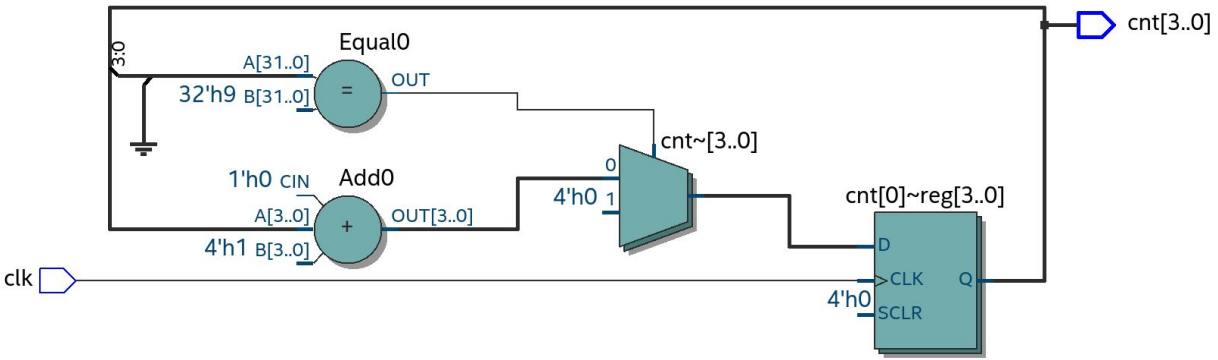
Мультиплексор. Декодер. Светодиодный индикатор

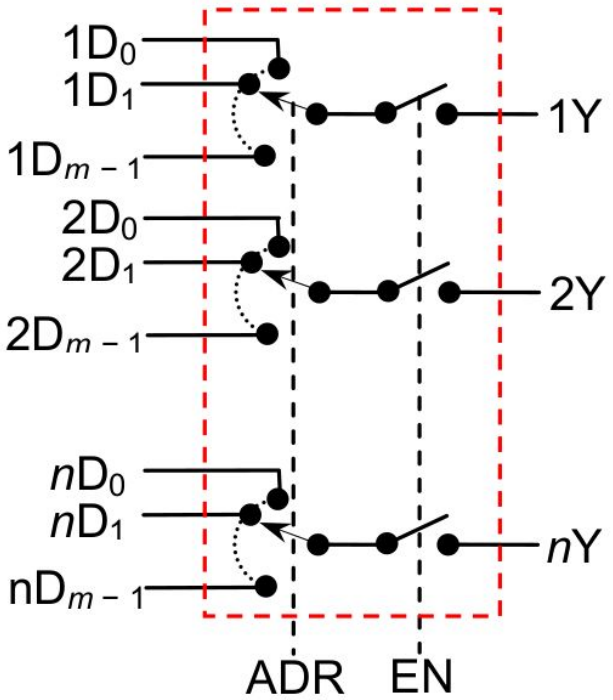
Счетчик до 10

```
module counter(
   input clk
);

reg [3:0]cnt = 0;

always @(posedge clk) begin
    if (cnt == 9)
        cnt <= 0;
    else
        cnt <= cnt + 1;
end

endmodule
```



