باید مقدار دماها را داشته باشند و فن به درستی تنظیم شده باشد. کد تست در زیر آمده است:

```
`include "FanController.v"

module FanControllerTest;
    reg clk, reset;
    reg [3:0] microupcode;
    reg signed [7:0] set_point, sensor1_temp, sensor2_temp;
    wire signed [7:0] user_output;
    wire [1:0] fan_status;
```

```
    wire cooler_status;

    initial clk = 0;
    always #5 clk = ~clk;

    FanController #(100) fn(clk, reset, microupcode, set_point,
   sensor1_temp, sensor2_temp, fan_status, cooler_status, user_output);

    initial begin
        microupcode = 4'b1000;
        reset = 1;
        #8;
        reset = 0;
        sensor1_temp = 100;
        sensor2_temp = 50;
        set_point = 10;
        #(100 * 11 * 80);
        $display("user output: %d fan status: %d cooler status: %d",
   user_output, fan_status, cooler_status);
        $finish;
    end
endmodule
```

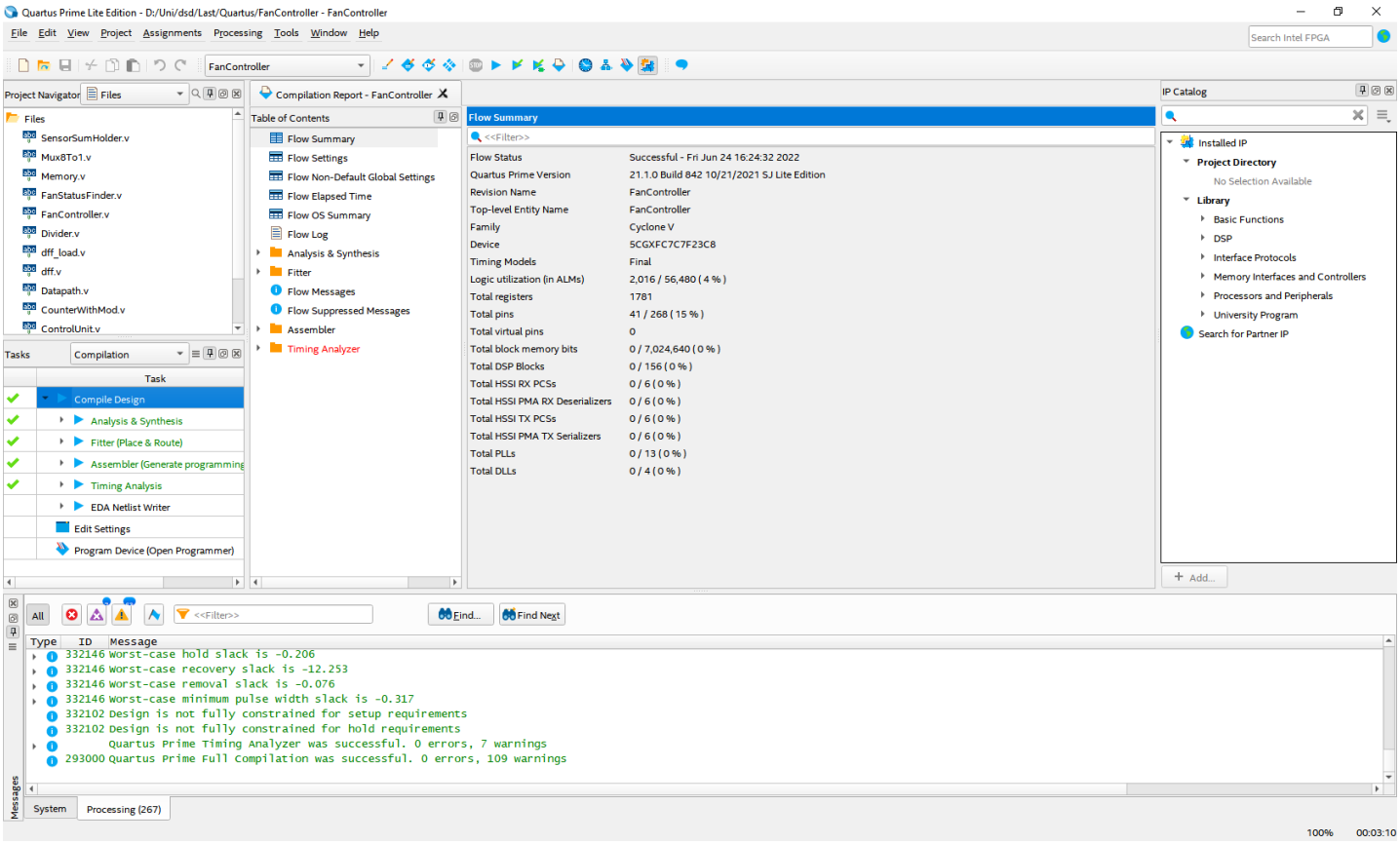
بعد از تست لاگ‌های modelsim را بررسی می‌کنیم که آیا جواب درست به ما داده شده است یا خیر.

