Finite State Machine & Memories

module Tesla\_Control #(

    parameter MIN\_DISTANCE = 7'd40      // 40 meters

)(

    input  wire        clk,

    input  wire        rst,             // active high async reset

    input  wire [7:0]  speed\_limit,     // allowable speed limit

    input  wire [7:0]  car\_speed,       // current car speed

    input  wire [6:0]  leading\_distance,// distance to front vehicle

    output reg         unlock\_doors,    // unlock doors when HIGH

    output reg         accelerate\_car   // accelerate car when HIGH

);

    // FSM state encoding

    localparam STOP       = 2'b00;

    localparam ACCELERATE = 2'b01;

    localparam DECELERATE = 2'b10;

    // State registers

    reg [1:0] crnt\_state, nxt\_state;

    // Sequential block: state update

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            crnt\_state <= STOP;

        end else begin

            crnt\_state <= nxt\_state;

        end

    end

    // Next state logic

    always @(\*) begin

        nxt\_state = crnt\_state; // default hold

        case (crnt\_state)

            STOP: begin

                if (leading\_distance >= MIN\_DISTANCE) begin

                    nxt\_state = ACCELERATE;

                end else begin

                    nxt\_state = STOP;

                end

            end

            ACCELERATE: begin

                if (leading\_distance < MIN\_DISTANCE || car\_speed > speed\_limit) begin

                    nxt\_state = DECELERATE;

                end else begin

                    nxt\_state = ACCELERATE;

                end

            end

            DECELERATE: begin

                if (car\_speed == 0) begin

                    nxt\_state = STOP;

                end else if (leading\_distance >= MIN\_DISTANCE && car\_speed < speed\_limit) begin

                    nxt\_state = ACCELERATE;

                end else begin

                    nxt\_state = DECELERATE;

                end

            end

            default: nxt\_state = STOP;

        endcase

    end

    // Output logic (Moore FSM: depends only on crnt\_state)

    always @(\*) begin

        case (crnt\_state)

            STOP: begin

                unlock\_doors   = 1'b1;

                accelerate\_car = 1'b0;

            end

            ACCELERATE: begin

                unlock\_doors   = 1'b0;

                accelerate\_car = 1'b1;

            end

            DECELERATE: begin

                unlock\_doors   = 1'b0;

                accelerate\_car = 1'b0;

            end

            default: begin

                unlock\_doors   = 1'b1;

                accelerate\_car = 1'b0;

            end

        endcase

    end

endmodule

module Tesla\_Control\_tb();

    // Testbench signals

    reg clk;

    reg rst;

    reg [7:0] speed\_limit;

    reg [7:0] car\_speed;

    reg [6:0] leading\_distance;

    wire unlock\_doors;

    wire accelerate\_car;

    // Instantiate DUT

    Tesla\_Control DUT (

        .clk(clk),

        .rst(rst),

        .speed\_limit(speed\_limit),

        .car\_speed(car\_speed),

        .leading\_distance(leading\_distance),

        .unlock\_doors(unlock\_doors),

        .accelerate\_car(accelerate\_car)

    );

    // Clock generation

    initial begin

        clk = 0;

        forever #5 clk = ~clk; // 100MHz clock

    end

    // Stimulus

    initial begin

        // Initialize signals

        rst = 1;

        speed\_limit = 8'd100;    // 100 km/h speed limit

        car\_speed = 8'd0;

        leading\_distance = 7'd20; // 20m (too close)

        // Apply reset

        #12 rst = 0;

        // Car is too close (should be STOP)

        #20;

        // Increase distance (safe to accelerate)

        leading\_distance = 7'd50;

        car\_speed = 8'd20;

        #40;

        // Car accelerates until close to speed limit

        car\_speed = 8'd95;

        #40;

        // Exceed speed limit (should DECELERATE)

        car\_speed = 8'd110;

        #40;

        // Reduce speed gradually

        car\_speed = 8'd80;

        leading\_distance = 7'd30; // too close again

        #40;

