

ОСНОВЫ

VerilogHDL/SystemVerilog

(синтез и моделирование)



Описание модулей памяти

Использование встроенных модулей памяти

- ❑ Средства синтеза позволяют «распознавать» модули памяти с поведенческим описанием
- ❑ Для успешного «распознавания» следует придерживаться определенного стиля кодирования
- ❑ Ограничения связаны с возможностями архитектуры
 - ✓ СБИС Altera требуют синхронизации всех входов
 - ✓ Ограничения по режимам синхронизации
 - ✓ Ограничения по объему памяти
 - ✓ Поддержка режима Read-during-write
- ❑ Необходимо создать массив переменных для описания памяти

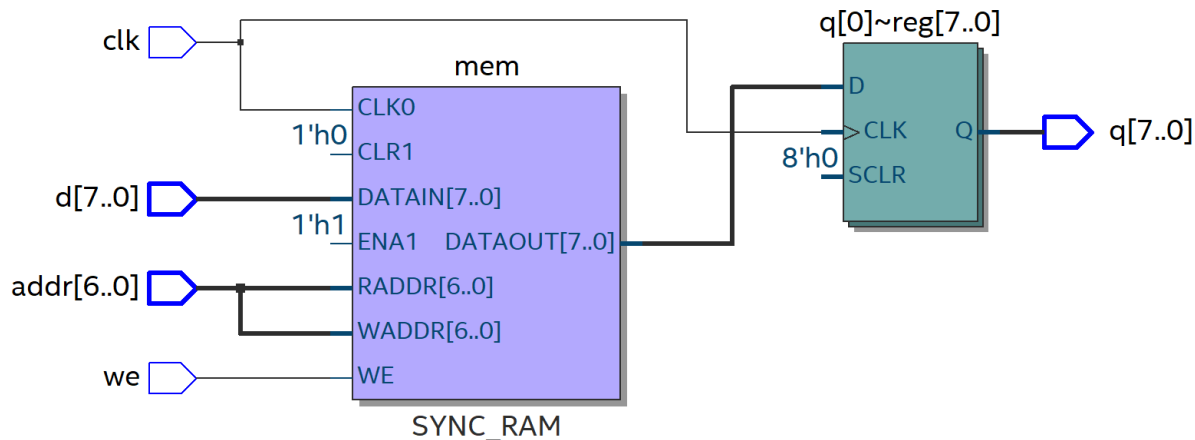
Описание Single-Port Memory *Old data read-during-write (1)*

```

1 module sp_ram_sync_rdwo (
2     output reg [7:0] q,
3     input [7:0] d,
4     input [3:0] addr,
5     input we, clk
6 );
7
8     reg [7:0] mem [0:15];
9
10    always @(posedge clk)
11    begin
12        if (we)
13            mem[addr] <= d;
14        q <= mem[addr];
15    end
16 endmodule
17

```

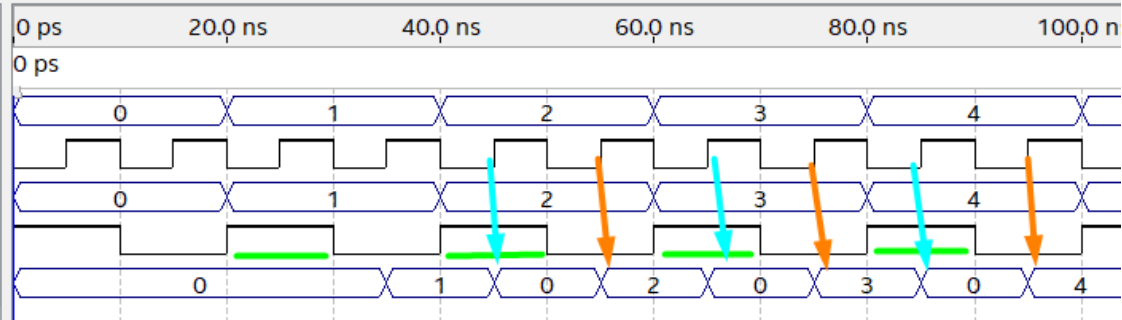
- **128 x 8 RAM**
 - synchronous write
 - synchronous read
- **Режим: Old data** read-during-write
 - т.к. используется **non-blocking** assignments



Описание Single-Port Memory *Old data read-during-write (2)*

```
1 module sp_ram_sync_rdwo (  
2     output reg [7:0] q,  
3     input [7:0] d,  
4     input [3:0] addr,  
5     input we, clk  
6 );  
7  
8 reg [7:0] mem [0:15];  
9  
10 always @(posedge clk)  
11 begin  
12     if (we)  
13         mem[addr] <= d;  
14     q <= mem[addr];  
15 end  
16 endmodule
```

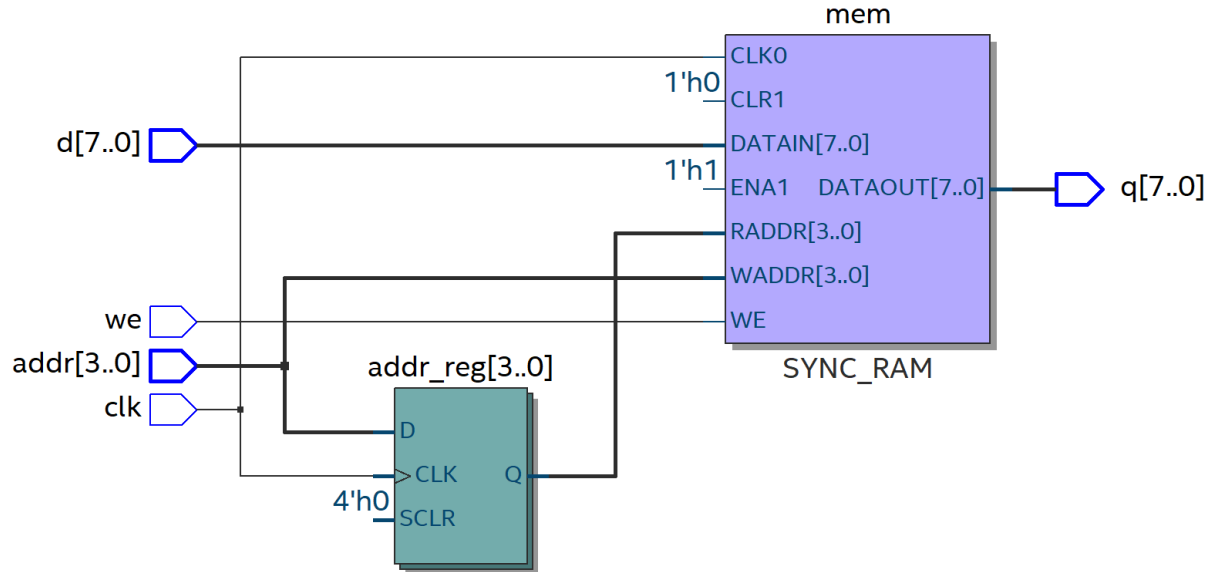
	Name	Value at 0 ps
in	> addr	U 0
in	clk	B 0
in	> d	U 0
in	we	B 1
out	> q	U 0



