بسمه تعالی

در این فایل مختصری از نحوه­ی عملکرد کدهای مقاله شرح داده می­شود.

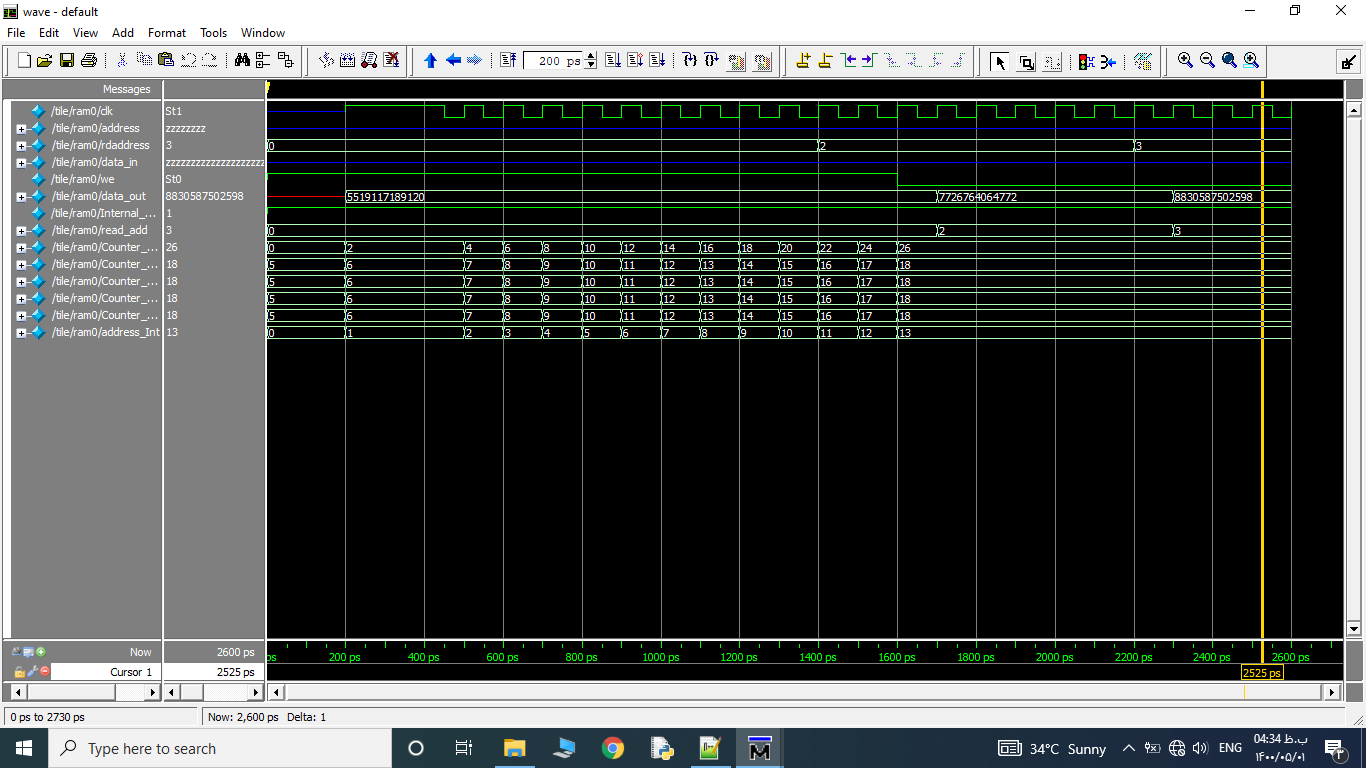
**Ram0:**

در این کد تمام عملیات با لبه­­ی بالارونده­ی clk انجام می­شود.

عملیات تغییر خانه­ی حافظه، یعنی عمل کردن کد Counter های نوشته شده که خانه­های حافظه را خط به خط پر می­کند با یک شدن we و internal-RAM اتفاق می­افتد.

تا زمانی که we برابر صفر نشود، rdaddress تغییر نگرده لذا read-add تغییر نمی­کند. این امر به معنای ثابت ماندن خروجی است. زمانی که we صفر شود rdaddress قابلیت تغییر دارد.

حال برای rdaddress در واقع current\_state بوده که از باتوجه به mux-out تغییر می­کند. Mux-out هم با توجه به انتخاب sel در قسمت mux-4-1 تغییر می­کند. بنابر این با انتخاب خط Selec می­توان خط rdaddress را کنترل نمود.

