Verilog HDL (3)

Hsi-Pin Ma

http://lms.nthu.edu.tw/course/24953
Department of Electrical Engineering
National Tsing Hua University

Behavior Modeling

At every positive edge of CLOCK

If PRESET and CLEAR are not low

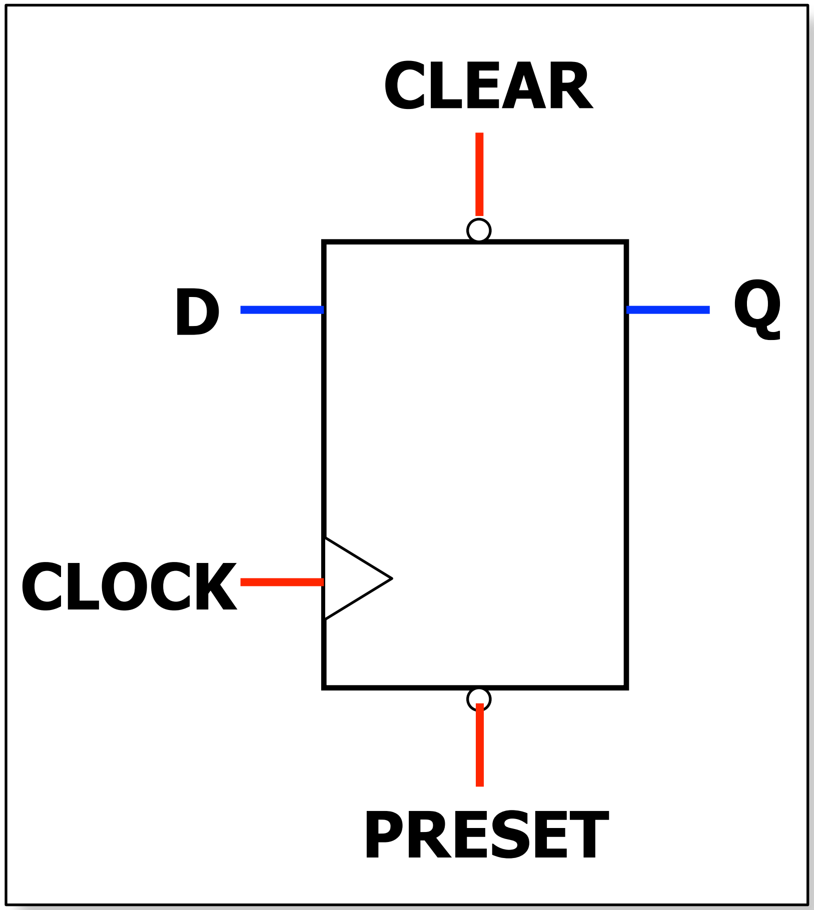
set Q to the value of D

Whenever PRESET goes low

set Q to logic 1

Whenever CLEAR goes low

set Q to logic 0

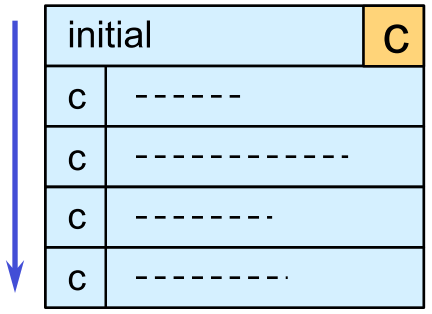


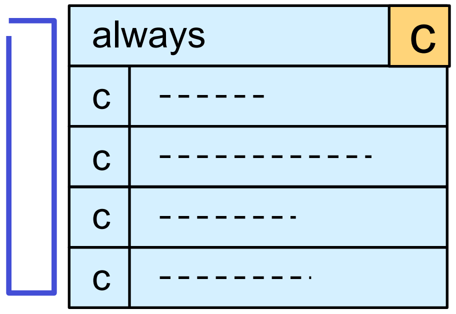
Behavior Modeling

- In behavior modeling, you must specify your circuit's
  - Action
    - How to model the circuit's behavior
  - Timing control
    - Timing
    - Condition
- Verilog supports the following structures for behavior modeling
  - Procedural block
  - Procedural assignment
  - Timing control
  - Control statement