        // Car slows down to stop

        car\_speed = 8'd0;

        #40;

        // Resume safe distance

        leading\_distance = 7'd70;

        car\_speed = 8'd50;

        #40;

        // End simulation

        $stop;

    end

    // Monitor output

    initial begin

        $display("Time\tState\tSpeed\tLimit\tDistance\tUnlock\tAccel");

        $monitor("%0t\t%d\t%d\t%d\t%d\t%b\t%b",

                 $time, DUT.crnt\_state, car\_speed, speed\_limit, leading\_distance,

                 unlock\_doors, accelerate\_car);

    end

endmodule

QuestaSim snippet

صورة تحتوي على لقطة شاشة, برمجيات, برامج الوسائط المتعددة, نص

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

# Time State Speed Limit Distance Unlock Accel

# 0 0 0 100 20 1 0

# 32000 0 20 100 50 1 0

# 35000 1 20 100 50 0 1

# 72000 1 95 100 50 0 1

# 112000 1 110 100 50 0 1

# 115000 2 110 100 50 0 0

# 152000 2 80 100 30 0 0

# 192000 2 0 100 30 0 0

# 195000 0 0 100 30 1 0

# 232000 0 50 100 70 1 0

# 235000 1 50 100 70 0 1

Elaboration

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, الحاسوب, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, رسم بياني, لقطة شاشة, خط

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Synthesis

“Messages” tab

صورة تحتوي على لقطة شاشة, نص, الحاسوب, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على رسم بياني, نص, خط, لقطة شاشة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على لقطة شاشة, نص, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Netlist file

صورة تحتوي على نص, لقطة شاشة, برمجيات, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Implementation

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Device snippet

صورة تحتوي على لقطة شاشة, برمجيات, نص, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

module Gray\_Counter(

    input  wire clk,

    input  wire rst,      // active high async reset

    output reg [1:0] y    // 2-bit Gray output

);

    // FSM state encoding

    localparam A = 2'b00;

    localparam B = 2'b01;

    localparam C = 2'b11;

    localparam D = 2'b10;

    // State registers

    reg [1:0] crnt\_state, nxt\_state;

    // Sequential: State update

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            crnt\_state <= A;   // Reset to state A (00)

        end else begin

            crnt\_state <= nxt\_state;

        end

    end

    // Combinational: Next-state logic

    always @(\*) begin

        case (crnt\_state)

            A: nxt\_state = B;  // 00 -> 01

            B: nxt\_state = C;  // 01 -> 11

            C: nxt\_state = D;  // 11 -> 10

            D: nxt\_state = A;  // 10 -> 00

            default: nxt\_state = A;

        endcase

    end

    // Moore Output: depends only on crnt\_state

    always @(\*) begin

        case (crnt\_state)

            A: y = 2'b00;

            B: y = 2'b01;

            C: y = 2'b11;

            D: y = 2'b10;

            default: y = 2'b00;

        endcase

    end

endmodule

module Gray\_Counter\_Golden\_Model(

    input  wire clk,

    input  wire rst,           // active high async reset

    output reg [1:0] gray\_out  // 2-bit Gray output

);

    reg [1:0] binary\_count;

    // Sequential binary counter

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            binary\_count <= 2'b00;       // reset binary counter

        end else begin

            binary\_count <= binary\_count + 1'b1; // increment

        end

    end

    // Binary-to-Gray conversion

    always @(\*) begin

        gray\_out[1] = binary\_count[1];                 // MSB same

        gray\_out[0] = binary\_count[1] ^ binary\_count[0]; // XOR for LSB

    end

endmodule

module Gray\_Counter\_tb();

    // Testbench signals

    reg clk;

    reg rst;

    wire [1:0] gray\_fsm;     // DUT output

    wire [1:0] gray\_golden;  // Golden model output

    // Instantiate FSM-based Gray counter (DUT)