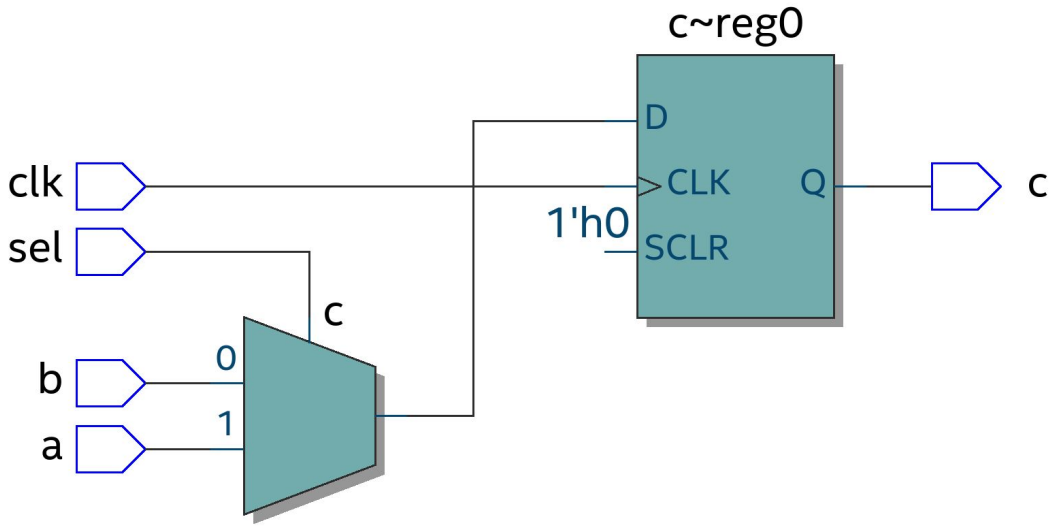
Мультиплексор

- Передает данные от одного из m источников на выход
- Имеет m источников n-разрядных данных, один k-разрядный адресный вход и один n-разрядный выход
- $k = \lceil \log_2(m) \rceil$



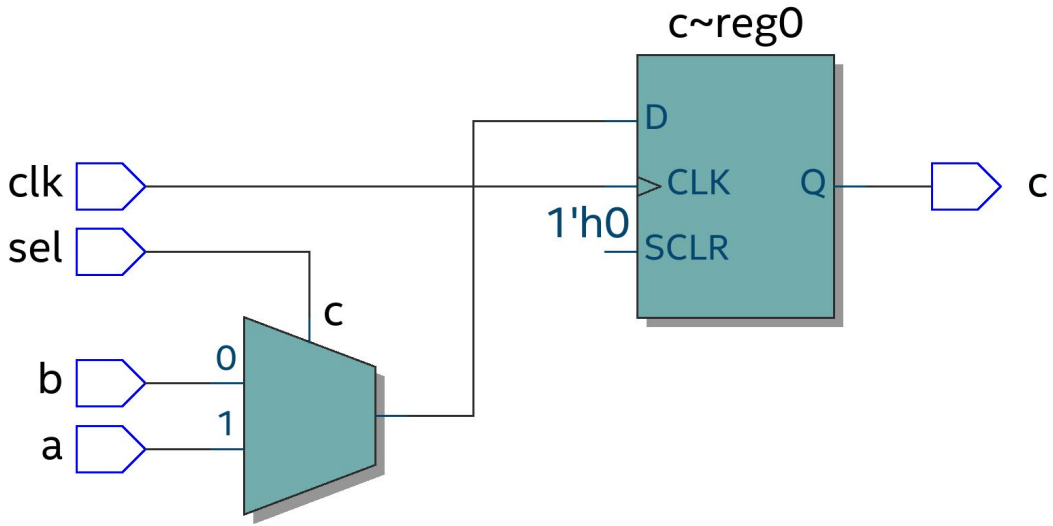
Выражение if-else

```
module mux(
     input clk, a, b, sel,

     output reg c
);

always @(posedge clk) begin
    if (sel)
        c <= a;
    else
        c <= b;
end

endmodule
```



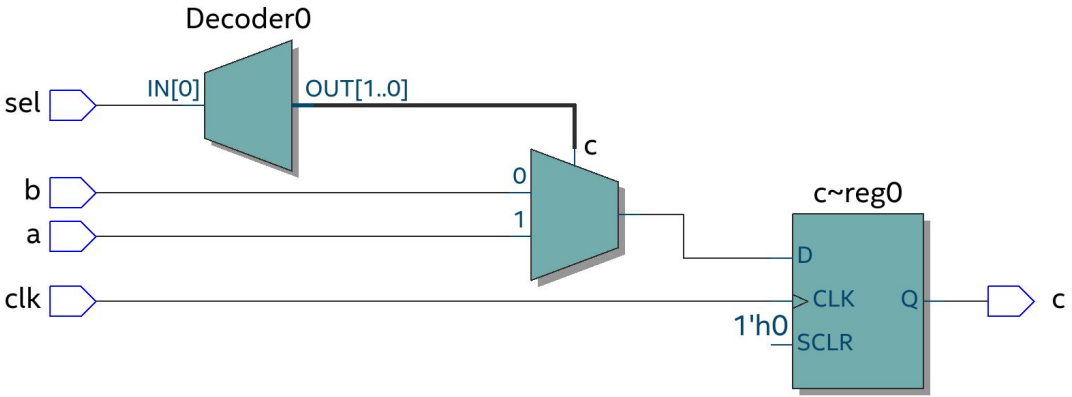
Тернарный оператор

```
module mux(
     input clk, a, b, sel,

     output reg c
);

always @(posedge clk) begin
    c <= sel ? a : b;
end

endmodule
```