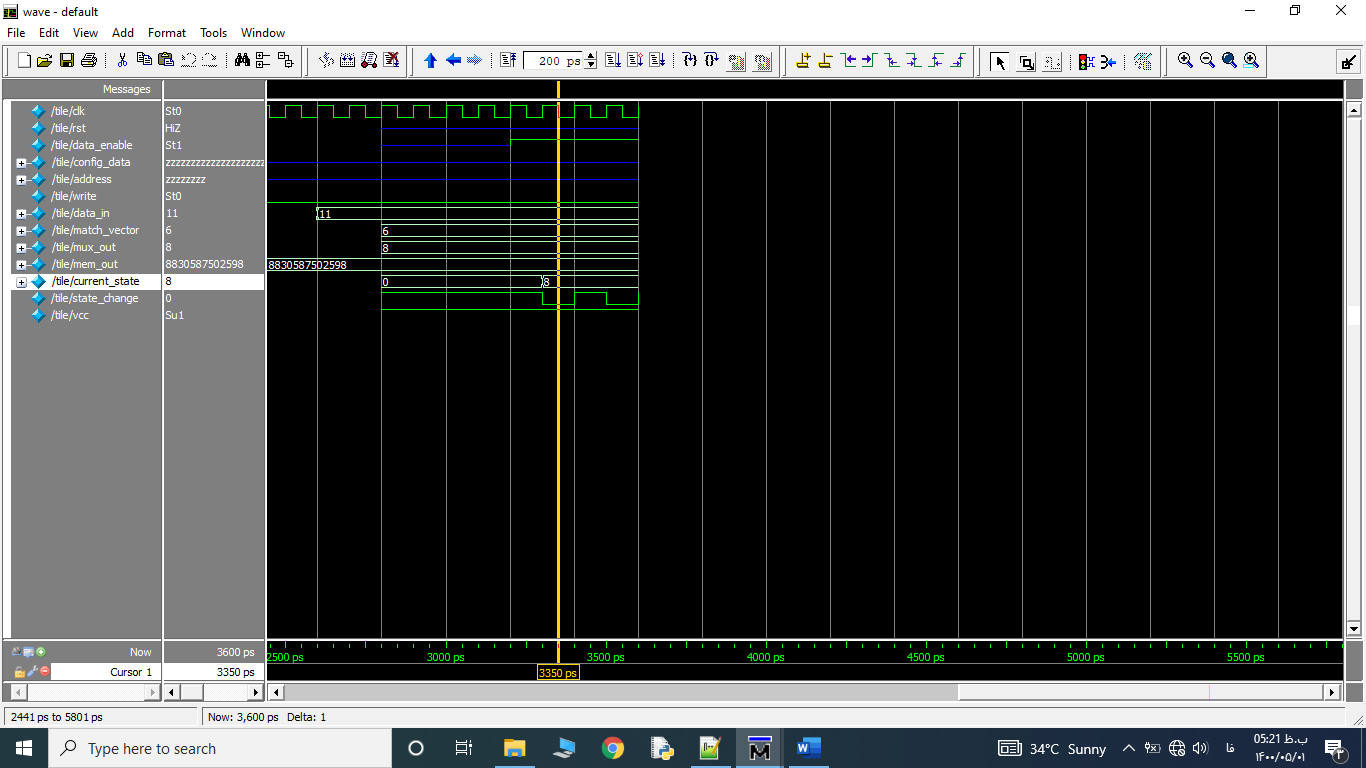


**Tile:**

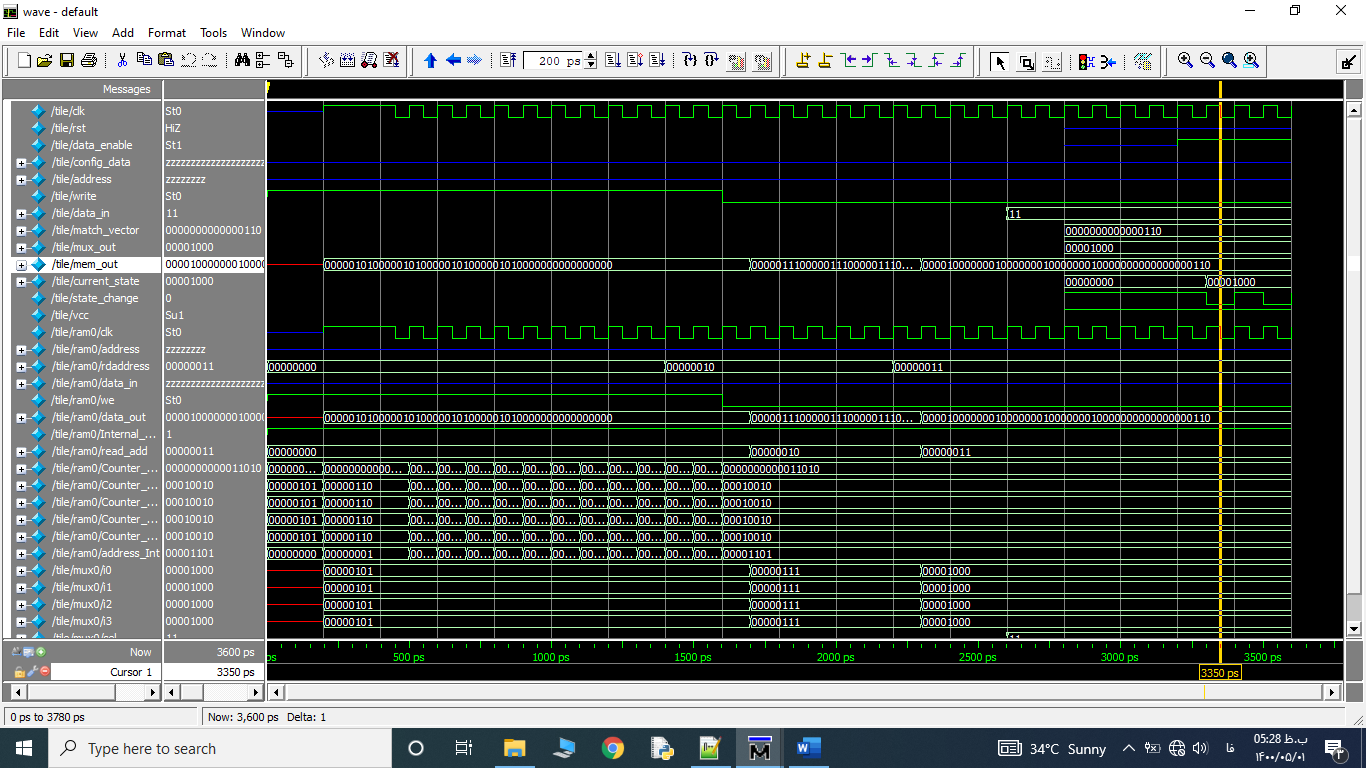
در این قسمت زمانی که rst یا write برابر 1 باشد current\_state صفر و state\_change یک می­شود. یعنی آماده می­شود برای آدرس دهی دوباره چرا که current\_state برابر صفر است.

در غیر این صورت، اگر data-enable و state\_change برابر 1 شود خروجی mux\_out که با توجه به انتخاب خط sel تعیین شده به current\_state می­رود. که current\_state در واقع همان آدرس خطی است که می­خواهیم بخوانیم. همچنین State\_change هم صفر می­شود.

و در نهایت باتوجه به خط آدرس انتخاب شده خروجی انتخاب می­شود و داده­­ی 0 تا 15 آن به match\_vector می­رود.



شکل زیر تمام شکل موج­ها را نشان می­دهد.



**کد بخش­های مختلف:**

**Mux\_4\_1:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 12:00:22 03/23/2021

// Design Name:

// Module Name: mux

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//----------------------------------------------------

//File Name: mux\_4\_1.v

//Function: a simple 4x1 mux

//Coder: Brett Brotherton (brett.brotherton@gmail.com)

//----------------------------------------------------

module mux\_4\_1(i0, i1, i2, i3, sel, mux\_out);

//-----------Input Ports---------------

input[7:0] i0, i1, i2, i3;

input[1:0] sel;

//-----------Output Ports---------------

output[7:0] mux\_out;

//------------Internal Variables--------

reg[7:0] mux\_out;

//-------------Code Starts Here---------

always @ (sel or i0 or i1 or i2 or i3)

begin : MUX

case(sel )

2'b00 : mux\_out = i0;

2'b01 : mux\_out = i1;

2'b10 : mux\_out = i2;

2'b11 : mux\_out = i3;

endcase

end

endmodule //End Of Module mux

**ram:**

//----------------------------------------------------

//File Name: ram.v

//Function: a synchronous read/write ram for the AC State machine tile

//Coder: Brett Brotherton (brett.brotherton@gmail.com)

//----------------------------------------------------

module ram( clk, address, rdaddress, data\_in, data\_out, we);

//inputs

input clk;

input[7:0] address;

input[7:0] rdaddress;

input[47:0] data\_in;

input we;

//outputs

output[47:0] data\_out;

//reg[DATA\_WIDTH-1:0] data\_out;

