



Counters

黃元豪

Yuan-Hao Huang

國立清華大學電機工程學系

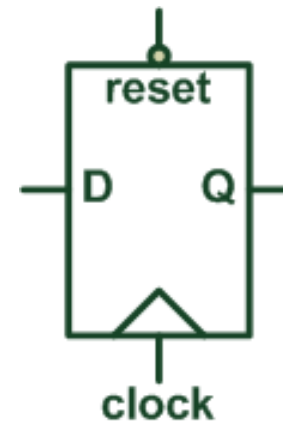
Department of Electrical Engineering

National Tsing-Hua University

D-type Flip Flop



```
module dff(  
  q, // output  
  d, // input  
  clk, // global clock  
  rst // active high reset  
);  
  
output q; // output  
input d; // input  
input clk; // global clock  
input rst; // active high reset  
  
reg q; // output (in always block)  
  
always @(posedge clk or posedge rst)  
  if (rst)  
    q<=0;  
  else  
    q<=d;  
  
endmodule
```



Binary Up Counter



```
`define BCD_BIT_WIDTH 4
`define BCD_ZERO 4'd0
`define BCD_ONE 4'd1
`define BCD_NINE 4'd9
module bcdcounter(
  q, // output
  clk, // global clock
  rst // active high reset
);

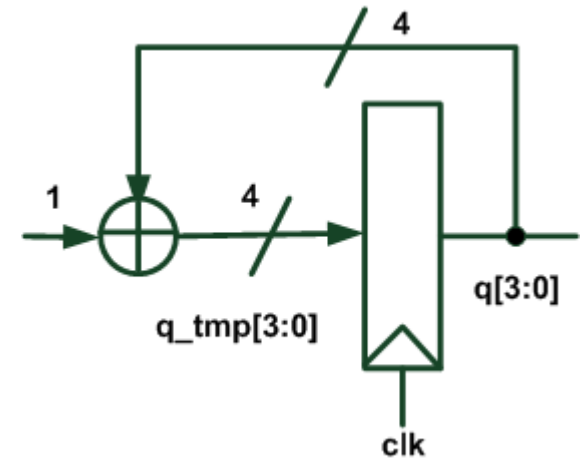
output [BCD_BIT_WIDTH-1:0] q; // output
input clk; // global clock
input rst; // active high reset

reg [BCD_BIT_WIDTH-1:0] q; // output (in always block)
reg [BCD_BIT_WIDTH-1:0] q_tmp; // input to dff (in always block)

// Combinational logics
always @(q)
  q_tmp = q + `BCD_ONE;

// Sequential logics: Flip flops
always @(posedge clk or posedge rst)
  if (rst) q<=`BCD_BIT_WIDTH'd0;
  else q<=q_tmp;

endmodule
```



BCD Counter



```
`define BCD_BIT_WIDTH 4
`define BCD_ZERO 4'd0
`define BCD_ONE 4'd1
`define BCD_NINE 4'd9
module bcdcounter(
  q, // output
  clk, // global clock
  rst // active high reset
);

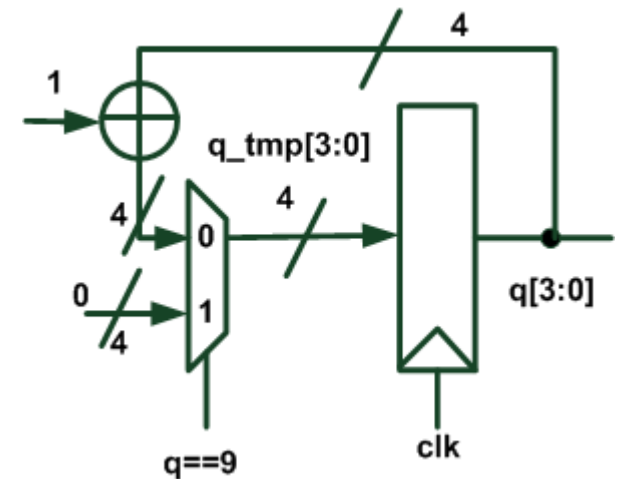
output [BCD_BIT_WIDTH-1:0] q; // output
input clk; // global clock
input rst; // active high reset

reg [BCD_BIT_WIDTH-1:0] q; // output (in always block)
reg [BCD_BIT_WIDTH-1:0] q_tmp; // input to dff (in always block)

// Combinational logics
always @(q)
  if (q== `BCD_NINE) q_tmp = `BCD_ZERO;
  else q_tmp = q + `BCD_ONE;

// Sequential logics: Flip flops
always @(posedge clk or posedge rst)
  if (rst) q<=`BCD_BIT_WIDTH'd0;
  else q<=q_tmp;

endmodule
```