Procedural Blocks

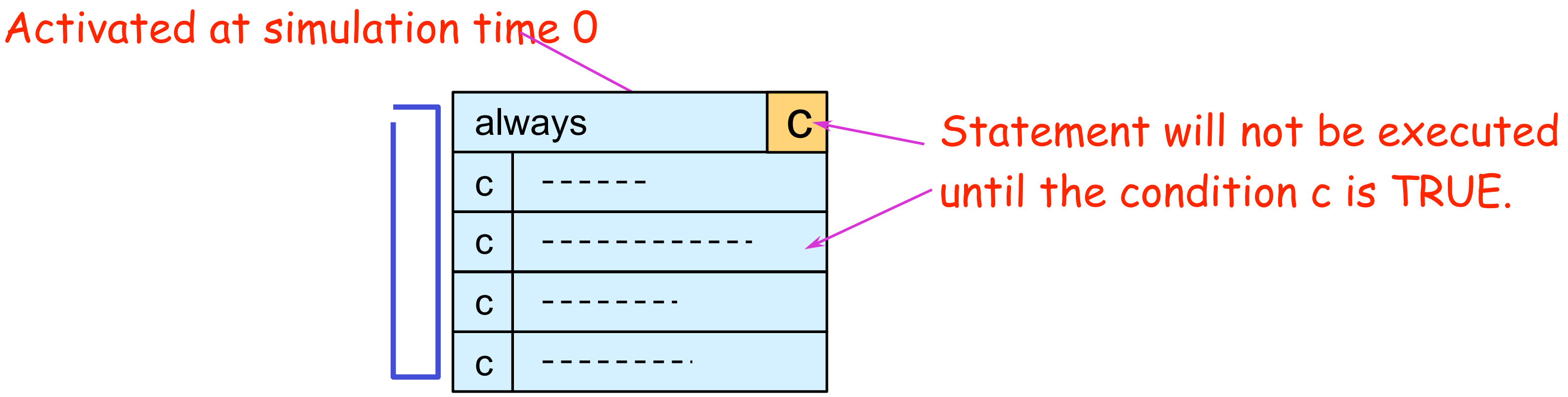
- In Verilog, procedural blocks are basis of behavior modeling
- Procedural blocks are of two types
  - initial procedural block, which execute only once
  - always procedural block, which execute in a loop





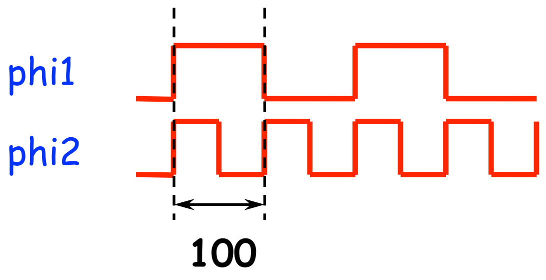
Procedural Blocks

- All procedural blocks are activated at simulation time 0.
  - The block will not be executed until the enabling condition evaluates TRUE.
  - Without the enabling condition, the block will be executed immediately.


Activated at simulation time 0
always
C
c ------
c -------------
c ---------
c ---------
Statement will not be executed until the condition c is TRUE.

Procedural Blocks

```
module clock_gen(phi1,phi2);
output       phi1,phi2;
reg  phi1,phi2;

initial
begin
      phi1=0;phi2=0
end

always
      #100 phi1=~phi1;

always       @(posedge phi1)
begin
      phi2=1;
      #50       phi2=0;
      #50       phi2=1;
      #50       phi2=0;
end
endmodule
```



These procedural blocks are activated and executed at simulation time 0

This procedural block is activated at simulation time 0 but executed at positive edge of phi1

Procedural Blocks

- Three components
  - Procedural assignment statements
  - High-level programming language constructs
  - Timing controls
- Using the first two components to model the actions of the circuit.
- Using timing controls to model when should these actions happen.