//internal variables

reg Internal\_RAM = 1'b1;

reg [47:0]mem[256:0];

reg [7:0] read\_add = 8'b0;

reg [15:0] Counter\_Input\_Data = 16'b0;

reg [7:0] Counter\_Input\_Addr = 8'h05;

reg [7:0] Counter\_Input\_Addr1 = 8'h05;

reg [7:0] Counter\_Input\_Addr2 = 8'h05;

reg [7:0] Counter\_Input\_Addr3 = 8'h05;

reg [7:0] address\_Int = 8'b0;

//code starts here

//mem write operation

always @ (posedge clk)

begin

if (we) begin

if (Internal\_RAM == 0) begin

mem[address] <= data\_in;

end

if (Internal\_RAM) begin

mem[address\_Int][15:0] <= Counter\_Input\_Data ;

mem[address\_Int][23:16] <= Counter\_Input\_Addr ;

mem[address\_Int][31:24] <= Counter\_Input\_Addr1 ;

mem[address\_Int][39:32] <= Counter\_Input\_Addr2 ;

mem[address\_Int][47:40] <= Counter\_Input\_Addr3 ;

Counter\_Input\_Data <= Counter\_Input\_Data + 2 ;

Counter\_Input\_Addr <= Counter\_Input\_Addr + 1 ;

Counter\_Input\_Addr1 <= Counter\_Input\_Addr1 + 1 ;

Counter\_Input\_Addr2 <= Counter\_Input\_Addr2 + 1 ;

Counter\_Input\_Addr3 <= Counter\_Input\_Addr3 + 1 ;

address\_Int <= address\_Int + 1 ;

if (address\_Int == 255) begin

Internal\_RAM <= 1'b0;

address\_Int <= 8'b0;

end

end

end

else begin

read\_add <= rdaddress;

end

end

assign data\_out = mem[read\_add];

endmodule //End of Module ram

**Tile:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 12:00:53 03/23/2021

// Design Name:

// Module Name: tilee

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//----------------------------------------------------

//File Name: tile.v

//Function: a tile for the AC state machine

//Coder: Brett Brotherton (brett.brotherton@gmail.com)

//----------------------------------------------------

module tile(clk, rst, data\_enable, config\_data, data\_in, match\_vector, address, write);

//width of data memory

parameter DATA\_WIDTH = 48;

//width of partial match vector

parameter VECTOR\_WIDTH = 16;

//width of data address

parameter ADDRESS\_WIDTH = 8;

//inputs

input clk;

input rst;

input data\_enable;

//data to be written to the tile for initialization and updates

input[DATA\_WIDTH-1:0] config\_data;

//address to write the data to

input[ADDRESS\_WIDTH-1:0] address;

//write enable

input write;

//input for AC state machine

input[1:0] data\_in;

//outputs

//partial match vector

output[VECTOR\_WIDTH-1:0] match\_vector;

//internal variables

//wire to connect mux to current\_state (output of mux is next state)

wire[7:0] mux\_out;

//wire to connect mem to mux and match vector

wire[DATA\_WIDTH-1:0] mem\_out;

//current state of AC state machine

reg[7:0] current\_state;

//makes sure only one state change per byte of data

reg state\_change;

//code begins here

//handle state transition

//change states only on rising clock edge

always @ (posedge rst or posedge write or posedge clk)

begin : INITIALIZATION

//handle reset of machine

if ((rst)||(write)) begin

//reset state and stall variables

current\_state <= 8'h00;

state\_change <= 1'b1;

end

//We don't want to change states when we are writing to memory

else if (data\_enable & state\_change) begin

//if enable is set (we are reading in data) then goto next cycle

current\_state <= mux\_out;

state\_change <= 1'b0;

end

else if (data\_enable & state\_change == 1'b0) begin

state\_change <= 1'b1;

end

//otherwise enable not set so do nothing (stay in current state)

end

//map match vector to LS 16 bits of the memory output

assign match\_vector = mem\_out[15:0];

supply1 vcc;

//now lets add other modules and connect them together

//connect the tile to memory for storing state machine data

//tile\_mem ram0(.data(config\_data), .wren(write), .wraddress(address), .rdaddress(current\_state), .clock(clk), .q(mem\_out));

ram ram0(.clk(clk), .address(address), .rdaddress(current\_state), .data\_in(config\_data), .data\_out(mem\_out), .we(write));

//hook up mux to determine next state

mux\_4\_1 mux0(.i0(mem\_out[47:40]), .i1(mem\_out[39:32]), .i2(mem\_out[31:24]), .i3(mem\_out[23:16]), .sel(data\_in), .mux\_out(mux\_out));

endmodule //End the tile module