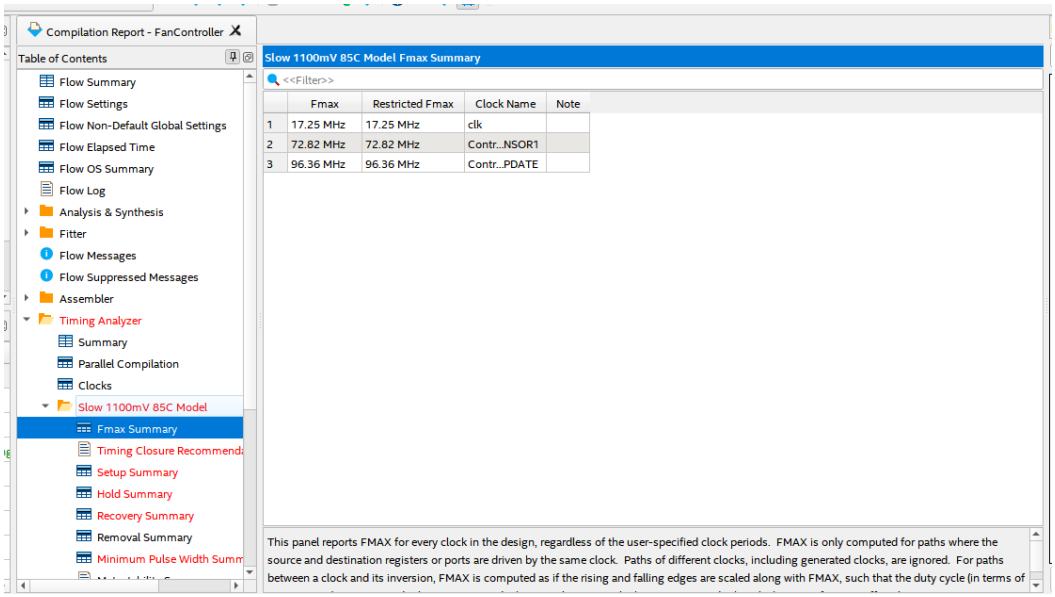




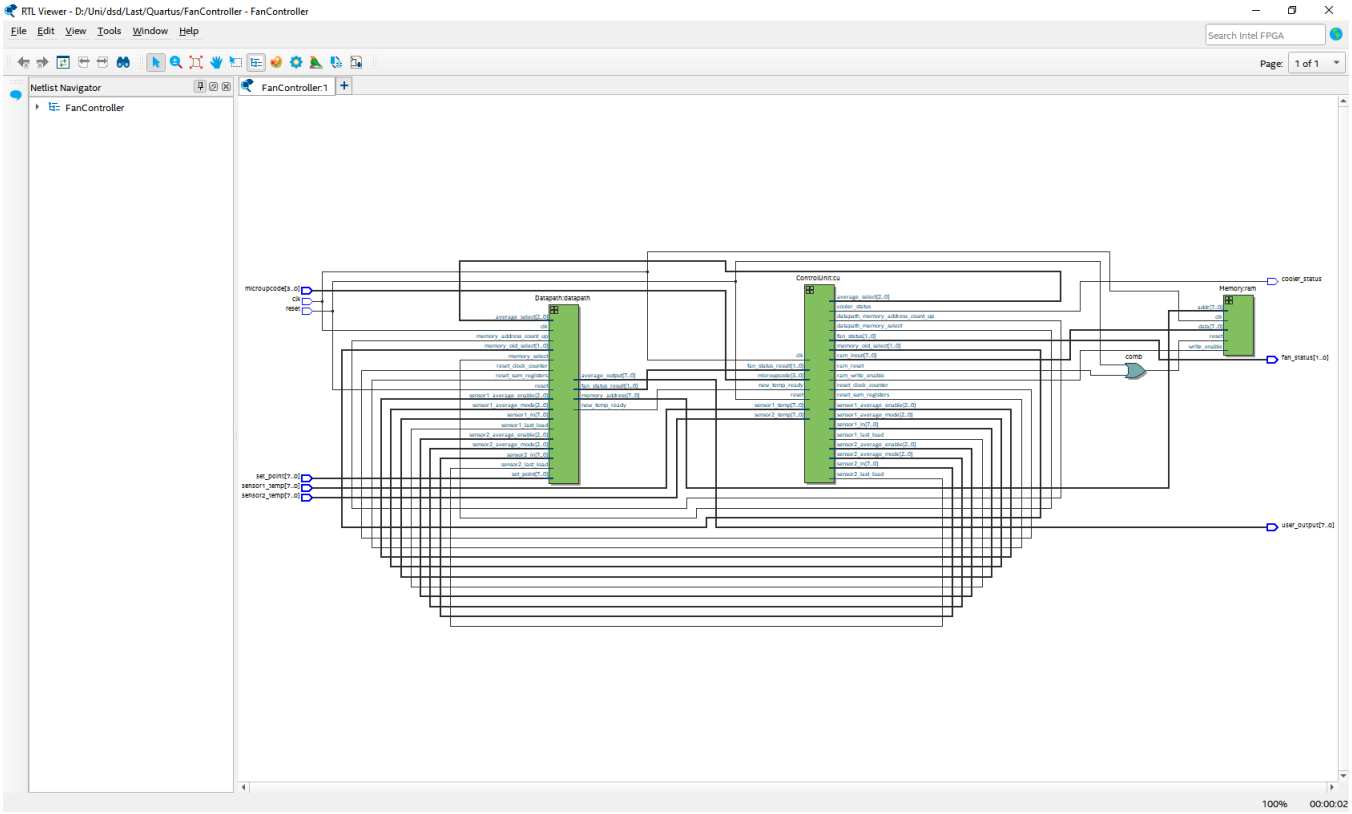
سنتز

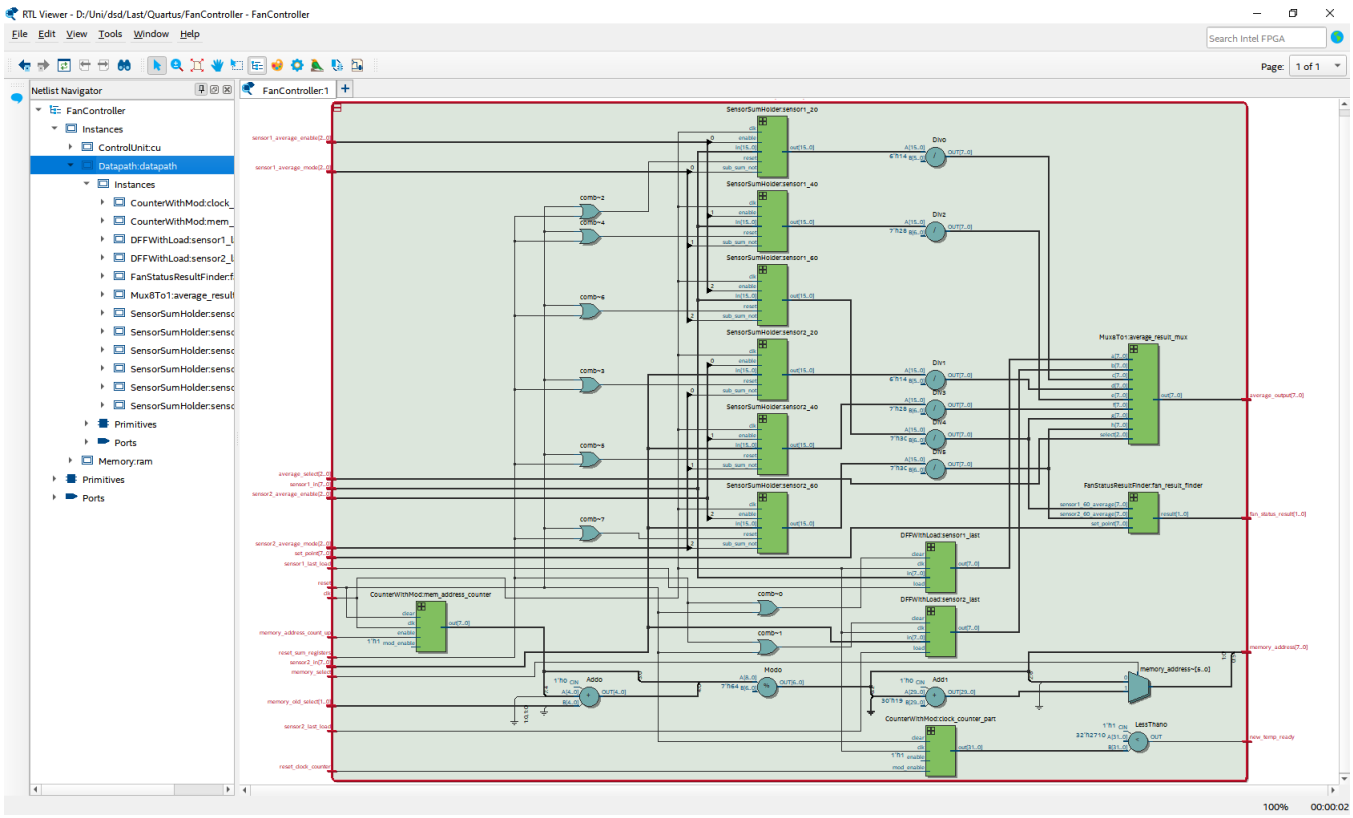
من سنتز را به کمک نرم افزار Quartus انجام دادم. نکته‌ای که قبل از سنتز باید توجه کرد این است که باید تمامی خط‌های include را از اول تمامی فایل‌ها برداشت چرا که کوارتوس ارور می‌دهد. بعد از این کار، سنتز را انجام می‌دهیم و مشاهده می‌شود که به درستی سنتز انجام می‌شود. حتی می‌توان کلاک پیشنهادی مدار را نیز پیدا کرد.