Описание Single-Port Memory **NEW** data read-during-write (1)

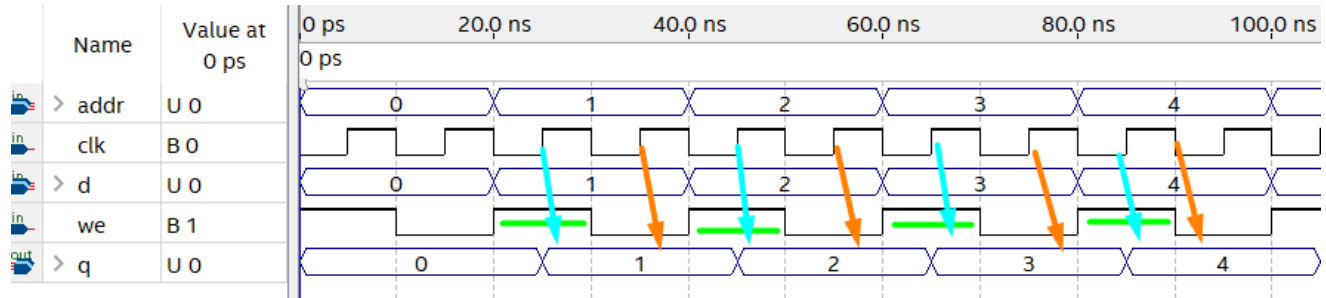
```
1 module sp_ram_sync_rdnw (  
2     output [7:0] q,  
3     input [7:0] d,  
4     input [3:0] addr,  
5     input we, clk  
6 );  
7  
8     reg [7:0] mem [0:15];  
9     reg [3:0] addr_reg;  
10  
11     always @(posedge clk)  
12     begin  
13         if (we)  
14             mem[addr] <= d;  
15         addr_reg <= addr;  
16     end  
17  
18     assign q = mem[addr_reg];  
19  
20 endmodule
```

- **128 x 8 RAM**
 - synchronous write
 - synchronous read
- **Режим: NEW data read-during-write**
 - т.к. используется запись адреса в промежуточный регистр



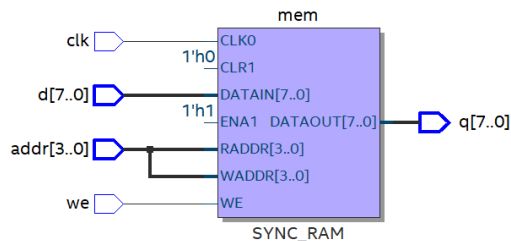
Описание Single-Port Memory *NEW data read-during-write (2)*

```
1 module sp_ram_sync_rdwn (
2     output [7:0] q,
3     input [7:0] d,
4     input [3:0] addr,
5     input we, clk
6 );
7
8 reg [7:0] mem [0:15];
9 reg [3:0] addr_reg;
10
11 always @(posedge clk)
12 begin
13     if (we)
14         mem[addr] <= d;
15     addr_reg <= addr;
16 end
17
18 assign q = mem[addr_reg];
19
20 endmodule
```



Описание Single-Port Memory ОШИБКА (1)

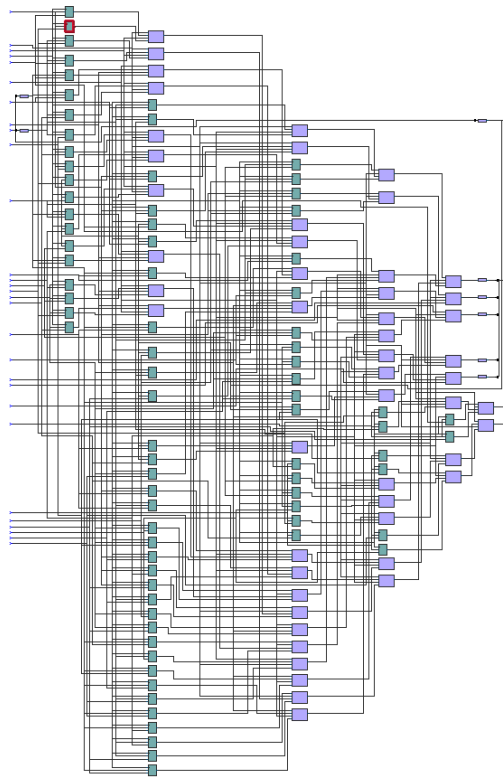
```
1 module sp_ram_async_read (  
2     output [7:0] q,  
3     input [7:0] d,  
4     input [3:0] addr,  
5     input we, clk  
6 );  
7  
8 reg [7:0] mem [0:15];  
9  
10 always @(posedge clk)  
11     if (we)  
12         mem[addr] <= d;  
13  
14 assign q = mem[addr];  
15  
16 endmodule
```



- **128 x 8 RAM**
 - synchronous write
 - asynchronous read
- **Не может быть реализован на базе встроенных модулей памяти т.к. чтение должно быть синхронным**
 - Будет реализован на РЕГИСТРАХ

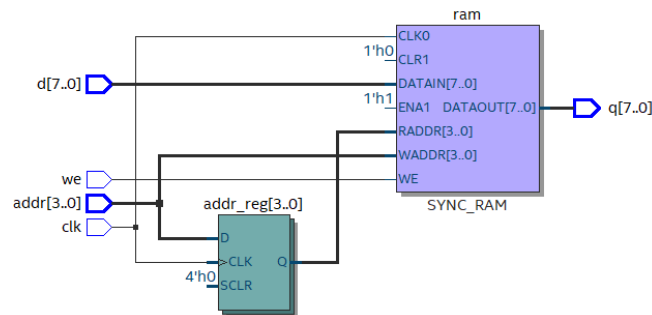
Flow Status	Successful - Thu Oct 21 16:13:57 2021
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	lab1
Top-level Entity Name	sp_ram_async_read
Family	Cyclone IV E
Device	EP4CE6E22C8
Timing Models	Final
Total logic elements	240
Total registers	128
Total pins	22
Total virtual pins	0
Total memory bits	0

Описание Single-Port Memory ОШИБКА (2)



Описание Single-Port Memory **NEW** data read-during-write PARAM

```
1 // Single port RAM with single read/write address
2 module single_port_ram_par
3   #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=4)
4   (
5     input [(DATA_WIDTH-1):0] d,
6     input [(ADDR_WIDTH-1):0] addr,
7     input we, clk,
8     output [(DATA_WIDTH-1):0] q
9   );
10  // Declare the RAM variable
11  reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
12  // Variable to hold the registered read address
13  reg [ADDR_WIDTH-1:0] addr_reg;
14
15  always @ (posedge clk)
16  begin
17    // write
18    if (we)
19      ram[addr] <= d;
20
21    addr_reg <= addr;
22  end
23  // Assignment implies read returns NEW data.
24  assign q = ram[addr_reg];
25
26 endmodule
```

