به نام خدا

آزمایشگاه طراحی سیستم‌های دیجیتال

استاد

دکتر انصاری

گزارش

آزمایش شماره ۱۰:

طراحی یک پردازنده ساده

اعضای گروه:

امیرحسین علمدار 400105144

پیام تائبی 400104867

علیرضا سلیمیان 400105036

شرح آزمایش

در این آزمایش می خواهیم یک پردازنده ساده با معماری پشته ای طراحی کنیم. طبق فرضیاتی که در دستور کار مطرح شده، این پردازنده یک پشته با 8 ثبات 8 بیتی و حافظهای با ظرفیت 256 خان هی 8 بیتی دارد که 8 خانه ی آخر آن برای ورودی خروجی به صورت Memory Mapped I/O مورد استفاده قرار می گیرند.
این پردازنده 8 دستور دارد که به شرح زیرند:

مقدار ۸ بیتی ثابت C را در پشته PUSH می‌کند.	**0000 C**	**PUSHC**
مقدار خانه حافظه یا درگاه که آدرس M دارد را در پشته PUSH می‌کند.	**0001 M**	**PUSH**
از پشته POP کرده و مقدار را در آدرس یا درگاه M قرار می‌دهد.	**0010 M**	**POP**
از پشته POP کرده و مقدار را در PC قرار می‌دهد.	**0011**	**JUMP**
اگر پرچم Z ۱ باشد، از پشته POP کرده و مقدار را در PC قرار می‌دهد.	**0100**	**JZ**
اگر پرچم S ۱ باشد، از پشته POP کرده و مقدار را در PC قرار می‌دهد.	**0101**	**JS**
دو مقدار بالای پشته را POP کرده، آنها را جمع می‌کند و حاصل را در پشته PUSH می‌کند.	**0110**	**ADD**
دو مقدار بالای پشته را POP کرده، آنها را تفریق می‌کند و حاصل را در پشته PUSH می‌کند.	**0111**	**SUB**

طبق دستورات فوق مشخص است که دو پرچم S (نشان دهندهی منفی بودن حاصل آخرین جمع یا تفریق) و Z (نشان دهنده صفر بودن حاصل آخرین جمع یا تفریق) داریم که تنها با دستورات ADD و SUB می‌توان مقادیرشان را تغییر داد. ضمنا تمامی محاسبات علامت دار و با مکمل ۲ انجام می شوند.

در نهایت می‌خواهیم مقدار X را از ورودی بگیریم و حاصل زیر را توسط پردازنده برگردانیم:

$$Y=((X+23)*2)-12$$

Stack_machine.v

ورودی ها

- Clk
- rstN
- in: مقدار ورودی ۸ بیتی

خروجی:

- Out: مقدار خروجی ۸ بیتی

ابتدا به مقداردهی های ابتدایی مدار می پردازیم.

برای نمایش حافظه از یک آرایه 256 تایی که هر خانه ی آن 8 بیت است استفاده می کنیم. برای نمایش حافظه ی دستورات از یک آرایه 32 تایی که هر خانه ی آن 12 بیت است استفاده می کنیم و برای نمایش پشته از یک آرایه 8تایی که هر خانه آن 8 بیت است استفاده می کنیم. با توجه به اینکه حافظه دستورات، 32 خانه دارد، از pc که یک رجیستر5 بیتی است برای مشخص کردن دستوری که در آن قرار داریم، استفاده می کنیم. برای مشخص کردن سر پشته، از رجیستر 3 بیتی sp استفاده می کنیم.

مقادیر مفروض برای opcode هر دستور را توسط parameter مشخص می کنیم.

برای هر ورودی و خروجی in ، out خانه ای از حافظه را اختصاص میدهیم. با این کار در واقع I/O ها به خانه هایی از حافظه map می شوند که همان طراحی I/O mapped memory می باشد.