    Gray\_Counter dut (

        .clk(clk),

        .rst(rst),

        .y(gray\_fsm)

    );

    // Instantiate Binary-to-Gray counter (Golden model)

    Gray\_Counter\_Golden\_Model golden (

        .clk(clk),

        .rst(rst),

        .gray\_out(gray\_golden)

    );

    // Clock generation: 100 MHz

    initial begin

        clk = 0;

        forever #5 clk = ~clk;

    end

    // Stimulus

    initial begin

        // Apply reset

        rst = 1;

        #12;

        rst = 0;

        // Run for some cycles

        #200;

        $stop;

    end

    // Monitor outputs

    initial begin

        $display("Time\tFSM\_Out\tGolden\_Out\tMatch");

        $monitor("%0t\t%b\t%b\t%b",

                 $time, gray\_fsm, gray\_golden,

                 (gray\_fsm == gray\_golden));

    end

endmodule

QuestaSim snippet

صورة تحتوي على لقطة شاشة, برمجيات, نص, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

# Time FSM\_Out Golden\_Out Match

# 0 00 00 1

# 15000 01 01 1

# 25000 11 11 1

# 35000 10 10 1

# 45000 00 00 1

# 55000 01 01 1

# 65000 11 11 1

# 75000 10 10 1

# 85000 00 00 1

# 95000 01 01 1

# 105000 11 11 1

# 115000 10 10 1

# 125000 00 00 1

# 135000 01 01 1

# 145000 11 11 1

# 155000 10 10 1

# 165000 00 00 1

# 175000 01 01 1

# 185000 11 11 1

# 195000 10 10 1

# 205000 00 00 1

Elaboration

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, رسم بياني, لقطة شاشة, خط

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Synthesis

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, لقطة شاشة, رسم بياني, خط

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Implementation

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Device snippet

صورة تحتوي على لقطة شاشة, برنامج الرسومات, برامج الوسائط المتعددة, ميدان/ مربع

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

module Seq\_111\_Detector(

    input  wire clk,

    input  wire rst,   // async active-high reset

    input  wire Din,   // serial data input (1 bit per clock)

    output reg  ERR    // error flag: high when "111" detected

);

    // State encoding

    localparam START     = 3'b000;

    localparam D0\_IS\_1   = 3'b001; // first 1 seen

    localparam D1\_IS\_1   = 3'b010; // second consecutive 1 seen

    localparam D0\_NOT\_1  = 3'b011; // first bit was 0

    localparam D1\_NOT\_1  = 3'b100; // second bit was 0

    reg [2:0] crnt\_state, nxt\_state;

    // State register

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            crnt\_state <= START;

        end else begin

            crnt\_state <= nxt\_state;

        end

    end

    // Next-state logic + output (Mealy)

    always @(\*) begin

        // defaults

        nxt\_state = crnt\_state;

        ERR = 1'b0;

        case (crnt\_state)

            START: begin

                if (Din) begin

                    nxt\_state = D0\_IS\_1;

                end else begin

                    nxt\_state = D0\_NOT\_1;

                end

            end

            D0\_IS\_1: begin

                if (Din) begin

                    nxt\_state = D1\_IS\_1;

                end else begin

                    nxt\_state = D1\_NOT\_1;

                end

            end

            D0\_NOT\_1: begin

                // regardless of Din, move on

                nxt\_state = D1\_NOT\_1;

            end

            D1\_IS\_1: begin

                // regardless of Din, back to start

                nxt\_state = START;

                if (Din) begin

                    // detected 111

                    ERR = 1'b1;

                end

            end

            D1\_NOT\_1: begin

                // regardless of Din, back to start

                nxt\_state = START;

            end

            default: begin

                nxt\_state = START;

            end

        endcase

    end

endmodule

module Seq\_111\_Detector\_tb();

    reg clk;

    reg rst;

    reg Din;

    wire ERR;

    // Instantiate DUT

    Seq\_111\_Detector DUT (

        .Din(Din),

        .clk(clk),

        .rst(rst),

        .ERR(ERR)

