Моделиране на йерархична структура

[Модул](#_txtn0kyaw16b)

[Пример - module](#_1fob9te)

[Пример - йерархично описание](#_mf0wk586s0v2)

[Инстанцииране на модул](#_2et92p0)

[Пример](#_tyjcwt)

[Съответствие между портове и сигнали](#_1t3h5sf)

[Позиционно](#_qbauoypkmjwt)

[Поименно](#_55t9juw9n4cc)

[Съкратено (SystemVerilog)](#_si1pna9v6rcy)

[Generate оператори](#_cns5l81asw9m)

[Цикъл](#_khpn2y8q6v9g)

[Условни оператори](#_af5qsk8rw448)

# Модул

Mодулът е основната единица за описание на йерархия.

Дефинира се с оператор **module**. Състои се от:

* описание на портове, т.е. сигнали чрез които модулът се свързва с останалата част от схемата;
* функционално описание - какво прави този модул или от какви други модули се състои

## Пример - module

module shift\_register #(parameter N) (

input clock,

input [2:0] mode,

input [N-1:0] data\_in,

output logic [N-1:0] data\_out

);

always\_ff @(posedge clock)

case(mode)

3'b000: data\_out <= 0; // clear

3'b111: data\_out <= data\_in; // load

3'b100: data\_out <= data\_out << 1; // shift left

3'b001: data\_out <= data\_out >> 1; // shift right

default: data\_out <= data\_out; // hold

endcase

endmodule

## Пример - йерархично описание

| module lab1\_top (  input clk\_pin,  input rst\_pin,  output [15:0] led\_pins  );    logic clock;    lab\_1 #(.WIDTH(16)) **u1** (  .clock(clock),  .reset(rst\_pin),  .sr(led\_pins)  );    // 100 000 000 / 4 000 000 = 25 Hz  clk\_div #(.DIVIDE\_BY(4\_000\_000)) **div** (  .clock\_in(clk\_pin),  .clock\_out(clock),  .reset(rst\_pin)  );    endmodule |  |
| --- | --- |

| module lab1 #(parameter WIDTH = 8) (  input clock,  input reset,  output logic [WIDTH-1:0] sr  );  always\_ff @(posedge clock, posedge reset)  if(reset) sr <= 0;  else sr = {sr[WIDTH-2:0],~sr[WIDTH-1]};  endmodule | module clk\_div #(parameter DIVIDE\_BY = 2)(  input clock\_in,  input reset,  output logic clock\_out  );  integer counter;  const integer max\_count = DIVIDE\_BY / 2 - 1;  always\_ff @ (posedge clock\_in, posedge reset) begin  if (reset) begin  counter <= '0;  clock\_out <= '0;  end  else if (counter == max\_count) begin  counter <= '0;  clock\_out <= ~clock\_out;  end  else begin  counter <= counter + 1;  clock\_out <= clock\_out;  end  end  endmodule |
| --- | --- |

# Инстанцииране на модул

Поставяне на инстанция (instance) на модул в друг модул.

**module\_identifier** [parameter\_value\_assignment] **name\_of\_instance** ([list\_of\_port\_connections])

## Пример

lab\_1 **#(.**WIDTH**(**16**))** u1 **(**

**.**clock**(**clock**),**

**.**reset**(**rst\_pin**),**

**.**sr**(**led\_pins**)**

**);**

lab\_1 - идентификатор (име) на модула

#(.WIDTH(16)) - присвояване стойност на параметър

u1 - име на “инстанция” (instance)

.reset(rst\_pin) - свързване на порт **reset** към сигнала **rst\_pin**

## Съответствие между портове и сигнали

| module alu (      output [7:0] alu\_out,      output zero,      input [7:0] ain, bin,      input [2:0] opcode);      // RTL code for the alu module  endmodule  module accum (      output [7:0] dataout,      input [7:0] datain,      input clk, rst\_n);      // RTL code for the accumulator module  endmodule  module xtend (      output [7:0] dout,      input din,      input clk, rst\_n);      // RTL code for the sign-extension module  endmodule |  |
| --- | --- |

### Позиционно

| module alu\_accum1 (      output [15:0] dataout,      input [7:0] ain, bin,      input [2:0] opcode,      input clk, rst\_n);      logic [7:0] alu\_out;  **alu** u1 (alu\_out, , ain, bin, opcode);  **accum** u2 (dataout[7:0], alu\_out, clk, rst\_n);  **xtend** u3 (dataout[15:8], alu\_out[7], clk, rst\_n);  endmodule |  |
| --- | --- |