بیت های اول تا چهارم دستوری که در آن قرار داریم، به عنوان opcode آن دستور مشخص می شوند.

بیت های پنجم تا دوازدهم هر دستور درinst_value قرار میگیرد که مهم بودن یا نبودن این مقدار بستگی به دستور دارد.

مقدار موجود در خانه ی out_addr از حافظه در خروجی قرار می گیرد.

حاصل جمع و تفریق دو خانه ی آخر پشته، محاسبه شده و دراdd_result وsub_result قرار می گیرند و در دستور مناسب، pushخواهند شد .کد وریلاگ این بخش از آزمایش به شرح زیر است:

```
module stack_machine (
    input           clk,
    input           rstN,
    input   [7:0]   in,
    output  [7:0]   out
);

reg     [7:0]   data_mem    [255:0];
reg     [7:0]   stack       [7:0];
reg     [1:12]  inst_mem    [31:0];

reg     [4:0]   pc;
reg     [2:0]   sp;

reg             s_flag = 0, z_flag = 0;
wire    [3:0]   inst_op;
wire    [7:0]   inst_value;
wire    [7:0]   add_result, sub_result;


parameter op_pushc      = 0;
parameter op_pushmem    = 1;
parameter op_pop        = 2;
parameter op_j          = 3;
parameter op_jz         = 4;
parameter op_js         = 5;
parameter op_add        = 6;
parameter op_sub        = 7;

parameter in_addr       = 254;
parameter out_addr      = 255;

assign inst_op          = inst_mem[pc][1:4];
assign inst_value       = inst_mem[pc][5:12];
assign out              = data_mem[out_addr];
assign add_result       = stack[sp - 2] + stack[sp - 1];
assign sub_result       = stack[sp - 2] - stack[sp - 1];
```

حال به پیاده سازی بخش ترتیبی مدار می پردازیم. از یک block always استفاده می کنیم که به لبه ی بالارونده کلاک و لبه ی پایین رونده ریست حساس است .در صورت 0 شدن rstN ،مقادیر pc ، sp،s_flag، z_flagو تمام خانه های پشته برابر با صفر می شوند .در غیر این صورت و با بالا رفتن لبه ی کلاک، مقدار ورودی در خانه ی in_addr از حافظه قرار می گیرد .سپس با توجه به مقدارinst_op عملیاتی انجام می شود:

- 0 – pushc :مقدار inst_value در باالترین خانه ی پشته push شده و اشاره گر به بالای پشته یکی افزایش می یابد.
- 1 - pushmem :مقدار موجود در خانه ی inst_value از حافظه در بالاترین خانه ی پشته push شده و اشاره گر به بالای پشته یکی افزایش می یابد.
- 2 – pop :مقدار موجود در بالاترین خانه ی پشته در خانه ی inst_value از حافظه قرار گرفته و اشاره گر به بالای پشته یکی کاهش می یابد.
- 3 – j :مقدار آخرین خانه ی پشته در pc قرار گرفته و اشاره گر به بالای پشته یکی کاهش می یابد.
- 4 – jz :در صورت یک بودن مقدار z_flag،مقدار آخرین خانه ی پشته در pc قرار گرفته و اشاره گر به بالای پشته یکی کاهش می یابد.

- js –5 :در صورت یک بودن مقدار s_flag ،مقدار آخرین خانه ی پشته در pc قرار گرفته و اشاره گر به بالای پشته یکی کاهش می یابد.
- add – 6 :مقدار جمع دو خانه ی آخر پشته در بالاترین خانه ی پشته قرار گرفته و اشاره گر به بالای پشته یکی کاهش می یابد. مقدار z_flag برابر با NOR بیت های حاصل جمع می شود و اگر این حاصل صفر باشد، z_flag برابر با 1 و در غیر این صورت برابر با صفر میشود. مقدار s_flag در صورت منفی بودن مقدار حاصل جمع برابر با 1 و در غیر این صورت 0 می شود.
- sub – 7 :مقدار تفریق دو خانه ی آخر پشته در بالاترین خانه ی پشته قرار گرفته و اشاره گر به بالای پشته یکی کاهش می یابد. مقدار z_flag برابر با NOR بیت های حاصل تفریق می شود و اگر این حاصل صفر باشد، z_flag برابر با 1 و در غیر این صورت برابر با صفر میشود. مقدار s_flag در صورت منفی بودن مقدار حاصل تفریق برابر با 1 و در غیر این صورت 0 میشود.