    );

    // Clock generation (100 MHz ? 10 ns period)

    initial begin

        clk = 0;

        forever #5 clk = ~clk;

    end

    // Stimulus

    initial begin

        // Apply reset

        rst = 1;

        Din = 0;

        #15;

        rst = 0;

        // Generate random serial data for 50 cycles

        repeat (50) begin

            @(posedge clk);

            Din = $random % 2;   // Random 0 or 1

        end

        // Finish simulation

        #50;

        $stop;

    end

    // Display activity in console

    initial begin

        $display("Time\tDin\tERR\tState");

        $monitor("%0t\t%b\t%b\t%b", $time, Din, ERR, DUT.crnt\_state);

    end

endmodule

QuestaSim snippet

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

# Time Din ERR State

# 0 0 0 000

# 15000 0 0 011

# 25000 1 0 100

# 35000 1 0 000

# 45000 1 0 001

# 55000 1 1 010

# 65000 1 0 000

# 75000 1 0 001

# 85000 0 0 010

# 95000 1 0 000

# 105000 1 0 001

# 115000 0 0 010

# 125000 1 0 000

# 135000 1 0 001

# 145000 0 0 010

# 155000 1 0 000

# 165000 0 0 001

# 175000 1 0 100

# 185000 0 0 000

# 195000 1 0 011

# 205000 1 0 100

# 215000 0 0 000

# 225000 1 0 011

# 235000 0 0 100

# 245000 0 0 000

# 255000 0 0 011

# 265000 1 0 100

Elaboration

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, رسم بياني, لقطة شاشة, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Synthesis

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, لقطة شاشة, رسم بياني, خط

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Implementation

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Device snippet

صورة تحتوي على نص, لقطة شاشة, برامج الوسائط المتعددة, تحرير

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

module Single\_Port\_Sync\_RAM #(

    parameter MEM\_WIDTH      = 16,

    parameter MEM\_DEPTH      = 1024,

    parameter ADDR\_SIZE      = $clog2(MEM\_DEPTH),

    parameter ADDR\_PIPELINE  = "FALSE",  // "TRUE" or "FALSE"

    parameter DOUT\_PIPELINE  = "TRUE",   // "TRUE" or "FALSE"

    parameter PARITY\_ENABLE  = 1

)(

    input  wire                     clk,

    input  wire                     rst,        // synchronous active high reset

    input  wire                     wr\_en,      // write enable

    input  wire                     rd\_en,      // read enable

    input  wire                     blk\_select, // chip enable

    input  wire [ADDR\_SIZE-1:0]     addr,       // input address

    input  wire [MEM\_WIDTH-1:0]     din,        // data input

    input  wire                     addr\_en,    // enable for pipelined address

    input  wire                     dout\_en,    // enable for pipelined data out

    output reg  [MEM\_WIDTH-1:0]     dout,       // data output

    output wire                     parity\_out  // even parity of dout

);

    // Memory array

    reg [MEM\_WIDTH-1:0] mem [0:MEM\_DEPTH-1];

    // Address pipeline register

    reg [ADDR\_SIZE-1:0] addr\_reg;

    wire [ADDR\_SIZE-1:0] eff\_addr;

    generate

        if (ADDR\_PIPELINE == "TRUE") begin

            always @(posedge clk) begin

                if (rst) begin

                    addr\_reg <= {ADDR\_SIZE{1'b0}};

                end else if (addr\_en) begin

                    addr\_reg <= addr;

                end

            end

            assign eff\_addr = addr\_reg;

        end else begin

            assign eff\_addr = addr;

        end

    endgenerate

    // RAM write operation

    always @(posedge clk) begin

        if (blk\_select && wr\_en) begin

            mem[eff\_addr] <= din;

        end

    end

    // Read data (direct or pipelined)

    reg [MEM\_WIDTH-1:0] dout\_reg;