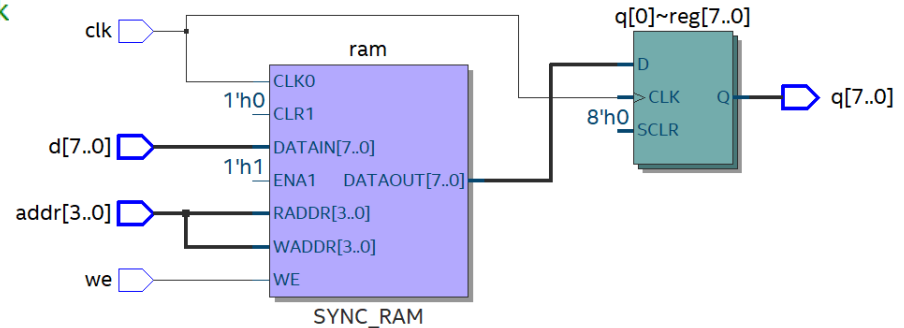


Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	lab1
Top-level Entity Name	single_port_ram_par
Family	Cyclone IV E
Device	EP4CE6E22C8
Timing Models	Final
Total logic elements	0
Total registers	0
Total pins	22
Total virtual pins	0
Total memory bits	128



Описание SPM *OLD* data read-during-write PARAM INIT (1)

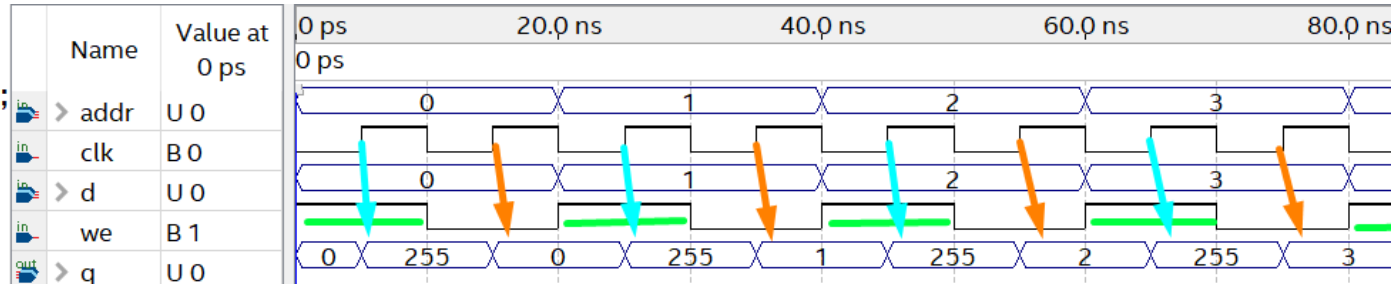
```
1 // Single port RAM with single read/write address and
2 // initial contents specified with an initial block
3 module single_port_ram_param_with_init
4 # (parameter DATA_WIDTH=8, parameter ADDR_WIDTH=4)
5   input [(DATA_WIDTH-1):0] d,
6   input [(ADDR_WIDTH-1):0] addr,
7   input we, clk,
8   output reg [(DATA_WIDTH-1):0] q);
9
10  reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
11
12  // Specify the initial contents.
13  initial begin : INIT
14    integer i;
15    for(i = 0; i < 2**ADDR_WIDTH; i = i + 1)
16      ram[i] = {DATA_WIDTH{1'b1}};
17  end
18
19  always @ (posedge clk)
20  begin
21    if (we)
22      ram[addr] <= d;
23    q <= ram[addr];
24  end
25 endmodule
```



Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	lab1
Top-level Entity Name	single_port_ram_param_with_init
Family	Cyclone IV E
Device	EP4CE6E22C8
Timing Models	Final
Total logic elements	0
Total registers	0
Total pins	22
Total virtual pins	0
Total memory bits	128

Описание SPM *OLD* data read-during-write PARAM INIT (2)

```
1 // Single port RAM with single read/write address and
2 // initial contents specified with an initial block
3 module single_port_ram_param_with_init
4   #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=4)
5   (input [(DATA_WIDTH-1):0] d,
6    input [(ADDR_WIDTH-1):0] addr,
7    input we, clk,
8    output reg [(DATA_WIDTH-1):0] q);
9
10  reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
11
12  // Specify the initial contents.
13  initial begin : INIT
14    integer i;
15    for(i = 0; i < 2**ADDR_WIDTH; i = i + 1)
16      ram[i] = {DATA_WIDTH{1'b1}};
17  end
18
19  always @ (posedge clk)
20  begin
21    if (we)
22      ram[addr] <= d;
23    q <= ram[addr];
24  end
25 endmodule
```



Использование Simple Dual-Port Memory

```
module sdp_sc_ram (  
    output reg [7:0] q,  
    input [7:0] d,  
    input [6:0] wr_addr, rd_addr,  
    input we, clk  
);
```

```
reg [7:0] mem [0:127];
```

```
always @(posedge clk) begin  
    if (we)  
        mem[wr_addr] <= d;  
        q <= mem[rd_addr];  
end  
  
endmodule
```

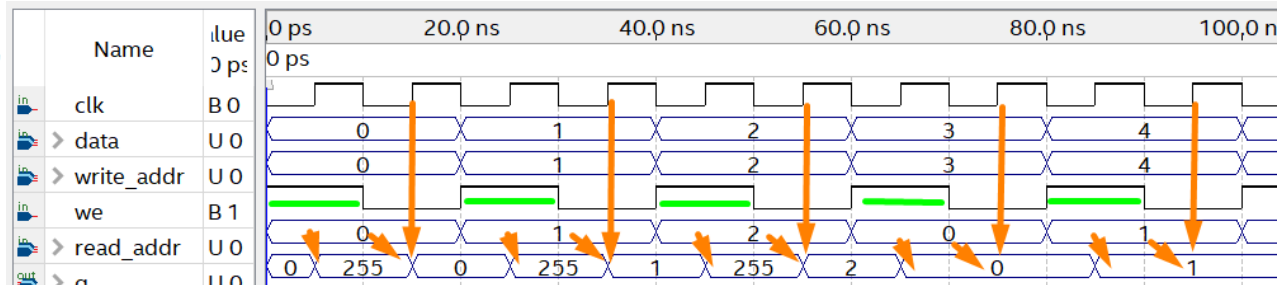
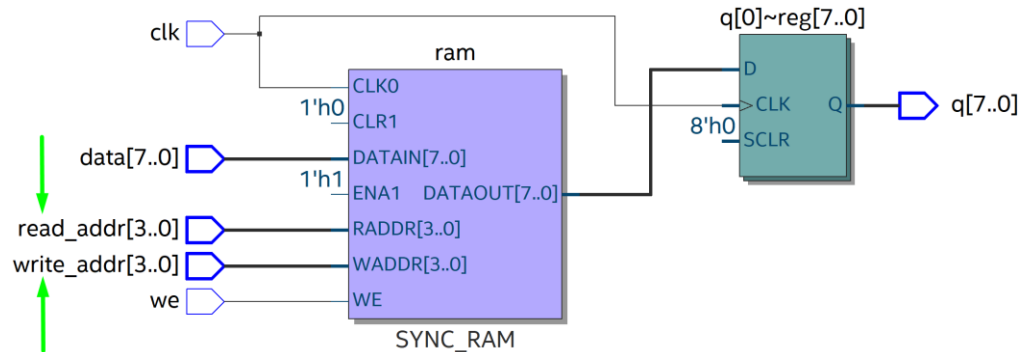
- **128 x 8 RAM** один тактовый сигнал; **simple dual-port** (отдельно read & write адреса)
- **Режим: old data** read-during-write behavior
 - Режим: New data read-during-write реализуется при использовании blocking assignments (требуется дополнительной логики для реализации)