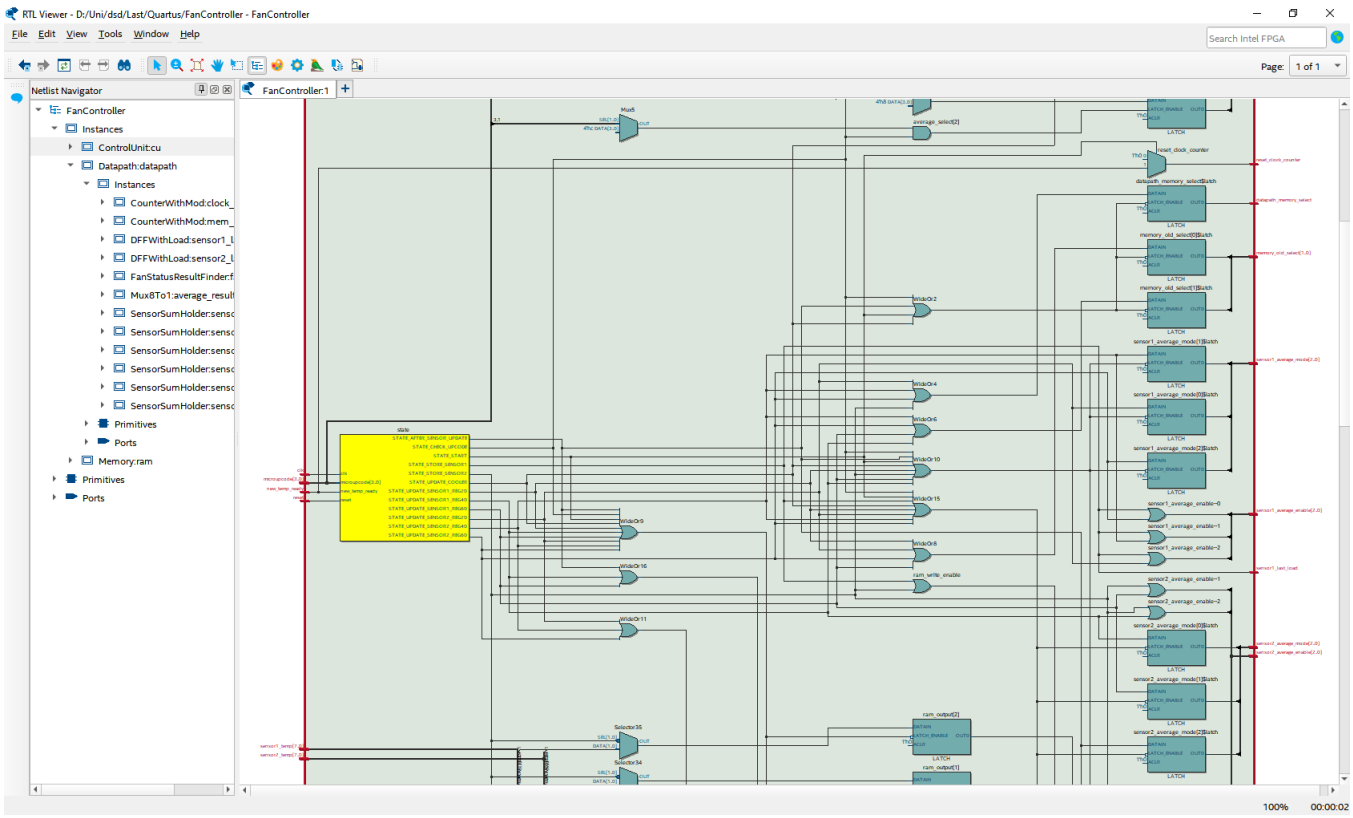

Quartus Prime Lite Edition - D:/Uni/dsd/Last/Quartus/FanController - FanController
File Edit View Project Assignments Processing Tools Window Help
Search Intel FPGA
FanController
Project Navigator
Files
SensorSumHolder.v
Mux8To1.v
Memory.v
FanStatusFinder.v
FanController.v
Divider.v
dff_load.v
dff.v
Datapath.v
CounterWithMod.v
ControlUnit.v
Tasks
Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming
Timing Analysis
EDA Netlist Writer
Edit Settings
Program Device (Open Programmer)
Compilation Report - FanController
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Settings
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Flow Messages
Flow Suppressed Messages
Assembler
Timing Analyzer
Flow Summary
<<Filter>>
Flow Status Successful - Fri Jun 24 16:24:32 2022
Quartus Prime Version 21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name FanController
Top-level Entity Name FanController
Family Cyclone V
Device 5CGXFC7C7F23C8
Timing Models Final
Logic utilization (in ALMs) 2,016 / 56,480 (4 %)
Total registers 1781
Total pins 41 / 268 (15 %)
Total virtual pins 0
Total block memory bits 0 / 7,024,640 (0 %)
Total DSP Blocks 0 / 156 (0 %)
Total HSSI RX PCSs 0 / 6 (0 %)
Total HSSI PMA RX Deserializers 0 / 6 (0 %)
Total HSSI TX PCSs 0 / 6 (0 %)
Total HSSI PMA TX Serializers 0 / 6 (0 %)
Total PLLs 0 / 13 (0 %)
Total DLLs 0 / 4 (0 %)
IP Catalog
Installed IP
Project Directory
No Selection Available
Library
Basic Functions
DSP
Interface Protocols
Memory Interfaces and Controllers
Processors and Peripherals
University Program
Search for Partner IP
Add...