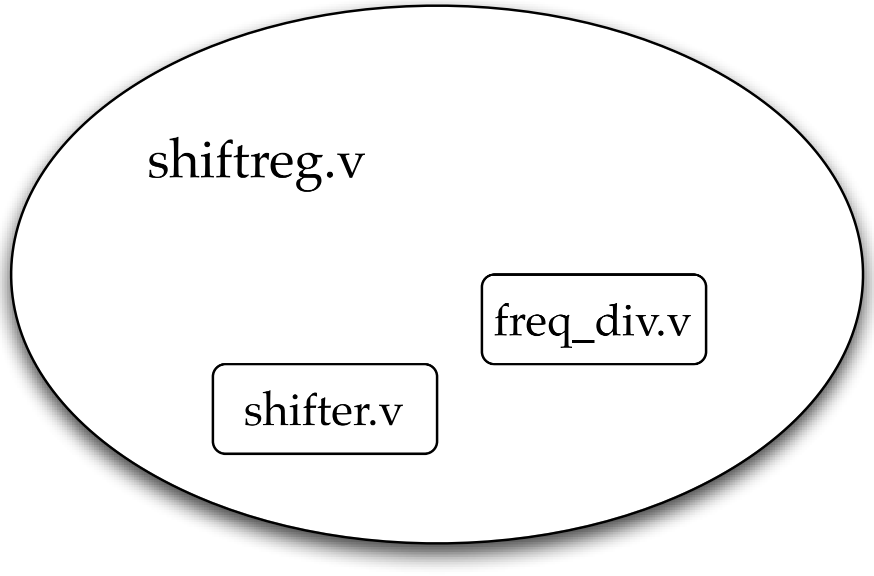
Procedural Timing Control

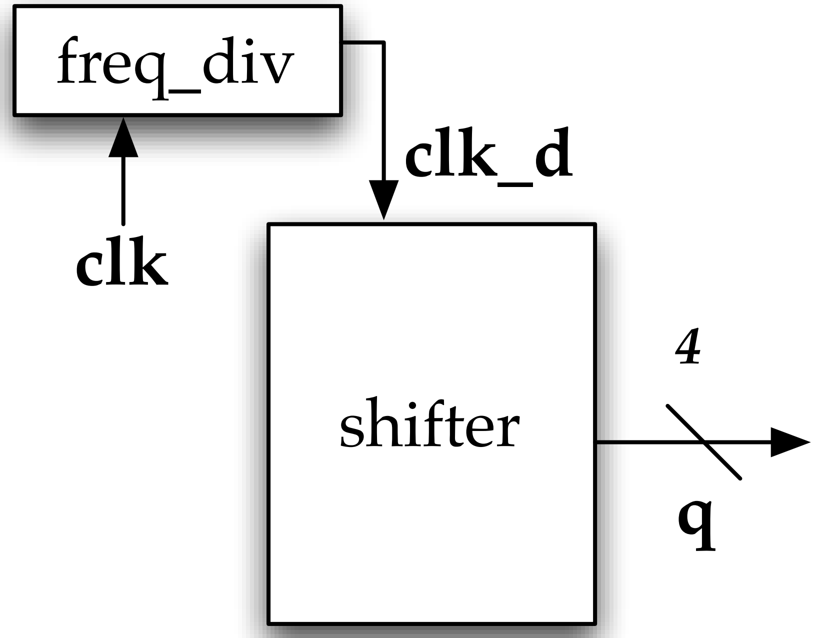
- Three types
  - Simple delay control
    - #50 clk=~clk;
  - Event control
    - @* sum=a+b+ci;
    - @(posedge clk) q<=d;
  - Level-sensitive timing control