Frequency Divider



```
`define FREQ_DIV_BIT 24
module freq_div(
  clk_out, // divided clock output
  clk, // global clock input
  rst // active high reset
);

output clk_out; // divided output
input clk; // global clock input
input rst; // active high reset

reg [FREQ_DIV_BIT-1:0] cnt; // remainder of the counter
reg [FREQ_DIV_BIT-1:0] cnt_tmp; // input to dff (in always block)

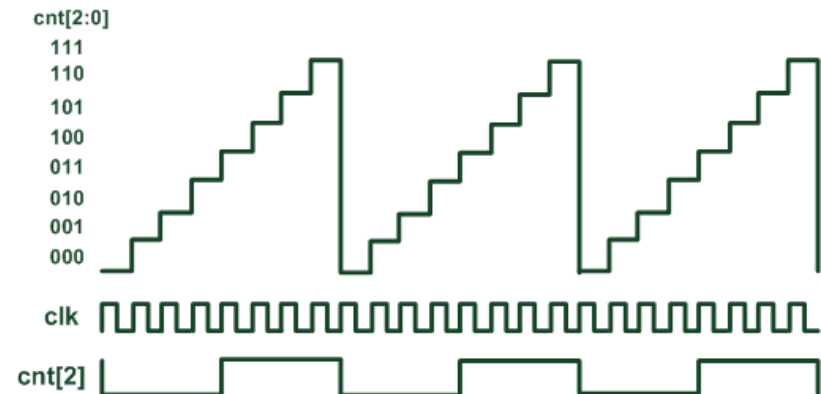
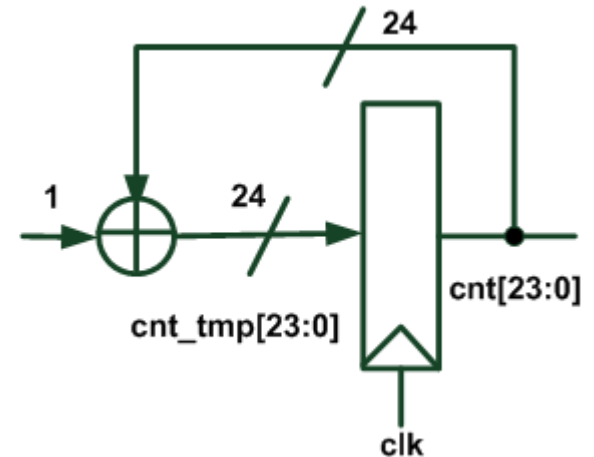
wire clk_out;

// Divided frequency
assign clk_out = cnt[23];

// Combinational logics: increment, neglecting overflow
always @(cnt)
  cnt_tmp = cnt + 1'b1;

// Sequential logics: Flip flops
always @(posedge clk or posedge rst)
  if (rst)
    cnt<= `FREQ_DIV_BIT'd0;
  else
    cnt<=cnt_tmp;

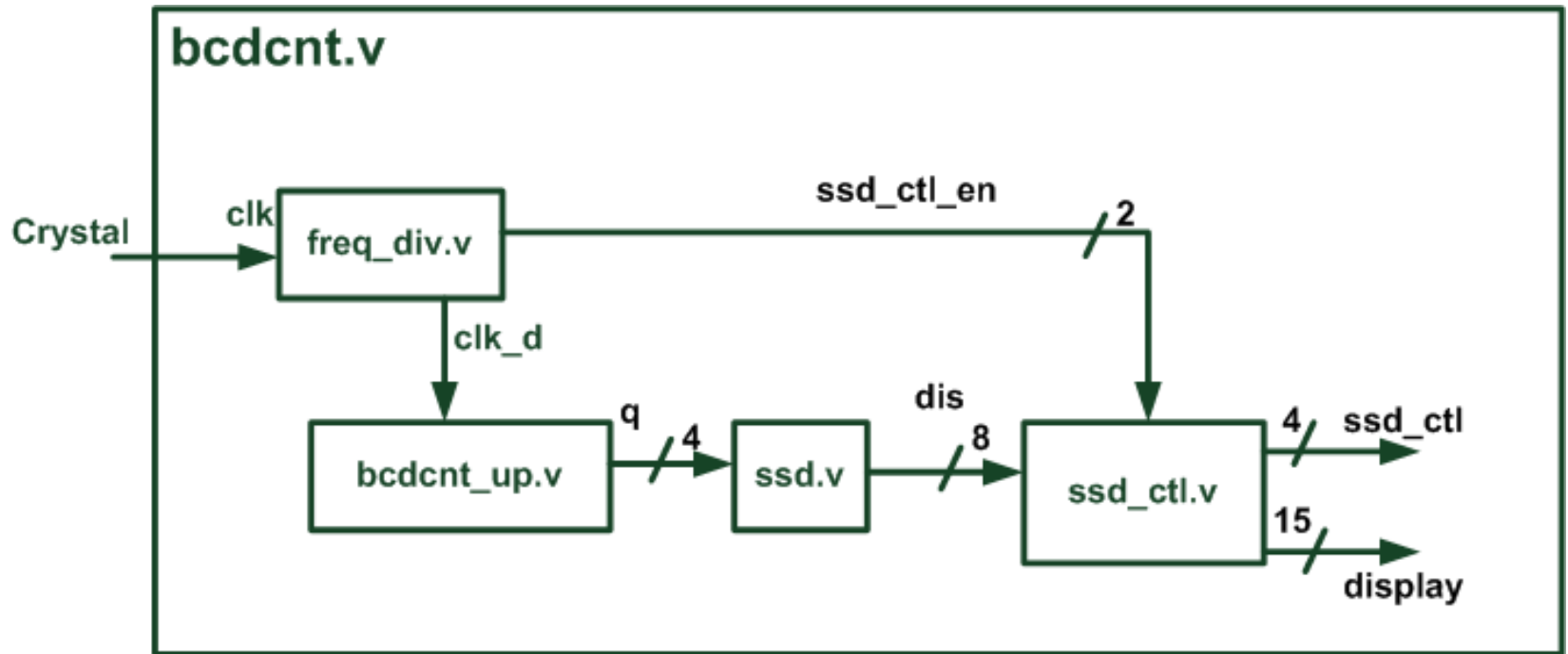
endmodule
```