کد وریلاگ این بخش از مدار به شرح زیر است:

```
integer i;
always @(posedge clk or negedge rstN) begin
    if (~rstN) begin
        pc                              <= 0;
        sp                              <= 0;
        s_flag                          <= 0;
        z_flag                          <= 0;
        for (i = 0; i < 8; i = i + 1)
            stack[i]                    <= 0;
    end
    else begin
        data_mem[in_addr]               <= in;

        pc                              <= pc + 1;
        case (inst_op)
            /* PUSH CONSTANT */
            op_pushc: begin
                stack[sp]               <= inst_value;
                sp                      <= sp + 1;
            end

            /* PUSH MEMORY */
            op_pushmem: begin
                stack[sp]               <= data_mem[inst_value];
                sp                      <= sp + 1;
            end

            /* POP */
            op_pop: begin
                data_mem[inst_value]    <= stack[sp - 1];
                sp                      <= sp - 1;
            end

            /* JUMP */
            op_j: begin
                pc                      <= stack[sp - 1];
                sp                      <= sp - 1;
            end
```

```
            /* JUMP Z */
            op_jz: if (z_flag) begin
                pc                      <= stack[sp - 1];
                sp                      <= sp - 1;
            end

            /* JUMP S */
            op_js: if (s_flag) begin
                pc                      <= stack[sp - 1];
                sp                      <= sp - 1;
            end

            /* ADD */
            op_add: begin
                stack[sp - 2]           <= add_result;
                stack[sp - 1]           <= 0;
                sp                      <= sp - 1;
                z_flag                  <= ~|add_result;
                s_flag                  <= $signed(add_result) < 0;
            end

            /* SUB */
            op_sub: begin
                stack[sp - 2]           <= sub_result;
                stack[sp - 1]           <= 0;
                sp                      <= sp - 1;
                z_flag                  <= ~|sub_result;
                s_flag                  <= $signed(sub_result) < 0;
            end
        endcase
    end
end

endmodule
```

Formula.v

در این ماژول به محاسبه مقدار خواسته شده در دستور کار می پردازیم.