Modularized Shift Register Design

Shift Register

Shift Register

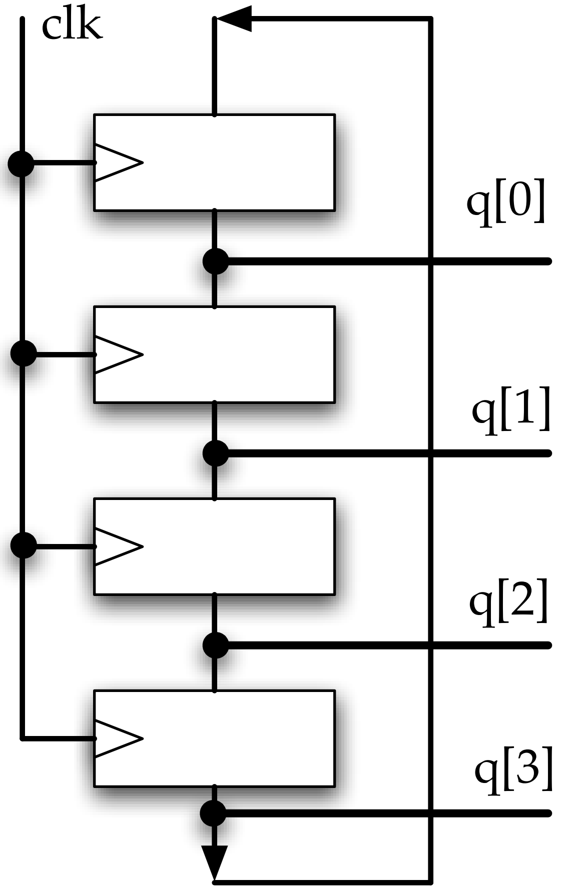




shifter.v

```
`define BIT_WIDTH 4
module shifter(
  q,  // shifter output
  clk,  // global clock
  rst_n  // active low reset
);

output [`BIT_WIDTH-1:0] q;  // output
input clk;  // global clock
input rst_n;  // active low reset

reg [`BIT_WIDTH-1:0] q;  // output

// Sequential logics: Flip flops
always @(posedge clk or negedge rst_n)
  if (~rst_n)
  begin
    q<=`BIT_WIDTH'b0101;
  end
  else
 begin
    q[0]<=q[3];
    q[1]<=q[0];
    q[2]<=q[1];
    q[3]<=q[2];
  end
endmodule
```

initial value 0101



shiftreg.v

1

```
`define BIT_WIDTH 4
module shift_reg(
  q,  // LED output
  clk, // global clock
  rst_n  // active low reset
);

output [`BIT_WIDTH-1:0] q;  // LED output
input clk;  // global clock
input rst_n;  // active low reset

wire clk_d; // divided clock
wire [`BIT_WIDTH-1:0] q;  // LED output
```

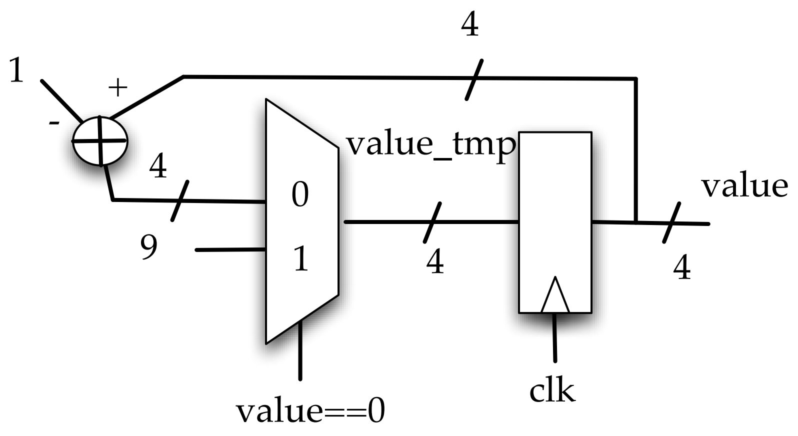
```
// Insert frequency divider (freq_div.v)
freq_div U_FD(
  .clk_out(clk_d),  // divided clock output
  .clk(clk),  // clock from the crystal
  .rst_n(rst_n)  // active low reset
);

// Insert shifter (shifter.v)
shifter U_D(
  .q(q),  // shifter output
  .clk(clk_d),  // clock from the frequency divider
  .rst_n(rst_n)  // active low reset
);

endmodule
```