All
<<Filter>>
Find...
Find Next
Type ID Message
332146 Worst-case hold slack is -0.206
332146 Worst-case recovery slack is -12.253
332146 Worst-case removal slack is -0.076
332146 Worst-case minimum pulse width slack is -0.317
332102 Design is not fully constrained for setup requirements
332102 Design is not fully constrained for hold requirements
Quartus Prime Timing Analyzer was successful. 0 errors, 7 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 109 warnings
Messages
System
Processing (267)
100%
00:03:10


Compilation Report - FanController
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Settings
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Flow Messages
Flow Suppressed Messages
Assembler
Timing Analyzer
Summary
Parallel Compilation
Clocks
Slow 1100mV 85C Model
Fmax Summary
Timing Closure Recommenda
Setup Summary
Hold Summary
Recovery Summary
Removal Summary
Minimum Pulse Width Summ
Slow 1100mV 85C Model Fmax Summary
<<Filter>>
Fmax Restricted Fmax Clock Name Note
1 17.25 MHz 17.25 MHz clk
2 72.82 MHz 72.82 MHz Contr...NSOR1
3 96.36 MHz 96.36 MHz Contr...PDATE
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of

حال برای بدست اوردن شماتیک مدار از منوی tools، Netlist Viewers، RTL-Viewer را انتخاب می‌کنیم. شماتیک برخی از قطعات مانند CU و datapath در زیر آورده شده است.




RTL Viewer - D:/Uni/dsd/Last/Quartus/FanController - FanController
File Edit View Tools Window Help
Search Intel FPGA
Page: 1 of 1
Netlist Navigator
FanController
Instances
ControlUnit:cu
Datapath:datapath
Instances
CounterWithMod:clock_
CounterWithMod:mem_
DFFWithLoad:sensor1_l
DFFWithLoad:sensor2_l
FanStatusResultFinder:f
Mux8To1:average_result
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
Primitives
Ports
Memory:ram
Primitives
Ports
FanController:1
SensorSumHolder:sensor1_20
SensorSumHolder:sensor1_40
SensorSumHolder:sensor1_60
SensorSumHolder:sensor2_20
SensorSumHolder:sensor2_40
SensorSumHolder:sensor2_60
Div0
Div1
Div2
Div3
Div4
Div5
Mux8To1:average_result_mux
FanStatusResultFinder:fan_result_finder
DFFWithLoad:sensor1_last
DFFWithLoad:sensor2_last
CounterWithMod:mem_address_counter
CounterWithMod:clock_counter_part
Add0
Add1
Mod0
LessThan0
comb~0
comb~1
comb~2
comb~3
comb~4
comb~5
comb~6
comb~7
comb~8
100%
00:00:02


RTL Viewer - D:/Uni/dsd/Last/Quartus/FanController - FanController
File Edit View Tools Window Help
Search Intel FPGA
Page: 1 of 1
Netlist Navigator
FanController
Instances
ControlUnit:cu
Datapath:datapath
Instances
CounterWithMod:clock_
CounterWithMod:mem_
DFFWithLoad:sensor1_l
DFFWithLoad:sensor2_l
FanStatusResultFinder:f
Mux8To1:average_result
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
SensorSumHolder:sensc
Primitives
Ports
Memory:ram
Primitives
Ports
FanController:1
100%
00:00:02