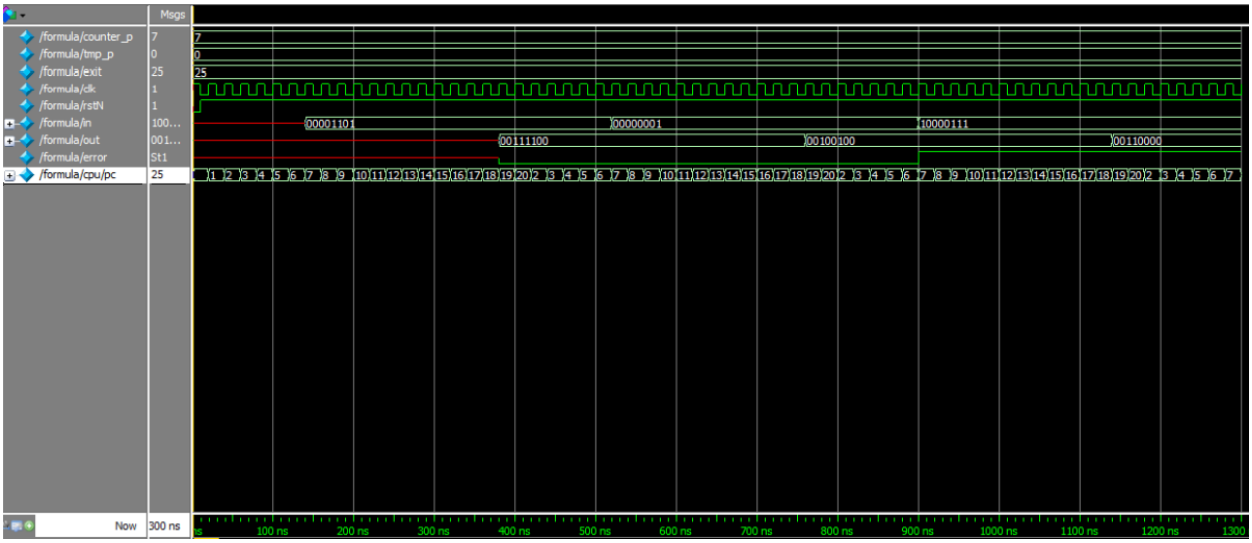
برای مشخص کردن دستورات، برای سهولت کار از یک ماکرو کمک گرفتیم که مقادیر addr و opcode و value را به عنوان ورودی گرفته، مقدار opcodeرا 8 بیت به چپ شیفت می دهد و آن را با value اور می کند. با این کار دستور مورد نظر ساخته میشود و فقط کافیست در خانه ی addr از حافظه ی دستورات قرار گیرد. ابتدا از ماژول stack_machine یک شیء با ورودی ها و خروجی مورد نظر می سازیم. مقدار error در صورت وجود خطا 1 می شود. در صورت بیشتر بودن خروجی out از 127 ،چون طبق دستورکار از حوزه قابل نمایش خارج است و یا در صورت منفی بودن ورودی خطا رخ می دهد.

در این bench test ،یک loop داریم که 3 بار تکرار می شود. در هر بار یک ورودی می گیرد و مقدار y را برای آن محاسبه می کند و خروجی می دهد. در صورت اتمام این loop ،از آن خارج شده و برنامه به اتمام می رسد .آدرس 0 از حافظه را به tmp اختصاص می دهیم که متغیر کمکی برای انجام محاسبات است. آدرس 7 از حافظه را به counter اختصاص می دهیم که تعداد دفعاتی ست که میخواهیم ورودی بگیریم. برای پایان کد از سطر 25 دستورات که عملا خالیست استفاده می شود. در یک block initial دستورات لازم برای 3 بار گرفتن ورودی و انجام محاسبات لازم برای هر یک از ورودی ها وارد می شود. سپس با 3 ورودی، کارایی ماژول را بررسی کرده و آنهارا مانیتور می کنیم. کد این بخش از آزمایش به شرح زیر است:

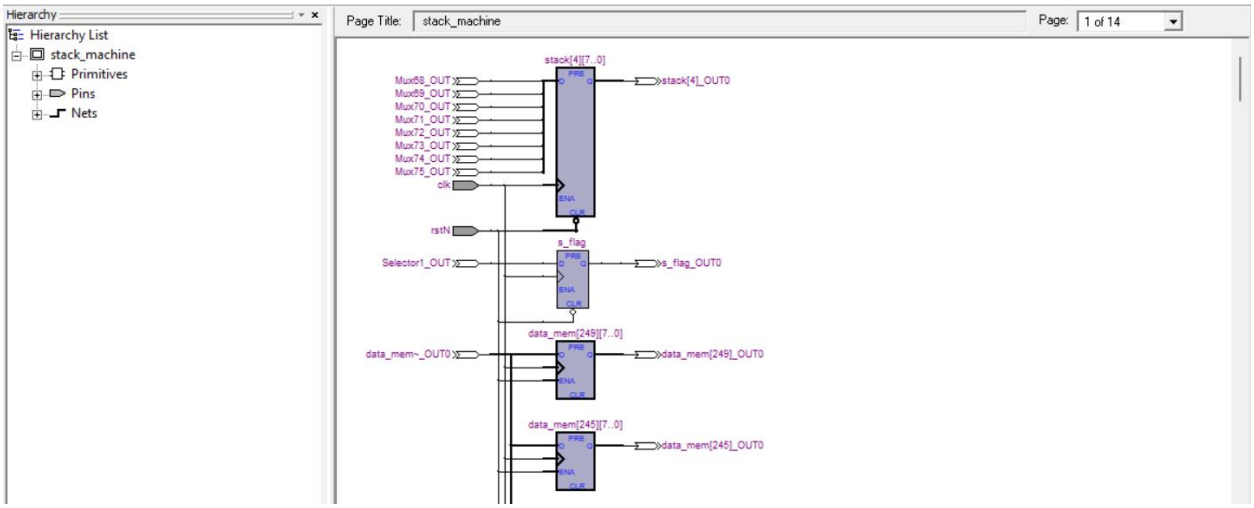
```
`define inst(ADDR, OP, VAL=0)    cpu.inst_mem[ADDR] = (OP << 8) | VAL

module formula;

reg             clk = 1, rstN = 0;
reg     [7:0]   in;
wire    [7:0]   out;

stack_machine cpu(clk, rstN, in, out);

assign error = (out > 127) || (in[7] == 1);
always #10 clk = ~clk;

/* Pointers to important memories */
localparam counter_p    = 7;
localparam tmp_p        = 0;

/* Address of exit program */
localparam exit         = 25;

initial begin
    /* THE CODE */

    /* counter = 3 */
    `inst(0,    cpu.op_pushc,   3);             // s_head  = 3
    `inst(1,    cpu.op_pop,     counter_p);     // counter = 3

    /* LOOP CONDITION */

    /* counter = counter - 1 */
    `inst(2,    cpu.op_pushmem, counter_p);     // s_head  = counter
    `inst(3,    cpu.op_pushc,   1);             // s_head  = 1
    `inst(4,    cpu.op_sub);                    // s_head  = counter - 1
    `inst(5,    cpu.op_pop,     counter_p);     // counter = counter - 1

    /* if (counter < 0): goto exit */
    `inst(6,    cpu.op_pushc,   exit);          // s_head  = exit
    `inst(7,    cpu.op_js);                     // if (counter == 0): goto exit
    `inst(8,    cpu.op_pop,     tmp_p);         // else: sp = sp - 1
```

```
    /* LOOP BODY */
    `inst(9,    cpu.op_pushmem, cpu.in_addr);   // s_head  = x
    `inst(10,   cpu.op_pushc,   23);            // s_head  = 23
    `inst(11,   cpu.op_add);                    // s_head  = x + 23
    `inst(12,   cpu.op_pop,     tmp_p);         // tmp     = x + 23
    `inst(13,   cpu.op_pushmem, tmp_p);         // s_head  = x + 23
    `inst(14,   cpu.op_pushmem, tmp_p);         // s_head  = x + 23
    `inst(15,   cpu.op_add);                    // s_head  = (x + 23) * 2
    `inst(16,   cpu.op_pushc,   12);            // s_head  = 12
    `inst(17,   cpu.op_sub);                    // s_head  = ((X + 23) * 2) - 12
    `inst(18,   cpu.op_pop,     cpu.out_addr);  // y       = ((X + 23) * 2) - 12

    /* jump to start of loop */
    `inst(19,   cpu.op_pushc,   2);             // s_head  = 2
    `inst(20,   cpu.op_j);                      // goto    2
end

initial begin

    #10 rstN = 1;
    wait(cpu.pc == 7);
    in = 13;
    wait(cpu.pc == 19);
    $display("((%d + 23) * 2) - 12 = %d, error = %b", $signed(in), $signed(out), error);

    wait(cpu.pc == 7);
    in = 1;
    wait(cpu.pc == 19);
    $display("((%d + 23) * 2) - 12 = %d, error = %b", $signed(in), $signed(out), error);

    wait(cpu.pc == 7);
    in = -121;
    wait(cpu.pc == 19);
    $display("((%d + 23) * 2) - 12 = %d, error = %b", $signed(in), $signed(out), error);

    wait(cpu.pc == exit);
    $stop;
end

endmodule
```