    generate

        if (DOUT\_PIPELINE == "TRUE") begin

            always @(posedge clk) begin

                if (rst) begin

                    dout\_reg <= {MEM\_WIDTH{1'b0}};

                end else if (blk\_select && rd\_en && dout\_en) begin

                    dout\_reg <= mem[eff\_addr];

                end

            end

            always @(\*) begin

                if (blk\_select && rd\_en) begin

                    dout = dout\_reg;

                end else begin

                    dout = {MEM\_WIDTH{1'b0}};

                end

            end

        end else begin

            always @(\*) begin

                if (blk\_select && rd\_en) begin

                    dout = mem[eff\_addr];

                end else begin

                    dout = {MEM\_WIDTH{1'b0}};

                end

            end

        end

    endgenerate

    // Even parity output

    generate

        if (PARITY\_ENABLE == 1) begin

            assign parity\_out = (blk\_select && rd\_en) ? ^dout : 1'b0;

        end else begin

            assign parity\_out = 1'b0;

        end

    endgenerate

endmodule

module Single\_Port\_Sync\_RAM\_tb();

    // Parameters

    localparam MEM\_WIDTH = 16;

    localparam MEM\_DEPTH = 1024;

    localparam ADDR\_SIZE = $clog2(MEM\_DEPTH);

    // DUT signals

    reg                     clk;

    reg                     rst;

    reg                     wr\_en;

    reg                     rd\_en;

    reg                     blk\_select;

    reg  [ADDR\_SIZE-1:0]    addr;

    reg  [MEM\_WIDTH-1:0]    din;

    reg                     addr\_en;

    reg                     dout\_en;

    wire [MEM\_WIDTH-1:0]    dout;

    wire                    parity\_out;

    // Instantiate DUT

    Single\_Port\_Sync\_RAM #(

        .MEM\_WIDTH(MEM\_WIDTH),

        .MEM\_DEPTH(MEM\_DEPTH),

        .ADDR\_SIZE(ADDR\_SIZE),

        .ADDR\_PIPELINE("TRUE"),

        .DOUT\_PIPELINE("TRUE"),

        .PARITY\_ENABLE(1)

    ) DUT (

        .clk(clk),

        .rst(rst),

        .wr\_en(wr\_en),

        .rd\_en(rd\_en),

        .blk\_select(blk\_select),

        .addr(addr),

        .din(din),

        .addr\_en(addr\_en),

        .dout\_en(dout\_en),

        .dout(dout),

        .parity\_out(parity\_out)

    );

    // Clock generation

    initial begin

        clk = 0;

        forever #5 clk = ~clk;  // 100 MHz clock

    end

    // Stimulus

    integer i;

    initial begin

        // Initialize

        rst = 1;

        wr\_en = 0;

        rd\_en = 0;

        blk\_select = 0;

        addr = 0;

        din = 0;

        addr\_en = 0;

        dout\_en = 0;

        // Release reset

        #20 rst = 0;

        // Perform 20 random write/read operations

        for (i = 0; i < 20; i = i + 1) begin

            @(posedge clk);

            blk\_select = 1;

            addr\_en    = 1;

            dout\_en    = 1;

            // Randomize control signals

            wr\_en = $random % 2;

            rd\_en = $random % 2;

            // Randomize address and data

            addr = $random % MEM\_DEPTH;

            din  = $random;

            // Ensure no simultaneous read & write (for clean demo)

            if (wr\_en && rd\_en)

                rd\_en = 0;

        end

        // End simulation

        #50 $stop;

    end

    // Monitor outputs

    initial begin

        $display("Time\tblk\_sel\twr\_en\trd\_en\taddr\tdin\tdout\tparity");