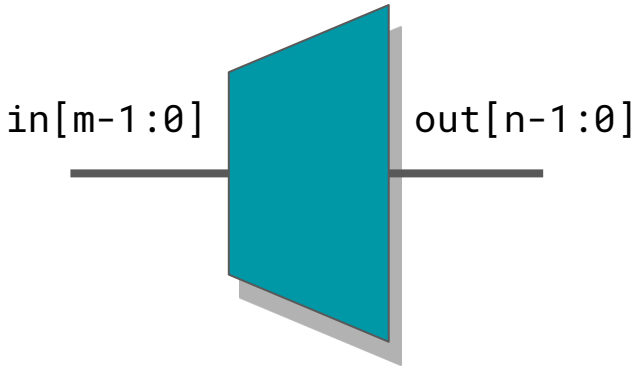
Выражение case

```
module mux_case(
     input clk, a, b, sel,

     output reg c
);

always @(posedge clk) begin
    case (sel)
    1'b1:
        c <= a;
    1'b0:
        c <= b;
    endcase
end

endmodule
```



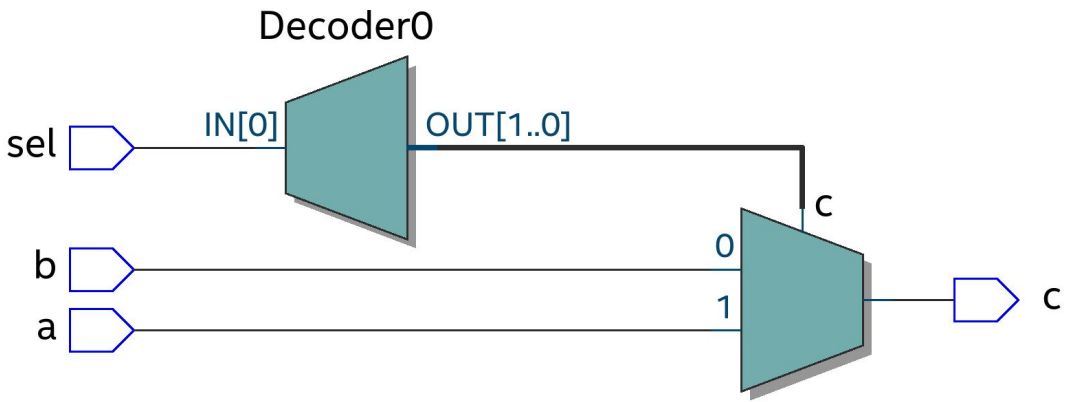
Декодер

- Превращает m-разрядное кодовое слово в соответствующее ему n-разрядное
- Имеет m-разрядный вход и n-разрядный выход
- $m \leq 2^n$



Case в комбинационной логике

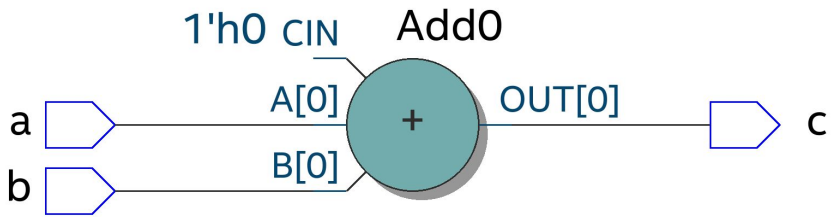
```
module mux_case(
     input a, b, sel,

     output reg c
);

always @(*) begin
    case (sel)
    1'b1: c = a;
    1'b0: c = b;
    endcase
end

endmodule
```