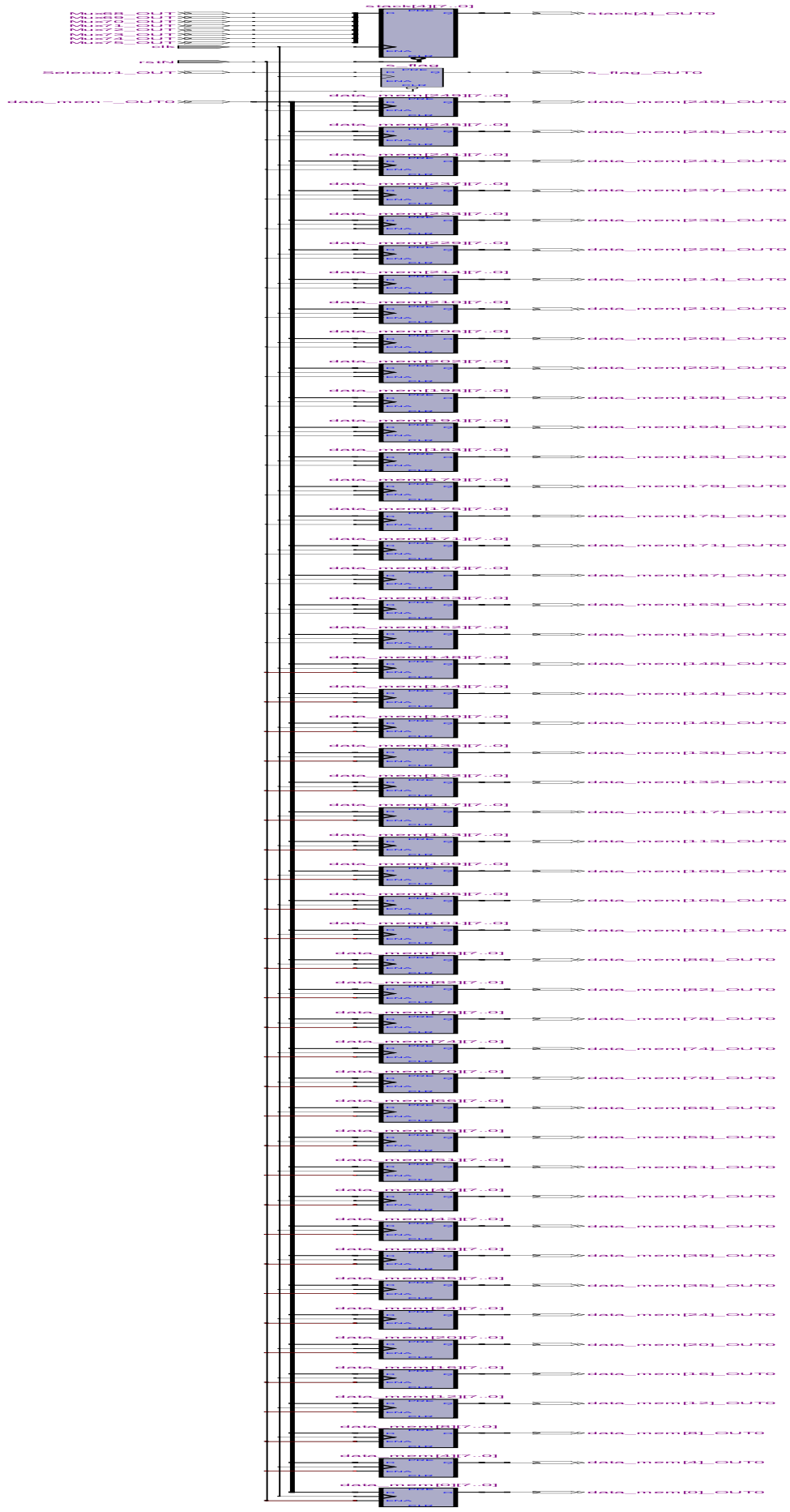
با شبیه سازی این ماژول توسط modelsim می توان نتایج زیر را مشاهده کرد:



همچنین با استفاده از ابزار سنتز کوارتوس می‌توانیم Rtl View این پردازنده ساده را هم ببینیم:



تصویر کامل آن بزرگ است در صفحه زیر و همچنین در فایل کنار این گزارش می توانید آن را ببینید.


stack[4][7..0]
stack[4]_OUT0
s_ram
s_flag_OUT0
data_mem[245][7..0]
data_mem[245]_OUT0
data_mem[241][7..0]
data_mem[241]_OUT0
data_mem[237][7..0]
data_mem[237]_OUT0
data_mem[233][7..0]
data_mem[233]_OUT0
data_mem[229][7..0]
data_mem[229]_OUT0
data_mem[214][7..0]
data_mem[214]_OUT0
data_mem[210][7..0]
data_mem[210]_OUT0
data_mem[206][7..0]
data_mem[206]_OUT0
data_mem[202][7..0]
data_mem[202]_OUT0
data_mem[198][7..0]
data_mem[198]_OUT0
data_mem[194][7..0]
data_mem[194]_OUT0
data_mem[183][7..0]
data_mem[183]_OUT0
data_mem[179][7..0]
data_mem[179]_OUT0
data_mem[175][7..0]
data_mem[175]_OUT0
data_mem[171][7..0]
data_mem[171]_OUT0
data_mem[167][7..0]
data_mem[167]_OUT0
data_mem[163][7..0]
data_mem[163]_OUT0
data_mem[152][7..0]
data_mem[152]_OUT0
data_mem[148][7..0]
data_mem[148]_OUT0
data_mem[144][7..0]
data_mem[144]_OUT0
data_mem[140][7..0]
data_mem[140]_OUT0
data_mem[136][7..0]
data_mem[136]_OUT0
data_mem[132][7..0]
data_mem[132]_OUT0
data_mem[117][7..0]
data_mem[117]_OUT0
data_mem[113][7..0]
data_mem[113]_OUT0
data_mem[109][7..0]
data_mem[109]_OUT0
data_mem[105][7..0]
data_mem[105]_OUT0
data_mem[101][7..0]
data_mem[101]_OUT0
data_mem[86][7..0]
data_mem[86]_OUT0
data_mem[82][7..0]
data_mem[82]_OUT0
data_mem[78][7..0]
data_mem[78]_OUT0
data_mem[74][7..0]
data_mem[74]_OUT0
data_mem[70][7..0]
data_mem[70]_OUT0
data_mem[66][7..0]
data_mem[66]_OUT0
data_mem[55][7..0]
data_mem[55]_OUT0
data_mem[51][7..0]
data_mem[51]_OUT0
data_mem[47][7..0]
data_mem[47]_OUT0
data_mem[43][7..0]
data_mem[43]_OUT0
data_mem[39][7..0]
data_mem[39]_OUT0
data_mem[35][7..0]
data_mem[35]_OUT0
data_mem[24][7..0]
data_mem[24]_OUT0
data_mem[20][7..0]
data_mem[20]_OUT0
data_mem[16][7..0]
data_mem[16]_OUT0
data_mem[12][7..0]
data_mem[12]_OUT0
data_mem[8][7..0]
data_mem[8]_OUT0
data_mem[4][7..0]
data_mem[4]_OUT0
data_mem[0][7..0]
data_mem[0]_OUT0