Описание Simple Dual Port RAM single read/write clock

```

1 // Simple Dual Port RAM with
2 // single read/write clock
3 module simple_dual_port_ram_single_clock
4 #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=4)
5 (input [(DATA_WIDTH-1):0] data,
6 input [(ADDR_WIDTH-1):0] read_addr, write_addr,
7 input we, clk,
8 output reg [(DATA_WIDTH-1):0] q);
9 // Declare the RAM variable
10 reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
11
12 // Specify the initial contents.
13 initial begin : INIT
14     integer i;
15     for(i = 0; i < 2**ADDR_WIDTH; i = i + 1)
16         ram[i] = {DATA_WIDTH{1'b1}};
17 end
18
19 always @ (posedge clk)
20 begin
21     // Write
22     if (we)
23         ram[write_addr] <= data;
24     // Read
25     // (if read_addr == write_addr,
26     q <= ram[read_addr];
27 end
28 endmodule

```



Использование True Dual-Port Memory

```
module dp_dc_ram (  
    output reg [7:0] q_a, q_b,  
    input [7:0] data_a, data_b,  
    input [6:0] addr_a, addr_b,  
    input clk_a, clk_b, we_a, we_b  
);
```

```
reg [7:0] mem [0:127];
```

```
always @(posedge clk_a)  
begin  
    if (we_a)  
        mem[addr_a] <= data_a;  
    q_a <= mem[addr_a];  
end
```

```
always @(posedge clk_b)  
begin  
    if (we_b)  
        mem[addr_b] <= data_b;  
    q_b <= mem[addr_b];  
end  
endmodule
```

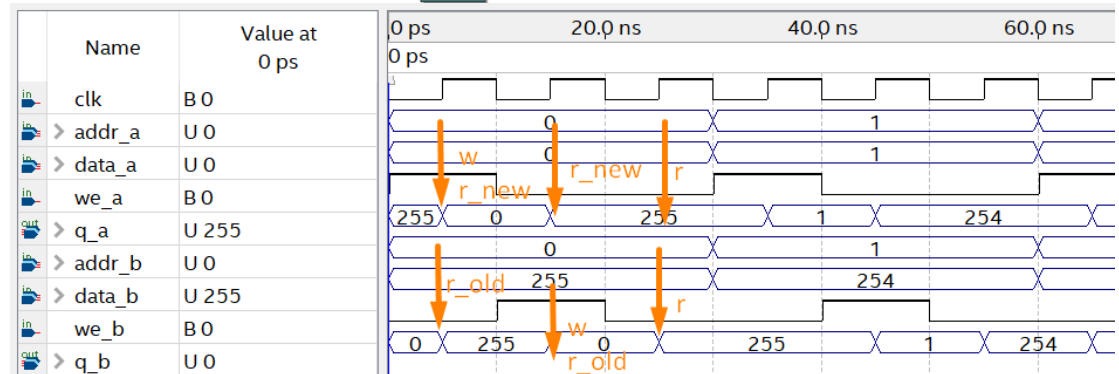
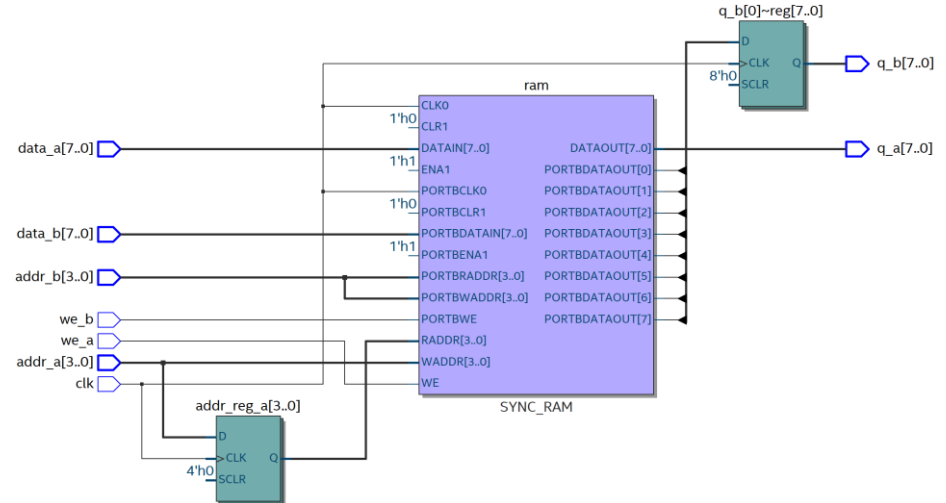
- **128 x 8 RAM** с двумя тактовыми сигналами; **true dual-port** (два набора адресов)
- **Режим: Old data same-port** read-during-write
 - Режим: New data **same-port** read-during-write может быть реализован с использованием blocking assignments (поддерживается не во всех СБИС)
- Поведение для Mixed port behavior (read and write on different ports for same address) не определено при использовании двух тактовых сигналов.

Описание True Dual Port RAM with single clock

```

1 // True Dual Port RAM with single clock
2 module true_dual_port_ram_single_clock
3   #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=4)
4   (input [(DATA_WIDTH-1):0] data_a, data_b,
5    input [(ADDR_WIDTH-1):0] addr_a, addr_b,
6    input we_a, we_b, clk,
7    output reg [(DATA_WIDTH-1):0] q_a, q_b);
8   // Declare the RAM variable
9   reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
10  // Variable to hold the registered read address port_A
11  reg [ADDR_WIDTH-1:0] addr_reg_a;
12  // Specify the initial contents.
13  initial begin : INIT
14    integer i;
15    for(i = 0; i < 2**ADDR_WIDTH; i = i + 1)
16      ram[i] = {DATA_WIDTH{1'b1}};
17  end
18  // Port A
19  always @ (posedge clk)
20  begin
21    if (we_a)
22      ram[addr_a] <= data_a;
23      addr_reg_a <= addr_a;
24  end
25  always @*
26  q_a <= ram[addr_reg_a];
27  // Port B
28  always @ (posedge clk)
29  begin
30    if (we_b)
31      ram[addr_b] <= data_b;
32      q_b <= ram[addr_b];
33  end
34 endmodule

```



Задание начального содержимого модулей памяти

```
module ram_init (  
    output reg [7:0] q,  
    input [7:0] d,  
    input [6:0] wr_addr, rd_addr,  
    input we, clk  
);
```

```
reg [7:0] mem [0:127];
```

```
initial  
    $readmemb("ram.dat", mem);
```

```
always @(posedge clk) begin  
    if (we)  
        mem[wr_addr] <= d;  
    q <= mem[rd_addr];  
end  
  
endmodule
```