Modularized BCD Up Counter

BCD Up Counter



```

`define FREQ_DIV_BIT 27
module freq_div(
    ssd_ctl_en, // seven-segment display control
    clk_out, // divided clock output
    clk, // global clock input
    rst // active high reset
);

output [1:0] ssd_ctl_en; // seven-segment display control
output clk_out; // divided output
input clk; // global clock input
input rst; // active high reset

reg [FREQ_DIV_BIT-1:0] cnt; // remainder of the counter
reg [FREQ_DIV_BIT-1:0] cnt_next; // input to dff (in always block)

wire clk_out;
wire [1:0] ssd_ctl_en;

// seven segment display control
assign ssd_ctl_en = cnt[20:19];

// Divided frequency
assign clk_out = cnt[FREQ_DIV_BIT-1];

// Combinational logics: increment, neglecting overflow
always @(cnt)
    cnt_next = cnt + 1'b1;

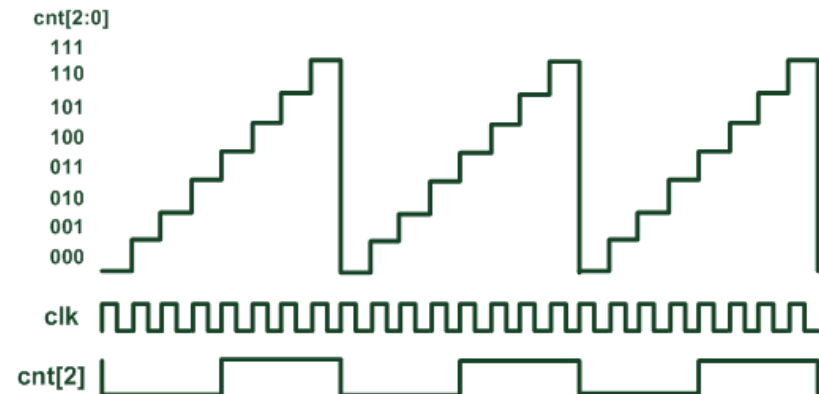
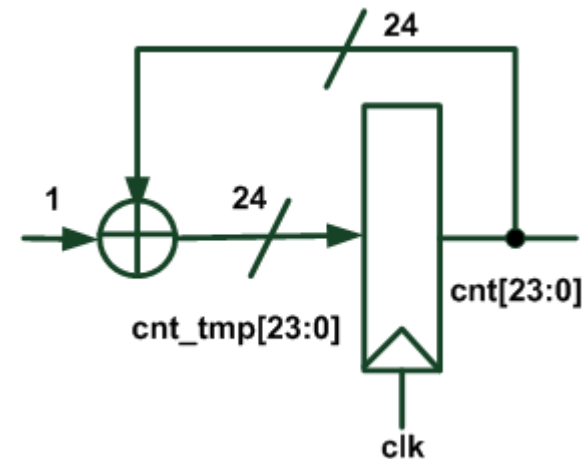
// Sequential logics: Flip flops
always @(posedge clk or posedge rst)
    if (rst)
        cnt<=`FREQ_DIV_BIT'd0;
    else
        cnt<=cnt_next;

endmodule