Присваивание

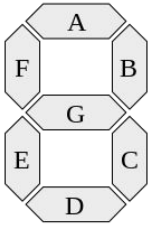
- Непрерывное — continuous assignment
- Неблокирующее — nonblocking assignment
- Блокирующее — blocking assignment
  - Используется для описания сложной комбинационной логики
  - Внутри блока `always @(*)`
  - Слева — reg, справа — любые выходы/операции

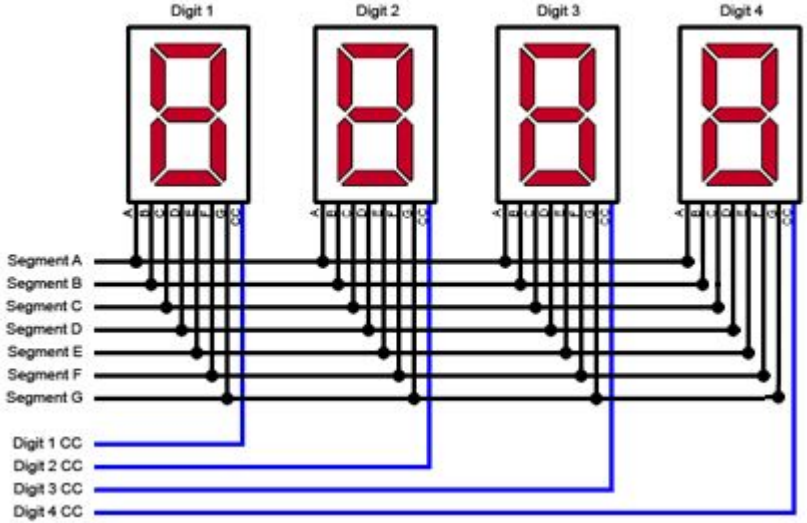
```
wire a, b;
reg c;

always @(*) begin
    c = a + b;
end
```



Семисегментный индикатор

- Динамическая индикация



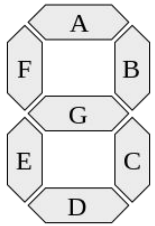


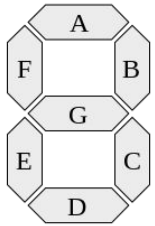
Делитель частоты

```
module clk_div(
   input clk,

   output clk2
);

reg [11:0]cnt = 0;

assign clk2 = cnt[11];

always @(posedge clk) begin
   cnt <= cnt + 12'b1;
end

endmodule
```

Семисегментный декодер

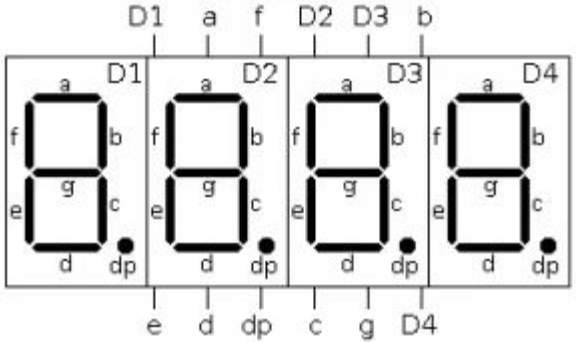
```
module bin_to_seg(
   input data,

   output reg [6:0]segments
);

always @(*) begin
   case (data)
       1'b0: segments = 7'b1111110;
       1'b1: segments = 7'b0110000;
   endcase
end

endmodule
```





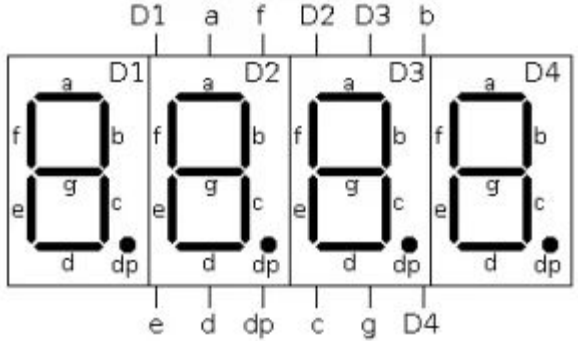
Динамическая индикация

```
module bin_display(
   input clk, [3:0]data,

   output [3:0]anodes, [6:0]segments
);

reg [1:0]i = 0;
assign anodes = (4'b1 << i);

always @(posedge clk) begin
   i <= i + 2'b1;
end

wire b = data[i];
bin_to_seg bin_to_seg(.data(b), .segments(segments));

endmodule
```