- Используйте `$readmemb` или `$readmemh` system tasks для задания начального содержимого модулей памяти
- Данные инициализации хранятся в файле `.dat` преобразуемом в `.mif` (Altera memory initialization file)
- Содержимое `.mif` загружается в СБИС при конфигурации микросхемы
- Альтернатива: использование `initial block` и цикла для задания значений элементам массива, описывающего память

- Каждое отдельное число в файле заносится по отдельному адресу в памяти
 - ✓ Пробелы и комментарии отделяют числа.
- Примеры
 - ✓ `$readmemb ("<file_name>", <memory_name>);`
 - ✓ `$readmemh ("<file_name>", <memory_name>);`

ram.dat

```
0000_0000  
0000_0101  
0000_1010  
0000_1111  
0001_0100  
0001_1001  
0001_1110  
0010_0011  
0010_1000  
// address 20 hex  
@20  
0000_0000  
0000_0101
```

Файл .dat

- ❑ Каждое отдельное число в файле заносится по отдельному адресу в памяти

ram.dat

- ✓ Пробелы и комментарии отделяют числа.

- ❑ Примеры

- ✓ `$readmemb ("<file_name>", <memory_name>);`
- ✓ `$readmemh ("<file_name>", <memory_name>);`
- ✓ `$readmemb ("<file_name>", <memory_name>, <mem_start_addr>, <mem_finish_addr>);`
 - **<mem_start_addr>**, **<mem_finish_addr>** -опционально
 - задает начальный и конечный адрес

```
0000_0000
0000_0101
0000_1010
0000_1111
0001_0100
0001_1001
0001_1110
0010_0011
0010_1000
// address 20 hex
@20
0000_0000
0000_0101
```

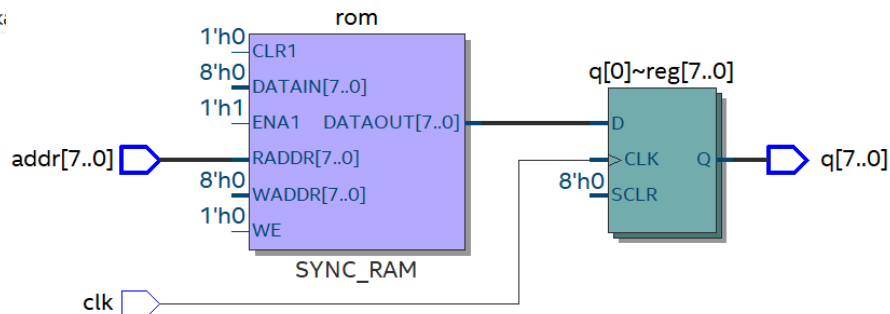
Описание Single Port ROM

```
1 // Single Port ROM
2 module single_port_rom
3   #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=8)
4   (input [(ADDR_WIDTH-1):0] addr,
5    input clk,
6    output reg [(DATA_WIDTH-1):0] q);
7   // Declare the ROM variable
8   reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];
9   // Initialize the ROM with $readmemb.
10  // Put the memory contents
11  // in the file single_port_rom_init.txt.
12  // without this file, this design will not compile.
13  initial
14    $readmemb("single_port_rom_init.txt", rom);
15  always @ (posedge clk)
16  begin
17    q <= rom[addr];
18  end
19 endmodule
```

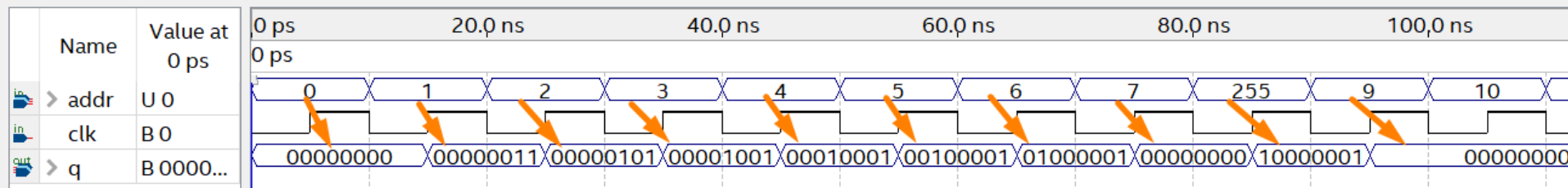
single_port_rom_init.txt – Блокнот

Файл Правк

```
00000000
00000011
00000101
00001001
00010001
00100001
01000001
01000001
@ff
10000001
```



Master Time Bar: 0 ps Pointer: 77.77 ns Interval: 77.77 ns Start: 0 ps End: 0 ps




Описание Dual Port ROM

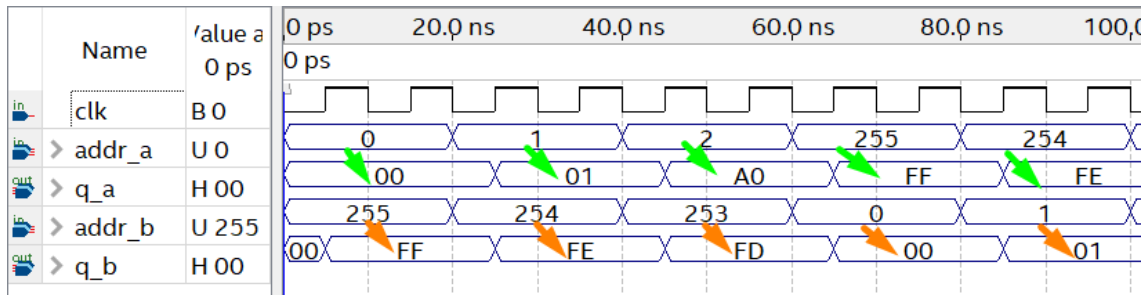
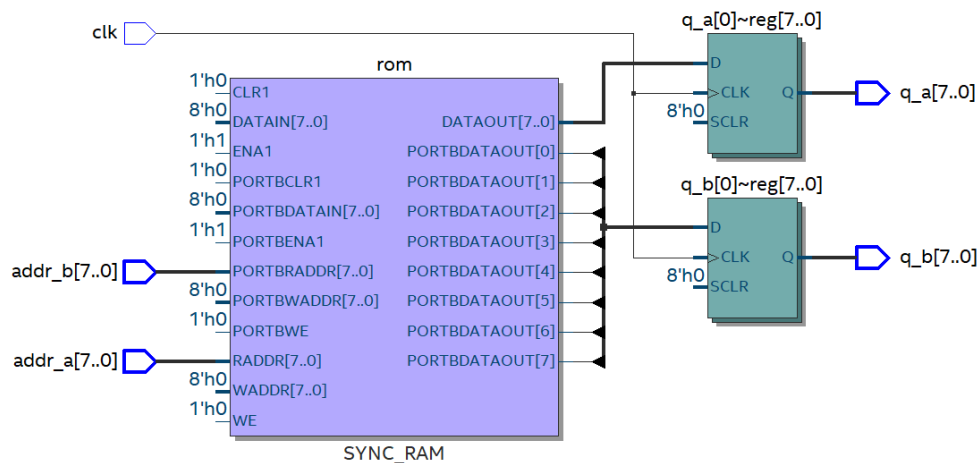
```

1 // Dual Port ROM
2 module dual_port_rom
3   #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=8)
4   (input [(ADDR_WIDTH-1):0] addr_a, addr_b,
5    input clk,
6    output reg [(DATA_WIDTH-1):0] q_a, q_b);
7   // Declare the ROM variable
8   reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];
9   // Initialize the ROM with $readmemh from the file
10  // dual_port_rom_init.txt.
11  // Without this file this design will not compile.
12  initial
13    $readmemh("dual_port_rom_init.txt", rom);
14  always @ (posedge clk)
15  begin
16    q_a <= rom[addr_a];
17    q_b <= rom[addr_b];
18  end
19 endmodule