```

Frequency Divider

- 2-bit `cnt[17:16]` are used to divide the display time to four time-slots with equal durations.



bcdcnt_up.v



```
`define BCD_BIT_WIDTH 4 // BCD bit width
`define BCD_ZERO 4'd0
`define BCD_ONE 4'd1
`define BCD_NINE 4'd9

module bcdcnt_up(
    q, // output
    clk, // global clock
    rst // active high reset
);

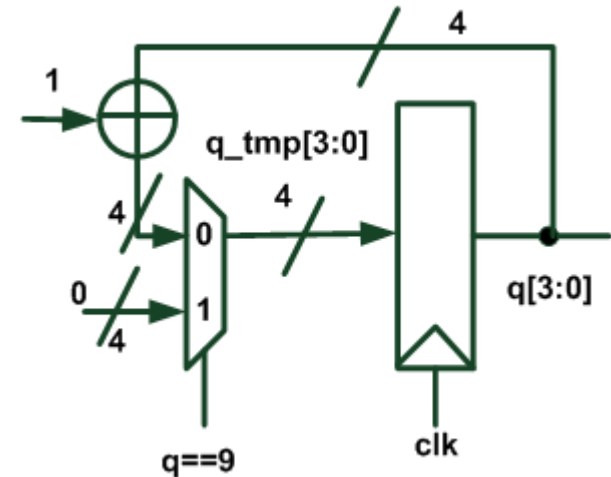
output [BCD_BIT_WIDTH-1:0] q; // output
input clk; // global clock
input rst; // active high reset

reg [BCD_BIT_WIDTH-1:0] q; // output (in always block)
reg [BCD_BIT_WIDTH-1:0] q_tmp; // input to dff (in always block)

// Combinational logics
always @(q)
    if (q==`BCD_NINE) q_tmp = `BCD_ZERO;
    else q_tmp = q + `BCD_ONE;

// Sequential logics: Flip flops
always @(posedge clk or posedge rst)
    if (rst) q<=`BCD_BIT_WIDTH'd0;
    else q<=q_tmp;

endmodule
```



Seven-Segment Display Decoder



ssd.v

```
`define BCD_BIT_WIDTH 4 // BCD bit width
`define SSD_BIT_WIDTH 8 // SSD display control bit width
`define SSD_ZERO `SSD_BIT_WIDTH'b0000001_1 // 0
`define SSD_ONE `SSD_BIT_WIDTH'b1001111_1 // 1
`define SSD_TWO `SSD_BIT_WIDTH'b0010010_1 // 2
`define SSD_THREE `SSD_BIT_WIDTH'b000110_1 // 3
`define SSD_FOUR `SSD_BIT_WIDTH'b1001100_1 // 4
`define SSD_FIVE `SSD_BIT_WIDTH'b0100100_1 // 5
`define SSD_SIX `SSD_BIT_WIDTH'b0100000_1 // 6
`define SSD_SEVEN `SSD_BIT_WIDTH'b0001111_1 // 7
`define SSD_EIGHT `SSD_BIT_WIDTH'b0000000_1 // 8
`define SSD_NINE `SSD_BIT_WIDTH'b0001100_1 // 9
`define SSD_DEF `SSD_BIT_WIDTH'b0000000_0 // default
```

```
module ssd(
    display, // SSD display output
    bcd // BCD input
);

output [0:`SSD_BIT_WIDTH-1] display; // SSD display output
input [ BCD_BIT_WIDTH-1:0] bcd; // BCD input

reg [0:`SSD_BIT_WIDTH-1] display; // SSD display output (in
always)

// Combinational logics:
always @(bcd)
    case (bcd)
        `BCD_BIT_WIDTH'd0: display = `SSD_ZERO;
        `BCD_BIT_WIDTH'd1: display = `SSD_ONE;
        `BCD_BIT_WIDTH'd2: display = `SSD_TWO;
        `BCD_BIT_WIDTH'd3: display = `SSD_THREE;
        `BCD_BIT_WIDTH'd4: display = `SSD_FOUR;
        `BCD_BIT_WIDTH'd5: display = `SSD_FIVE;
        `BCD_BIT_WIDTH'd6: display = `SSD_SIX;
        `BCD_BIT_WIDTH'd7: display = `SSD_SEVEN;
        `BCD_BIT_WIDTH'd8: display = `SSD_EIGHT;
        `BCD_BIT_WIDTH'd9: display = `SSD_NINE;
        default: display = `SSD_DEF;
    endcase

endmodule
```