### Поименно

module alu\_accum2 (

    output [15:0] dataout,

    input [7:0] ain, bin,

    input [2:0] opcode,

    input clk, rst\_n);

    logic [7:0] alu\_out;

**alu** u1 (.alu\_out(alu\_out), .zero(), .ain(ain), .bin(bin), .opcode(opcode));

accum u2 (.dataout(dataout[7:0]), .datain(alu\_out), .clk(clk), .rst\_n(rst\_n));

xtend u3 (.dout(dataout[15:8]), .din(alu\_out[7]), .clk(clk), .rst\_n(rst\_n));

endmodule

### Съкратено (SystemVerilog)

| module alu\_accum3 (      output [15:0] dataout,      input [7:0] ain, bin,      input [2:0] opcode,      input clk, rst\_n);      logic [7:0] alu\_out;  alu u1 (.\*, .zero());  accum u3 (.\*, .dataout(dataout[7:0]), .datain(alu\_out));  xtend u3 (.\*, .dout(dataout[15:8]), .din(alu\_out[7]));  endmodule |  |
| --- | --- |

# Generate оператори

Описват регулярни структури. Изпълняват се по време на анализа (elaboration) на модела. Резултатите са определени преди да започне симулацията.

## Цикъл

Пример

| module gray2bin #(parameter SIZE = 8)(  output [SIZE-1:0] bin,  input [SIZE-1:0] gray  );  for (genvar i=0; i<SIZE; i=i+1)  assign bin[i] = ^gray[SIZE-1:i];  endmodule  module gray2bin\_5bit (  output [4:0] bin,  input [4:0] gray  );  gray2bin #(.SIZE(5)) uut (.\*);  endmodule |  |
| --- | --- |

module full\_adder (

input a, b, cin,

output s, cout

);

assign s = a ^ b ^ cin;

assign cout = (a & b) | (cin & a) | (cin & b);

endmodule

module ripple\_carry\_adder #(parameter N = 4) (

input carry\_in,

input [N-1:0] op1, op2,

output logic [N-1:0] sum,

output logic carry\_out

);

logic [N:0] carry;

assign carry[0] = carry\_in;

assign carry\_out = carry[N];