```

 dual_port_rom_init.txt

00
01
A0
@FC
FC
FD
FE
FF



Не поддерживаемые сигналы управления

- ❑ e.g. clearing RAM contents with reset

```
module ram_unsupported (  
    output reg [7:0] q,  
    input [7:0] d,  
    input [6:0] addr,  
    input we, clk  
);  
  
reg [7:0] mem [0:127];  
  
always @(posedge clk, negedge aclr_n)  
begin  
    if (!aclr_n) begin  
        mem[addr] <= 0;  
        q <= 0;  
    end  
    else if (we) begin  
        mem[addr] <= d;  
        q <= mem[addr];  
    end  
end  
  
endmodule
```

- Модуль памяти (элемент модуля памяти) нельзя очистить сигналом *reset*
- Будет реализовано на логических элементах
- Рекомендации:
 1. Избегать сигнала сброса при описании RAM read или write
 2. Внимательно относиться к использованию других управляющих сигналов (например: разрешение работы)



Создание экземпляров модулей, задач, функций, операторов, примитивов

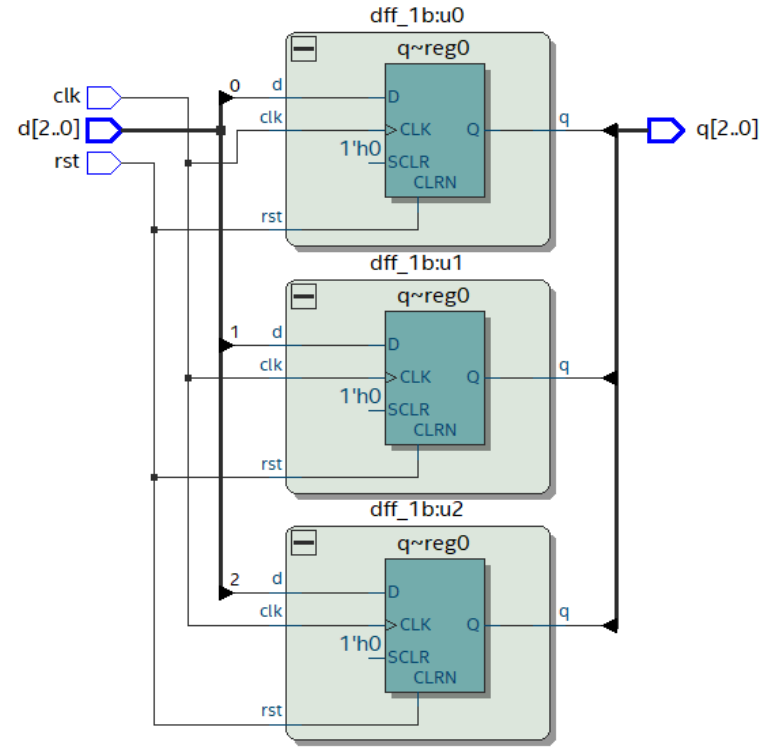
Создание нескольких экземпляров модулей

- ❑ Создание нескольких экземпляров модулей или функций
 - ✓ В Verilog '95 определено три способа
 - Отдельные объявления
 - Список экземпляров
 - Массив экземпляров

Несколько экземпляров модулей (список экземпляров)

```
1 module dff_1b
2   ( input d,
3     input clk,
4     input rst,
5     output reg q);
6   always @(posedge clk, posedge rst)
7     if (rst) q <= 1'b0;
8     else q <= d;
9 endmodule
```

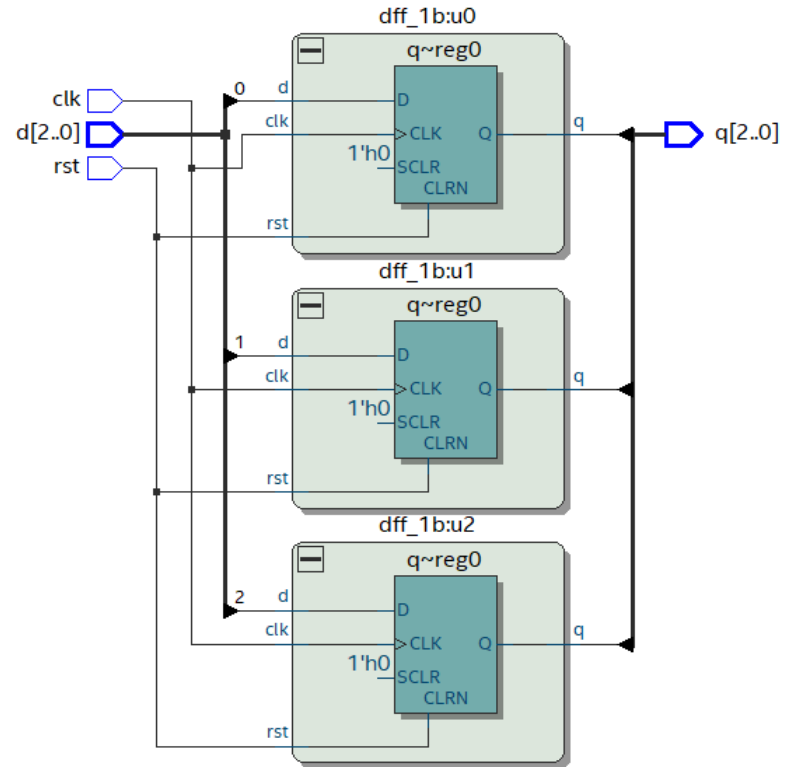
```
1 module multiple_comma_separated_list
2   ( input [2:0] d,
3     input clk,
4     input rst,
5     output [2:0] q);
6
7   dff_1b
8   u0 (d[0], clk, rst, q[0]),
9   u1 (d[1], clk, rst, q[1]),
10  u2 (d[2], clk, rst, q[2]);
11
12 endmodule
```



Несколько экземпляров модулей (массив экземпляров)

```
1 module dff_1b
2   ( input d,
3     input clk,
4     input rst,
5     output reg q);
6   always @(posedge clk, posedge rst)
7     if (rst) q <= 1'b0;
8     else q <= d;
9 endmodule
```

```
1 module multiple_instance_array
2   ( input [2:0] d,
3     input clk,
4     input rst,
5     output [2:0] q);
6
7   dff_1b u[2:0] (d[2:0], clk, rst, q[2:0]);
8
9 endmodule
```