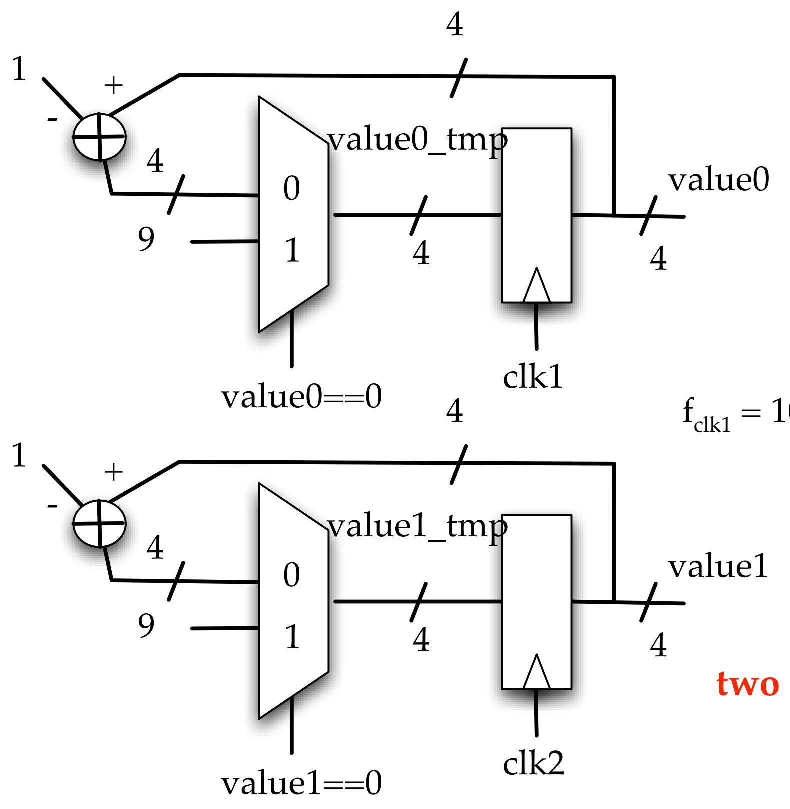
2

Modulized BCD Down Counter

BCD Down-Counter

single digit



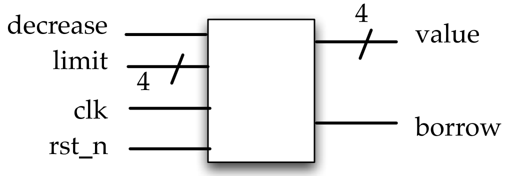


$f_{clk1} = 10 \times f_{clk2}$

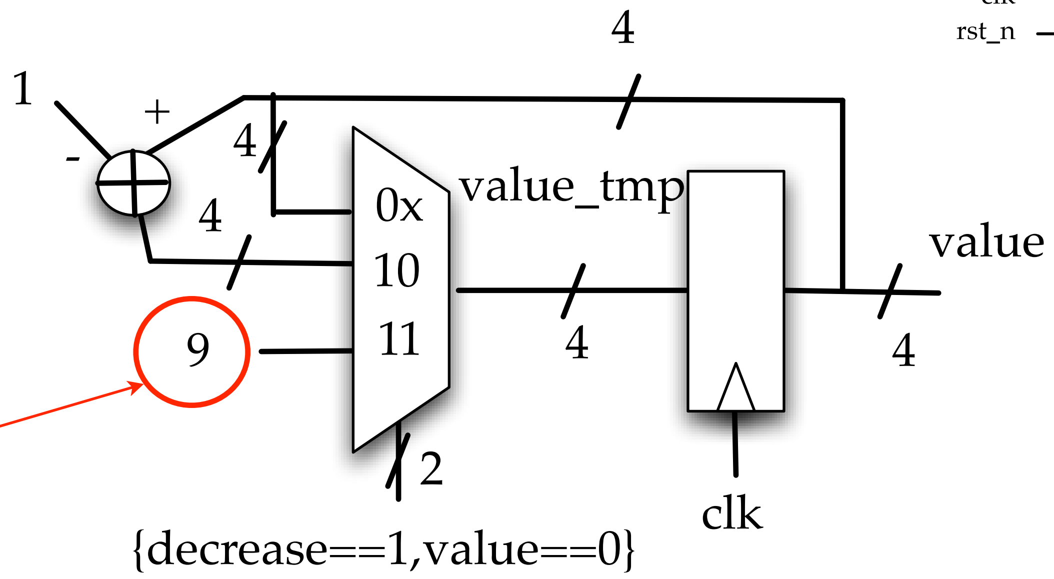
two digits

```
// Combinational logics
always @*
  if (value==`BCD_ZERO)
    value_tmp = `BCD_NINE;
  else
    value_tmp = value - `INCREMENT;

// register part for BCD counter
always @(posedge clk or negedge rst_n)
  if (~rst_n) value <= `BCD_ZERO;
  else value <= value_tmp;
```