        $monitor("%0t\t%b\t%b\t%b\t%0d\t%h\t%h\t%b",

                 $time, blk\_select, wr\_en, rd\_en, addr, din, dout, parity\_out);

    end

endmodule

QuestaSim snippet

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

# Time blk\_sel wr\_en rd\_en addr din dout parity

# 0 0 0 0 0 0000 0000 0

# 25000 1 0 1 521 5663 0000 0

# 35000 1 1 0 101 5212 0000 0

# 45000 1 1 0 374 cd3d 0000 0

# 55000 1 1 0 505 24c6 0000 0

# 65000 1 1 0 997 7277 0000 0

# 75000 1 0 1 498 96ce xxxx x

# 85000 1 0 1 348 28bd xxxx x

# 95000 1 1 0 611 870a 0000 0

# 105000 1 0 0 426 cc9d 0000 0

# 115000 1 0 1 13 d653 xxxx x

# 125000 1 1 0 514 3eae 0000 0

# 135000 1 1 0 291 650a 0000 0

Elaboration

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, عرض, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, رسم بياني, لقطة شاشة, خط

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Synthesis

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, لقطة شاشة, عرض, رسم بياني

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على لقطة شاشة, نص, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Implementation

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, لقطة شاشة, برمجيات, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, عرض, الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Device snippet

صورة تحتوي على لقطة شاشة, نص, برامج الوسائط المتعددة, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

module Async\_FIFO #(

    parameter FIFO\_WIDTH = 16,

    parameter FIFO\_DEPTH = 512,

    parameter ADDR\_SIZE  = $clog2(FIFO\_DEPTH)

)(

    input  wire                  clk\_a,   // write clock

    input  wire                  clk\_b,   // read clock

    input  wire                  rst,     // sync reset

    // Write port

    input  wire [FIFO\_WIDTH-1:0] din\_a,

    input  wire                  wen\_a,

    // Read port

    input  wire                  ren\_b,

    output reg  [FIFO\_WIDTH-1:0] dout\_b,

    // Status

    output wire                  full,

    output wire                  empty

);

    // Internal memory

    reg [FIFO\_WIDTH-1:0] mem [0:FIFO\_DEPTH-1];

    // Write and read pointers

    reg [ADDR\_SIZE:0] wr\_ptr;  // one bit wider for full detection

    reg [ADDR\_SIZE:0] rd\_ptr;

    // Write logic

    always @(posedge clk\_a) begin

        if (rst) begin

            wr\_ptr <= 0;

        end else if (wen\_a && !full) begin

            mem[wr\_ptr[ADDR\_SIZE-1:0]] <= din\_a;

            wr\_ptr <= wr\_ptr + 1;

        end

    end

    // Read logic

    always @(posedge clk\_b) begin

        if (rst) begin

            rd\_ptr  <= 0;

            dout\_b  <= {FIFO\_WIDTH{1'b0}};

        end else if (ren\_b && !empty) begin

            dout\_b <= mem[rd\_ptr[ADDR\_SIZE-1:0]];

            rd\_ptr <= rd\_ptr + 1;

        end

    end

    // Status flags

    assign empty = (wr\_ptr == rd\_ptr);

    assign full  = ( (wr\_ptr[ADDR\_SIZE]     != rd\_ptr[ADDR\_SIZE]) &&

                     (wr\_ptr[ADDR\_SIZE-1:0] == rd\_ptr[ADDR\_SIZE-1:0]) );

endmodule

module Async\_FIFO\_tb();

    // Parameters

    localparam FIFO\_WIDTH = 16;

    localparam FIFO\_DEPTH = 16;   // small for testing

    localparam ADDR\_SIZE  = $clog2(FIFO\_DEPTH);

    // DUT signals

    reg                     clk\_a;

    reg                     clk\_b;

    reg                     rst;

    reg  [FIFO\_WIDTH-1:0]   din\_a;

    reg                     wen\_a;

    reg                     ren\_b;

    wire [FIFO\_WIDTH-1:0]   dout\_b;

    wire                    full;

    wire                    empty;

    // Instantiate DUT

    Async\_FIFO #(

        .FIFO\_WIDTH(FIFO\_WIDTH),

        .FIFO\_DEPTH(FIFO\_DEPTH),

        .ADDR\_SIZE(ADDR\_SIZE)