Создание нескольких экземпляров модулей

- ❑ Создание нескольких экземпляров модулей или функций
 - ✓ В Verilog '95 определено три способа
 - Отдельные объявления
 - Список экземпляров
 - Массив экземпляров
 - ✓ Verilog '01 дополнительно введены операторы **generate ... endgenerate**, которые могут быть использованы для переопределения
 - параметров – `parameter`
 - операторов непрерывного назначения сигналов
 - для `initial\always` блоков
 - функций, задач.
 - примитивов

NOTE: порты модуля не могут быть использованы в generate ... endgenerate

Методы использования generate ... endgenerate

- ❑ В Verilog '01 определены три метода
 - ✓ generate - loop
 - ✓ generate – conditional (if-else)
 - ✓ generate – case
- ❑ В Verilog '01 введен тип genvar
 - ✓ genvar – положительное целое
 - значение должно быть константой после компиляции (Elaboration)
 - ✓ если genvar объявлен в блоке generate, то это - локальная переменная блока
 - ✓ если genvar объявлен вне блоков generate, то это переменная, которая может быть использована во всех блоках genvar

generate ... endgenerate (метод generate - loop)

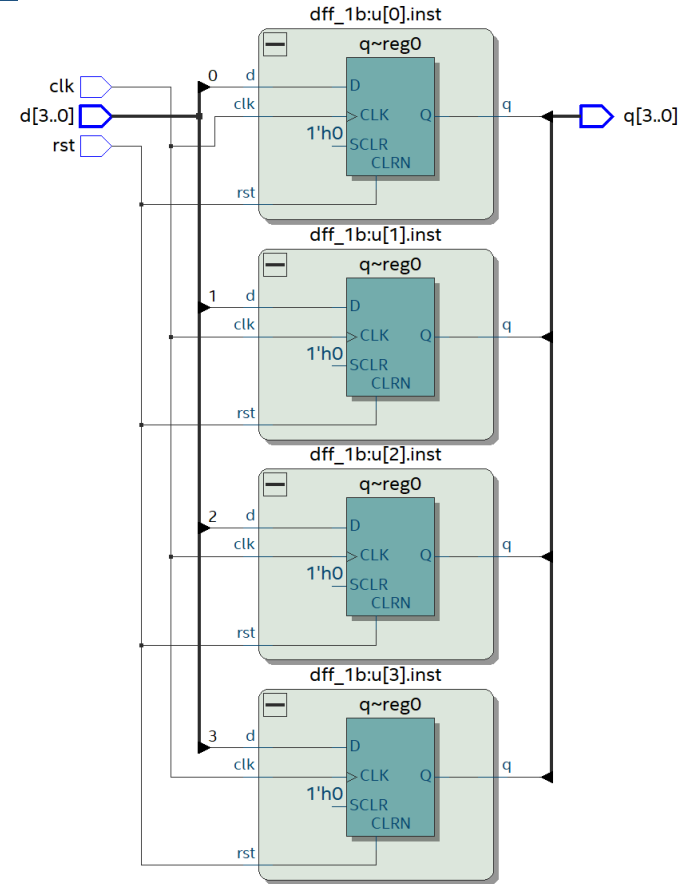
- ❑ Метод generate – loop определен для цикла **for loop**
- ❑ Переменная цикла должна быть объявлена как genvar
- ❑ Цикл должен содержать begin ... end не зависимо от количества операторов в цикле
- ❑ Цикл должен быть поименован

```
genvar <genvar_id>;  
for(<genvar_id> = <constant_expr>; <constant_expr>; <genvar_id> = <constant_expr>)  
begin : <required_block_name>  
    // Generate Items  
end
```

generate ... endgenerate (метод generate - loop)

```
1 module dff_1b
2   ( input d,
3     input clk,
4     input rst,
5     output reg q );
6   always @(posedge clk, posedge rst)
7     if (rst) q <= 1'b0;
8     else q <= d;
9 endmodule
```

```
1 module multiple_generate_loop
2   #(parameter N=4)
3   ( input [N-1:0] d,
4     input clk,
5     input rst,
6     output [N-1:0] q );
7
8   generate
9     genvar gi;
10    for (gi = 0; gi < N; gi = gi + 1) begin: u
11      dff_1b inst (d[gi], clk, rst, q[gi]);
12    end
13  endgenerate
14
15 endmodule
```



generate ... endgenerate (метод generate – if-else)

- ❑ Метод generate – if-else (conditional) определен для оператора **if-else**
- ❑ В операторе каждый выбор (if, else) должен содержать begin ... end
- ❑ В операторе каждый выбор (if, else) должен быть поименован (в некоторых случаях можно не именовать)

```
// If
if (<constant_expression>)
begin : <if_block_name>
    // Generate Items
end

// If-Else
if(<constant_expression>)
begin : <if_block_name>
    // Generate Items
end
else
begin : <else_block_name>
    // Generate Items
end

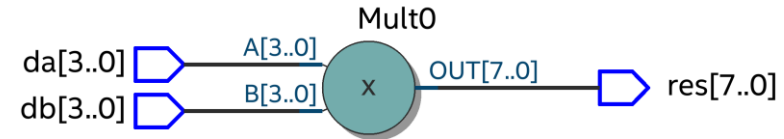
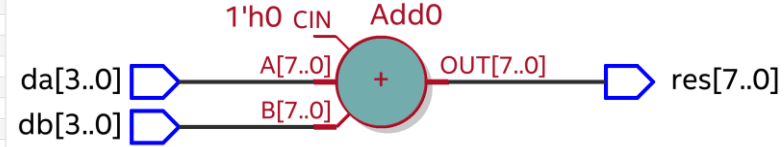
//NOTE:Block names are optional but recommended.
```

generate ... endgenerate (метод generate – if-else)

```
1 module multiple_conditional
2   #(parameter N=4, op = "add")
3   (input signed [N-1:0] da, db,
4    output reg signed [2*N-1:0] res);
5
6   generate
7   always @*
8   if (op == "add") begin :adder
9     res = da + db; end
10  else begin :mult
11    res = da * db; end
12  endgenerate
13
14 endmodule
```

```
1 module multiple_conditional
2   #(parameter N=4, op = "xxx") // "add"
3   (input signed [N-1:0] da, db,
4    output reg signed [2*N-1:0] res);
5
6   generate
7   always @*
8   if (op == "add") begin //:adder
9     res = da + db; end
10  else begin //:mult
11    res = da * db; end
12  endgenerate
13
14 endmodule
```

Input Port	Fan-in Node
CIN	GND
▼ A[7..0]	
[0]	da[0]
[1]	da[1]
[2]	da[2]
[3]	da[3]
[4]	da[3]
[5]	da[3]
[6]	da[3]
[7]	da[3]
▼ B[7..0]	
[0]	db[0]
[1]	db[1]
[2]	db[2]
[3]	db[3]
[4]	db[3]
[5]	db[3]
[6]	db[3]
[7]	db[3]



generate ... endgenerate (метод generate – case)