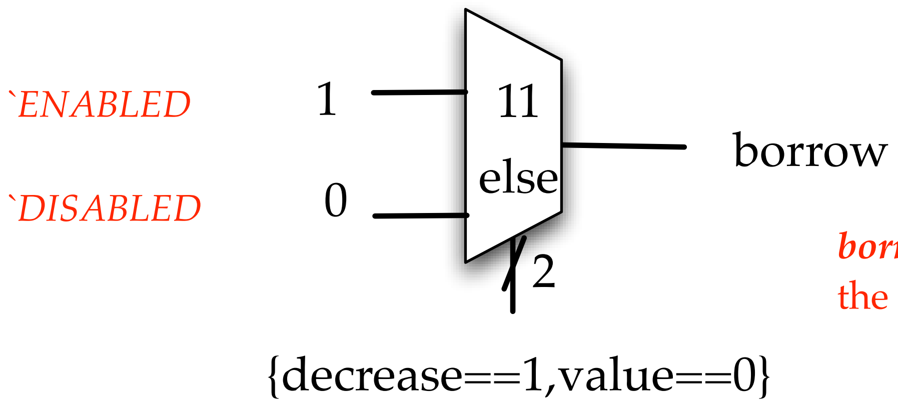
BCD Down-Counter



How to use one single clock for different digit counting?



Use ***limit*** as different ceiling for each digit



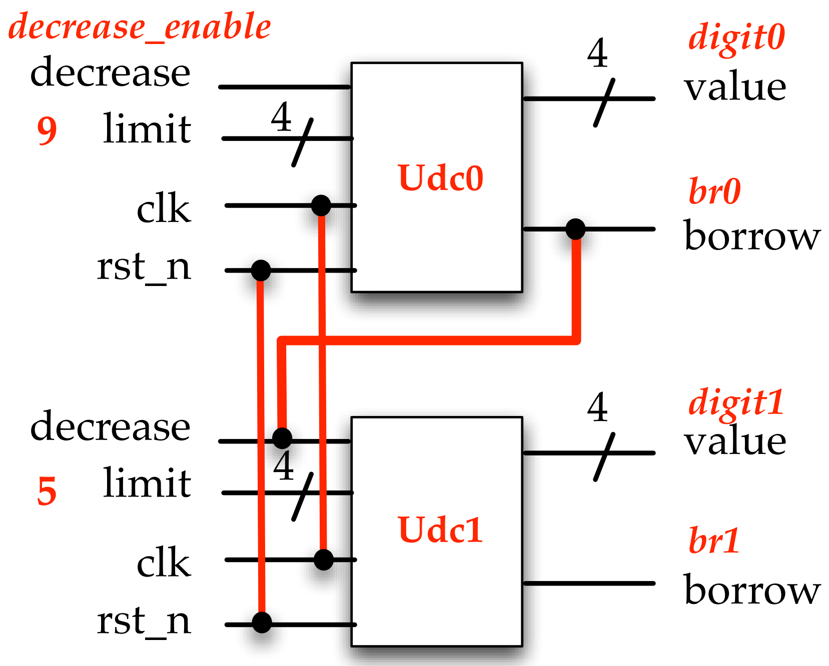
borrow signal in lower digit is the ***decrease*** control in the upper digit

BCD Down-Counter

```
module downcounter(
 value,  // counter output
 borrow,  // borrow indicator
 clk, // global clock
 rst_n, // active low reset
 decrease, // counter enable control
 limit // limit for the counter
);
... ...
// Combinational logics
always @*
 if (value==`BCD_ZERO && decrease)
  begin
   value_tmp = limit;
   borrow = `ENABLED;
  end
 else if (value!=`BCD_ZERO && decrease)
  begin
   value_tmp = value - `INCREMENT;
   borrow = `DISABLED;
  end
 else
  begin
   value_tmp = value;
   borrow = `DISABLED;
  end
```

```
// register part for BCD counter
always @(posedge clk or negedge rst_n)
 if (~rst_n) value <= `BCD_ZERO;
 else value <= value_tmp;

endmodule
```

BCD Down-Counter



```
// 30 sec down counter
downcounter Udc0(
  .value(digit0),  // counter value
  .borrow(br0),  // borrow indicator
  .clk(clk), // global clock signal
  .rst_n(rst_n),  // low active reset
  .decrease(decrease_enable),  // counter enable control
  .limit(`BCD_NINE)  // limit for the counter
);

downcounter Udc1(
  .value(digit1),  // counter value
  .borrow(br1),  // borrow indicator
  .clk(clk), // global clock signal
  .rst_n(rst_n),  // low active reset
  .decrease(br0),  // counter enable control
   .limit(`BCD_FIVE)  // limit for the counter
);
```