    ) DUT (

        .clk\_a(clk\_a),

        .clk\_b(clk\_b),

        .rst(rst),

        .din\_a(din\_a),

        .wen\_a(wen\_a),

        .ren\_b(ren\_b),

        .dout\_b(dout\_b),

        .full(full),

        .empty(empty)

    );

    // Clock generation

    initial begin

        clk\_a = 0;

        forever #5 clk\_a = ~clk\_a;   // 100 MHz

    end

    initial begin

        clk\_b = 0;

        forever #7 clk\_b = ~clk\_b;   // ~71 MHz

    end

    // Stimulus

    integer i;

    initial begin

        // Init

        rst   = 1;

        wen\_a = 0;

        ren\_b = 0;

        din\_a = 0;

        // Release reset

        #20 rst = 0;

        // Run for 50 cycles of write clock

        for (i = 0; i < 50; i = i + 1) begin

            @(posedge clk\_a);

            // Randomize write

            wen\_a = $random % 2;

            din\_a = $random;

            // Randomize read (using read clock domain)

            @(posedge clk\_b);

            ren\_b = $random % 2;

            // Avoid write when full

            if (full) wen\_a = 0;

            // Avoid read when empty

            if (empty) ren\_b = 0;

        end

        #50 $stop;

    end

    // Monitor activity

    initial begin

        $display("Time\tclk\_a\twen\tdin\tfull\tempty\tclk\_b\tren\tdout");

        $monitor("%0t\t%b\t%b\t%h\t%b\t%b\t%b\t%b\t%h",

                 $time, clk\_a, wen\_a, din\_a, full, empty, clk\_b, ren\_b, dout\_b);

    end

endmodule

QuestaSim snippet

صورة تحتوي على نص, لقطة شاشة, برمجيات, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Elaboration

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, رسم بياني, لقطة شاشة, خط

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Synthesis

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Schematic snippet

صورة تحتوي على نص, رسم بياني, خطة, تخطيطي

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Implementation

“Messages” tab

صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Utilization report

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Timing report

صورة تحتوي على نص, لقطة شاشة, برمجيات, عرض

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Device snippet

صورة تحتوي على نص, لقطة شاشة, عرض, مستطيل

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

module ALSU #(

    parameter INPUT\_PRIORITY = "A", // "A" or "B"

    parameter FULL\_ADDER     = "ON" // "ON" or "OFF"

)(

    input  wire        clk,

    input  wire        rst,

    input  wire        cin,

    input  wire        red\_op\_A,

    input  wire        red\_op\_B,

    input  wire        bypass\_A,

    input  wire        bypass\_B,

    input  wire        direction,

    input  wire        serial\_in,

    input  wire [2:0]  opcode,

    input  wire [2:0]  A,

    input  wire [2:0]  B,

    output reg  [15:0] leds,

    output reg  [5:0]  out

);

    // Registered inputs

    reg cin\_reg, red\_op\_A\_reg, red\_op\_B\_reg, bypass\_A\_reg, bypass\_B\_reg;

    reg direction\_reg, serial\_in\_reg;

    reg [2:0] opcode\_reg, A\_reg, B\_reg;

    // IP outputs

    wire [3:0] out\_add;   // adder output (3-bit + carry)

    wire [5:0] out\_mult;  // multiplier output

    // Invalid operation detection

    wire invalid\_red\_op, invalid\_opcode, invalid;

    assign invalid\_red\_op = (red\_op\_A\_reg | red\_op\_B\_reg) & (opcode\_reg[1] | opcode\_reg[2]);

    assign invalid\_opcode = opcode\_reg[1] & opcode\_reg[2];

    assign invalid        = invalid\_red\_op | invalid\_opcode;