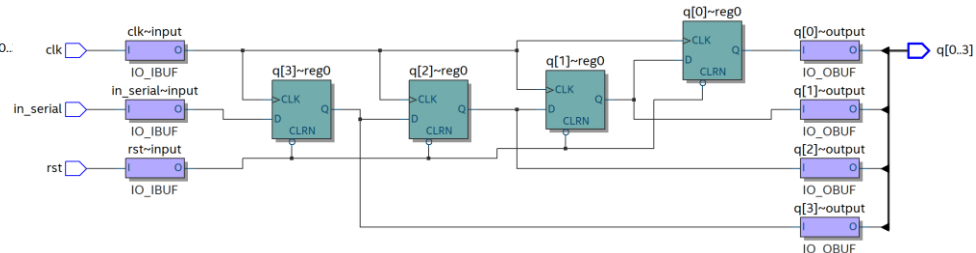
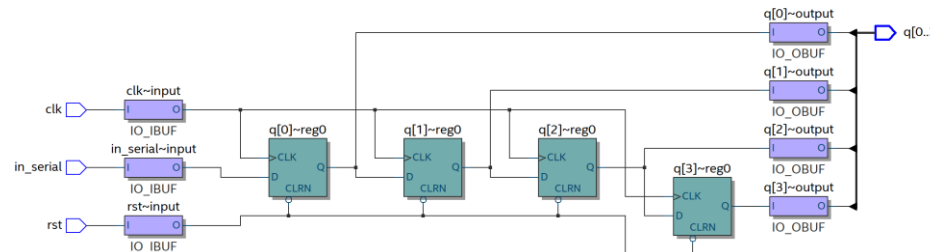
- ❑ Метод generate – case определен для оператора **case**
- ❑ В операторе каждый выбор должен содержать begin ... end
- ❑ В операторе каждый выбор должен быть поименован (в некоторых случаях можно не именовать)

```
case(<constant_expr>
|<constant_expr>:
|     begin : <block_name>
|           // Generate Items
|     end
|<constant_expr>:
|     begin : <block_name>
|           // Generate Items
|     end
|// ...
|default:
|     begin : <block_name>
|           // Generate Items
|     end
|endcase
|
|// NOTE: Block names are optional but recommended.
```


generate ... endgenerate (метод generate – case)

```
1 module multiple_case
2   #(parameter N=4, dir = "left")
3   input  in_serial,
4   input  clk, rst,
5   output reg [N-1:0] q;
6
7   generate
8   if (dir == "left")
9   begin :left_sh
10    always @(posedge clk, posedge rst)
11      if (rst) q <= {N{1'b0}};
12      else q <= {q[N-2:0], in_serial};
13    end
14  else
15  begin :right_sh
16    always @(posedge clk, posedge rst)
17      if (rst) q <= {N{1'b0}};
18      else q <= { in_serial, q[N-1:1],};
19    end
20  end
21 endgenerate
22
23
24 endmodule
```

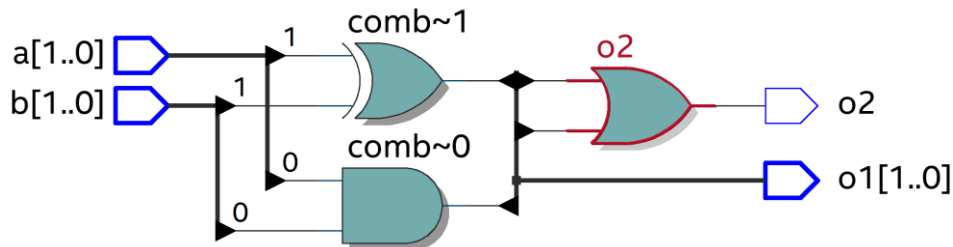
```
1 module multiple_case
2   #(parameter N=4, dir = "xxx")// "left"
3   input  in_serial,
4   input  clk, rst,
5   output reg [N-1:0] q;
6
7   generate
8   if (dir == "left")
9   begin //:left_sh
10    always @(posedge clk, posedge rst)
11      if (rst) q <= {N{1'b0}};
12      else q <= {q[N-2:0], in_serial};
13    end
14  else
15  begin //:right_sh
16    always @(posedge clk, posedge rst)
17      if (rst) q <= {N{1'b0}};
18      else q <= { in_serial, q[N-1:1],};
19    end
20  end
21 endgenerate
22
23
24 endmodule
```



Иерархическое именование цепей и компонентов

- Цепи, компоненты, модули, функции, задачи, созданные оператором `generate` имеют иерархические имена.

```
1 module generate_labels
2   ( input  [1:0] a, b,
3     output [1:0] o1,
4     output o2);
5
6   genvar i;
7
8   generate
9     for(i = 0; i < 2; i = i + 1)
10    begin: GEN_LOOP
11      case(i)
12      0:
13        begin : GEN_CASE0
14          wire q = a[0] & b[0];
15          assign o1[0] = q;
16        end
17      1:
18        begin : GEN_CASE1
19          wire q = a[1] ^ b[1];
20          assign o1[1] = q;
21        end
22      endcase
23    end
24  endgenerate
25  // access objects defined inside the generate loop
26  // by using their hierarchical name
27  assign o2 = GEN_LOOP[0].GEN_CASE0.q | GEN_LOOP[1].GEN_CASE1.q;
28
29 endmodule
```



Input Port	Fan-in Node
IN0	comb~0
IN1	comb~1

Совместное использование конструкций generate

```
1 module generate_design
2   #(parameter N = 4)
3   input a, b,
4   output [N * (N + 1) / 2 - 1 : 0] o);
5
6   genvar i, j;
7
8   generate for(i = 0; i < N; i = i + 1) begin: GEN_LOOP
9     // Width of q on each iteration depends on genvar i
10    (* keep = 1*) reg [i:0] q;
11
12    if(i % 2 == 0) begin: GEN_TRUE
13      always@* q[i:0] = {i+1{a}};
14    end
15    else begin : GEN_FALSE
16      always@* q[i:0] = {i+1{b}};
17    end
18  end
19 endgenerate
20
21 generate for(j = 0; j < N; j = j + 1) begin : GEN_LOOP2
22   assign o[j + j * (j + 1) / 2 : 0 + j * (j + 1) / 2] = GEN_LOOP[j].q[j:0];
23 end
24 endgenerate
25 endmodule
26
```

Input Port	Fan-in Node
✓ DATAIN	
[0]	GEN_LOOP[0].q[0]
[1]	GEN_LOOP[1].q[0]
[2]	GEN_LOOP[1].q[1]
[3]	GEN_LOOP[2].q[0]
[4]	GEN_LOOP[2].q[1]
[5]	GEN_LOOP[2].q[2]
[6]	GEN_LOOP[3].q[0]
[7]	GEN_LOOP[3].q[1]
[8]	GEN_LOOP[3].q[2]
[9]	GEN_LOOP[3].q[3]