Seven-Segment Display Control

```
module ssd_ctl(display_c, display, ssd_ctl_en, display0, display1, display2, display3);
```

```
input [1:0] ssd_ctl_en;  
input [0:7] display0, display1, display2, display3;
```

```
output [0:7] display;  
reg [0:7] display;  
output [3:0] display_c;  
reg [3:0] display_c;
```

```
always @(ssd_ctl_en or display0 or display1 or display2 or display3)
```

```
// display value selection
```

```
case(ssd_ctl_en)  
  2'b00: display = display0;  
  2'b01: display = display1;  
  2'b10: display = display2;  
  2'b11: display = display3;  
  default : display = 8'd0;  
endcase
```

```
// 7-segment control
```

```
always @(ssd_ctl_en)  
  case(ssd_ctl_en)  
    2'b00: display_c = 4'b1110;  
    2'b01: display_c = 4'b1101;  
    2'b10: display_c = 4'b1011;  
    2'b11: display_c = 4'b0111;  
    default : display_c = 4'b1111;  
  endcase
```

```
endmodule
```



bcdcnt.v



```
`define BCD_BIT_WIDTH 4 // BCD bit width
`define SSD_BIT_WIDTH 8 // SSD display control bit width
`define SSD_DIGIT_NUM 4 // number of SSD digit

module bcdcnt( q, // LED output
display, // seven-segment display output
ssd_ctl, // scan control for 7-segment display
clk, // global clock
rst // active high reset
);

output [`BCD_BIT_WIDTH-1:0] q; // LED output
// 7-segment display control
output [0:`SSD_BIT_WIDTH-1] display;
output [`SSD_DIGIT_NUM-1:0] ssd_ctl; // scan control for ssd
input clk; // global clock
input rst; // active high reset
wire clk_d; // divided clock
wire [1:0] ssd_ctl_en; // ssd control signal

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

```
// BCD up counter
bcdcnt_up U_CNT(
.q(q), // output
.clk(clk_d), // global clock
.rst(rst) // active high reset
);

wire [0:`SSD_BIT_WIDTH-1] display_temp;
// Seven-segment display decoder
ssd U_SSD(
.display(display_temp), // SSD display output
.bcd(q) // BCD input
);

// Let four seven-segment displays show the same digit

ssd_ctl CTRLR(
.display_c(ssd_ctl),
.display(display),
.ssd_ctl_en(ssd_ctl_en),
.display0(display_temp),
.display1(display_temp),
.display2(display_temp),
.display3(display_temp)
);

endmodule
```

Bcdcnt_constrain.xdc (1/2)



```
# I/O PIN ASSIGNMENT
set_property CFGBVS VCCO [current_design]
#where value1 is either VCCO or GND
set_property CONFIG_VOLTAGE 2.5 [current_design]
#where value2 is the voltage provided to configuration bank 0

set_property PACKAGE_PIN T17 [get_ports {rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst}]
set_property PACKAGE_PIN W5 [get_ports {clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
set_property PACKAGE_PIN V19 [get_ports {q[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[3]}]
set_property PACKAGE_PIN U19 [get_ports {q[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[2]}]
set_property PACKAGE_PIN E19 [get_ports {q[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[1]}]
set_property PACKAGE_PIN U16 [get_ports {q[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[0]}]
```



Bcdcnt_constrain.xdc (2/2)

```
set_property PACKAGE_PIN W4 [get_ports {ssd_ctl[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ssd_ctl[3]}]
set_property PACKAGE_PIN V4 [get_ports {ssd_ctl[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ssd_ctl[2]}]
set_property PACKAGE_PIN U4 [get_ports {ssd_ctl[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ssd_ctl[1]}]
set_property PACKAGE_PIN U2 [get_ports {ssd_ctl[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ssd_ctl[0]}]
```

```
set_property PACKAGE_PIN W7 [get_ports {display[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[0]}]
set_property PACKAGE_PIN W6 [get_ports {display[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[1]}]
set_property PACKAGE_PIN U8 [get_ports {display[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[2]}]
set_property PACKAGE_PIN V8 [get_ports {display[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[3]}]
set_property PACKAGE_PIN U5 [get_ports {display[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[4]}]
set_property PACKAGE_PIN V5 [get_ports {display[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[5]}]
set_property PACKAGE_PIN U7 [get_ports {display[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[6]}]
set_property PACKAGE_PIN V7 [get_ports {display[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[7]}]
```