    // Pipeline inputs

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            cin\_reg        <= 0;

            red\_op\_B\_reg   <= 0;

            red\_op\_A\_reg   <= 0;

            bypass\_B\_reg   <= 0;

            bypass\_A\_reg   <= 0;

            direction\_reg  <= 0;

            serial\_in\_reg  <= 0;

            opcode\_reg     <= 0;

            A\_reg          <= 0;

            B\_reg          <= 0;

        end else begin

            cin\_reg        <= cin;

            red\_op\_B\_reg   <= red\_op\_B;

            red\_op\_A\_reg   <= red\_op\_A;

            bypass\_B\_reg   <= bypass\_B;

            bypass\_A\_reg   <= bypass\_A;

            direction\_reg  <= direction;

            serial\_in\_reg  <= serial\_in;

            opcode\_reg     <= opcode;

            A\_reg          <= A;

            B\_reg          <= B;

        end

    end

    // LEDs blink if invalid

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            leds <= 0;

        end else begin

            if (invalid)

                leds <= ~leds;

            else

                leds <= 0;

        end

    end

    // Generate block for adder depending on FULL\_ADDER

    generate

        if (FULL\_ADDER == "ON") begin : ADDER\_FULL

            c\_addsub\_0 a1 (

                .A    (A\_reg),     // input wire [2:0]

                .B    (B\_reg),     // input wire [2:0]

                .C\_IN (cin\_reg),   // input wire

                .S    (out\_add)    // output wire [3:0]

            );

        end else begin : ADDER\_NO\_CIN

            c\_addsub\_0 a1 (

                .A    (A\_reg),     // input wire [2:0]

                .B    (B\_reg),     // input wire [2:0]

                .C\_IN (1'b0),      // force carry-in = 0

                .S    (out\_add)    // output wire [3:0]

            );

        end

    endgenerate

    // Multiplier IP

    mult\_gen\_1 m1 (

        .A (A\_reg),  // input wire [2:0]

        .B (B\_reg),  // input wire [2:0]

        .P (out\_mult) // output wire [5:0]

    );

    // ALSU output logic

    always @(posedge clk or posedge rst) begin

        if (rst) begin

            out <= 0;

        end else begin

            if (bypass\_A\_reg && bypass\_B\_reg) begin

                out <= (INPUT\_PRIORITY == "A") ? A\_reg : B\_reg;

            end else if (bypass\_A\_reg) begin

                out <= A\_reg;

            end else if (bypass\_B\_reg) begin

                out <= B\_reg;

            end else if (invalid) begin

                out <= 0;

            end else begin

                case (opcode\_reg)

                    3'h0: begin // AND / reduction

                        if (red\_op\_A\_reg && red\_op\_B\_reg)

                            out <= (INPUT\_PRIORITY == "A") ? &A\_reg : &B\_reg;

                        else if (red\_op\_A\_reg)

                            out <= &A\_reg;

                        else if (red\_op\_B\_reg)

                            out <= &B\_reg;

                        else

                            out <= A\_reg & B\_reg;

                    end

                    3'h1: begin // XOR / reduction

                        if (red\_op\_A\_reg && red\_op\_B\_reg)

                            out <= (INPUT\_PRIORITY == "A") ? ^A\_reg : ^B\_reg;

                        else if (red\_op\_A\_reg)

                            out <= ^A\_reg;

                        else if (red\_op\_B\_reg)

                            out <= ^B\_reg;

                        else

                            out <= A\_reg ^ B\_reg;

                    end

                    3'h2: out <= out\_add;   // Use IP adder

                    3'h3: out <= out\_mult;  // Use IP multiplier

                    3'h4: begin // Shift with serial input

                        if (direction\_reg)

                            out <= {out[4:0], serial\_in\_reg};

                        else

                            out <= {serial\_in\_reg, out[5:1]};

                    end

                    3'h5: begin // Rotate

                        if (direction\_reg)

                            out <= {out[4:0], out[5]};

                        else

                            out <= {out[0], out[5:1]};

                    end

                    default: out <= 0;

                endcase

            end

        end

    end

endmodule