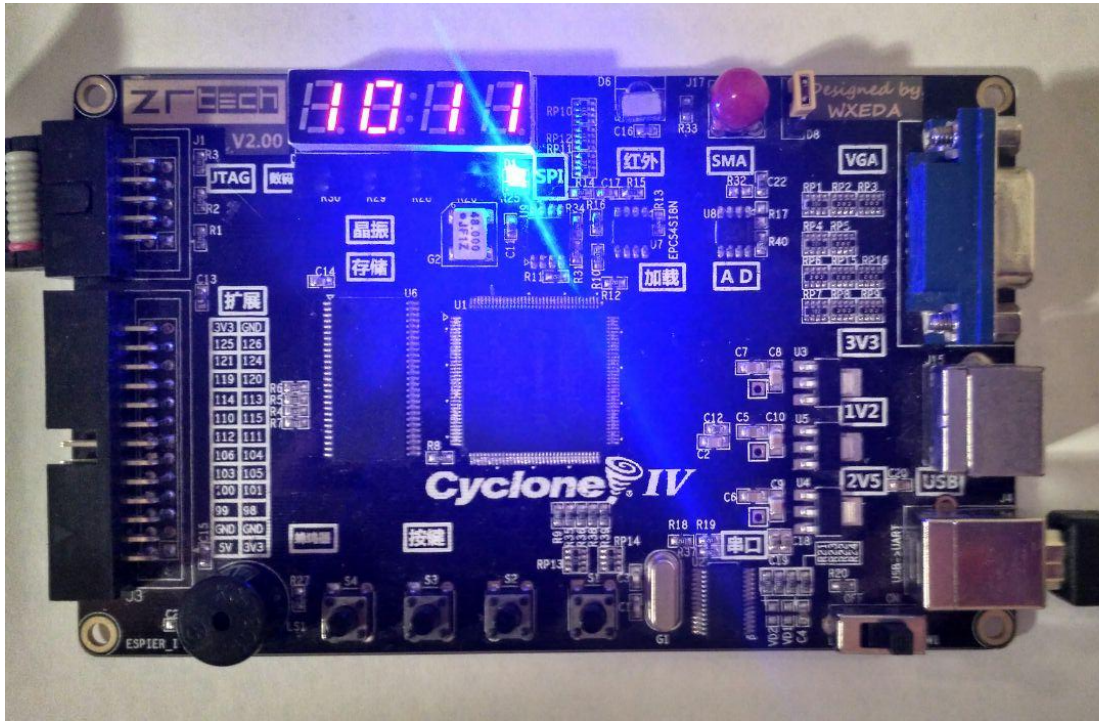


top.v

```
module top(
   input CLK,

   output DS_EN1, DS_EN2, DS_EN3, DS_EN4,
   output DS_A, DS_B, DS_C, DS_D, DS_E, DS_F, DS_G
);

wire [3:0]d = 4'b1011;
wire [3:0]anodes;
assign {DS_EN1, DS_EN2, DS_EN3, DS_EN4} = ~anodes;

wire [6:0]segments;
assign {DS_A, DS_B, DS_C, DS_D, DS_E, DS_F, DS_G} = segments;

clk_div clk_div(.clk(CLK), .clk2(clk2));

bin_display disp(.clk(clk2), .data(d), .anodes(anodes), .segments(segments));

endmodule
```



Результат


zrtech
V2.00
Designed by
WXEDA
JTAG
SPI
红外
SMA
VGA
晶振
存储
加载
A D
扩展
3V3
1V2
2V5
USB
按键
串口
Cyclone IV

GitHub

github.com/viktor-prutyanov/drec-fpga-intro