generate // optinal

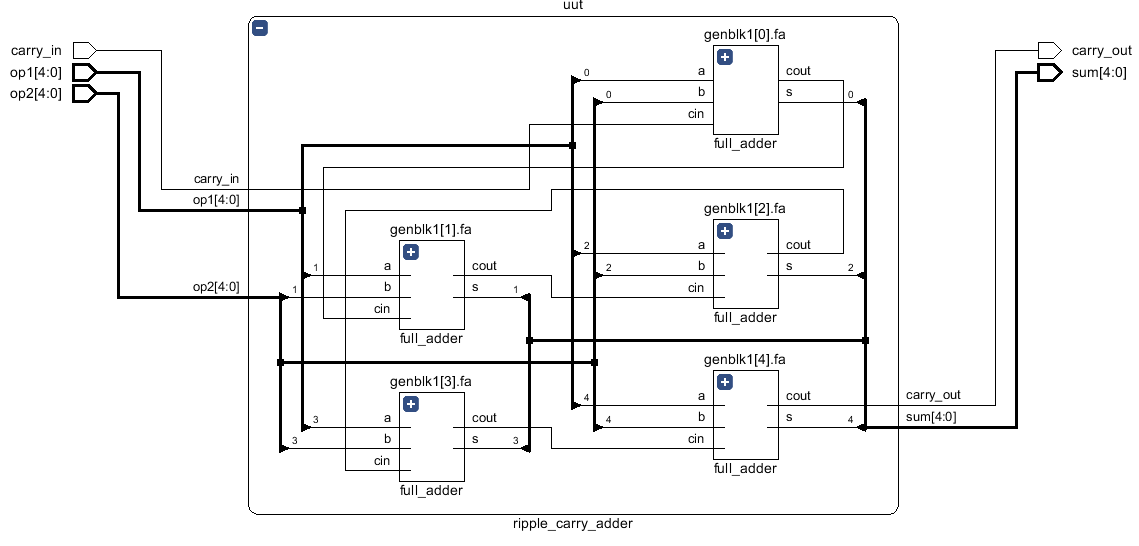
for(genvar i=0; i<N; i++) begin

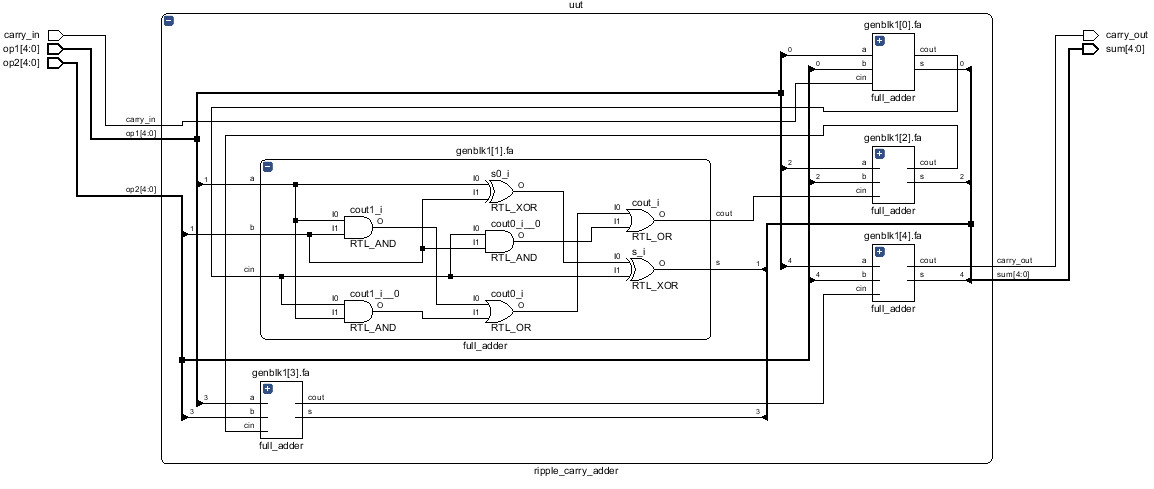
full\_adder fa (.a(op1[i]), .b(op2[i]),.cin(carry[i]),.s(sum[i]),.cout(carry[i+1]));

end

endgenerate // optional

endmodule





## Условни оператори

Пример

module multiplier #(parameter a\_width = 8, b\_width = 8) (

input [a\_width-1:0] a;

input [b\_width-1:0] b;

output [a\_width+b\_width-1:0] product

);

generate

if((a\_width < 8) || (b\_width < 8)) begin: mult

foo #(a\_width,b\_width) u1(a, b, product);

end

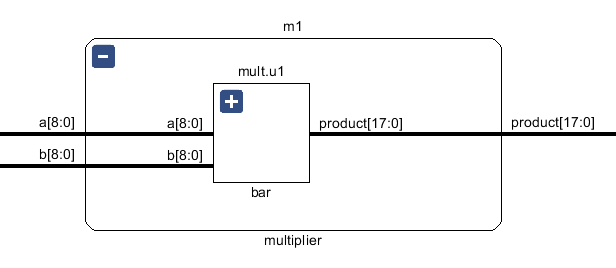
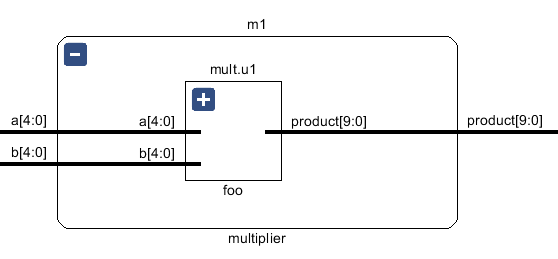
else begin: mult

bar #(a\_width,b\_width) u1(a, b, product);

end

endgenerate

endmodule



Пример

generate

case (WIDTH)

1: begin: adder // 1-bit adder implementation

adder\_1bit x1(co, sum, a, b, ci);

end

2: begin: adder // 2-bit adder implementation

adder\_2bit x1(co, sum, a, b, ci);

end

default:

begin: adder // others - carry look-ahead adder

adder\_cla #(WIDTH) x1(co, sum, a, b, ci);

end

endcase

// The hierarchical instance name is adder.x1

endgenerate