Sequential Logic Examples

Hsi-Pin Ma

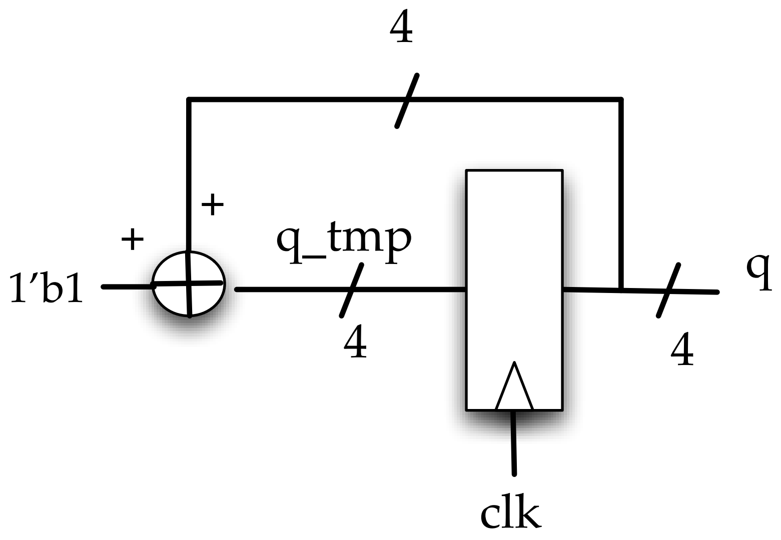
http://lms.nthu.edu.tw/course/40757

Department of Electrical Engineering

National Tsing Hua University

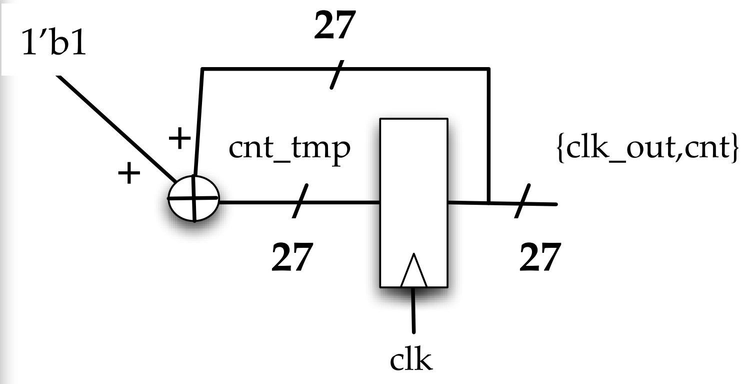
Binary Up Counter

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

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

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

// Combinational logics
always @*
  q_tmp = q + 1'b1;

// Sequential logics: Flip flops
always @(posedge clk or negedge rst_n)
  if (~rst_n) q<=`CNT_BIT_WIDTH'd0;
  else q<=q_tmp;

endmodule
```



Frequency Divider

```
`define FREQ_DIV_BIT 27
module freqdiv(
  clk_out,  // divided clock output
  clk,  // global clock input
  rst_n  // active low reset
);

output clk_out;  // divided output
input clk;  // global clock input
input rst_n;  // active low reset

reg clk_out; // clk output (in always block)
reg [`FREQ_DIV_BIT-2:0] cnt; // remainder of the counter
reg [`FREQ_DIV_BIT-1:0] cnt_tmp; // input to dff (in always block)

// Combinational logics: increment, neglecting overflow
always @*
  cnt_tmp = {clk_out,cnt} + 1'b1;

// Sequential logics: Flip flops
always @(posedge clk or negedge rst_n)
  if (~rst_n) {clk_out, cnt}<=`FREQ_DIV_BIT'd0;
  else {clk_out,cnt}<=cnt_tmp;

endmodule
```

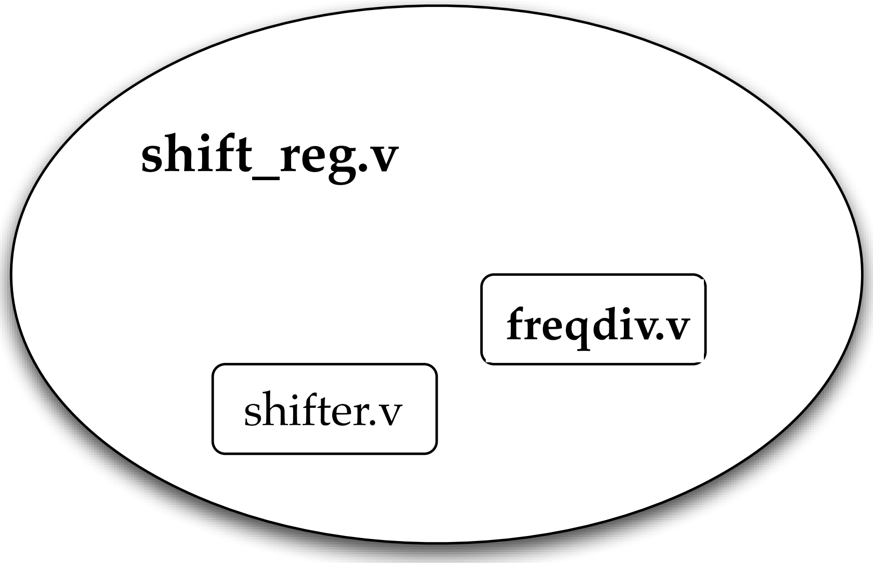


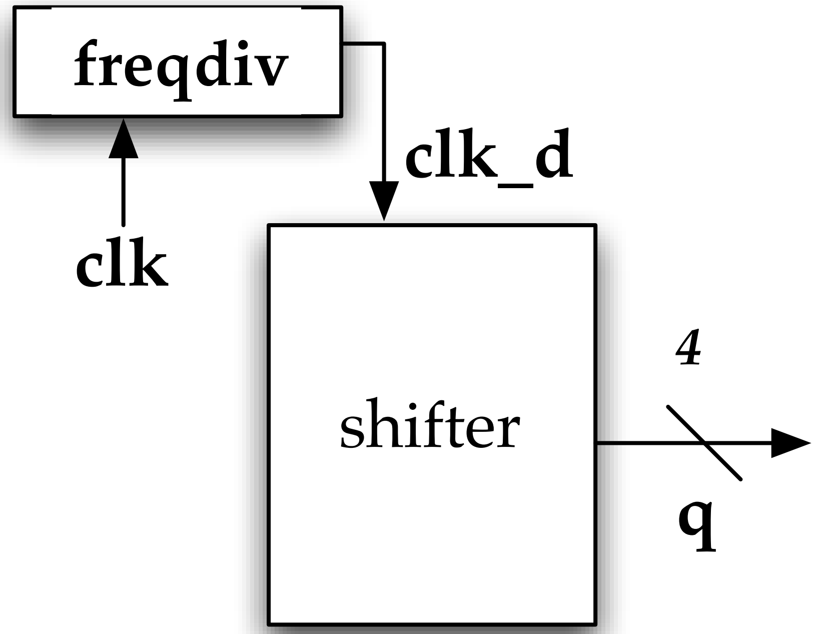
cnt_tmp[26:0]

cnt[26:0]

Modularized Shift Register

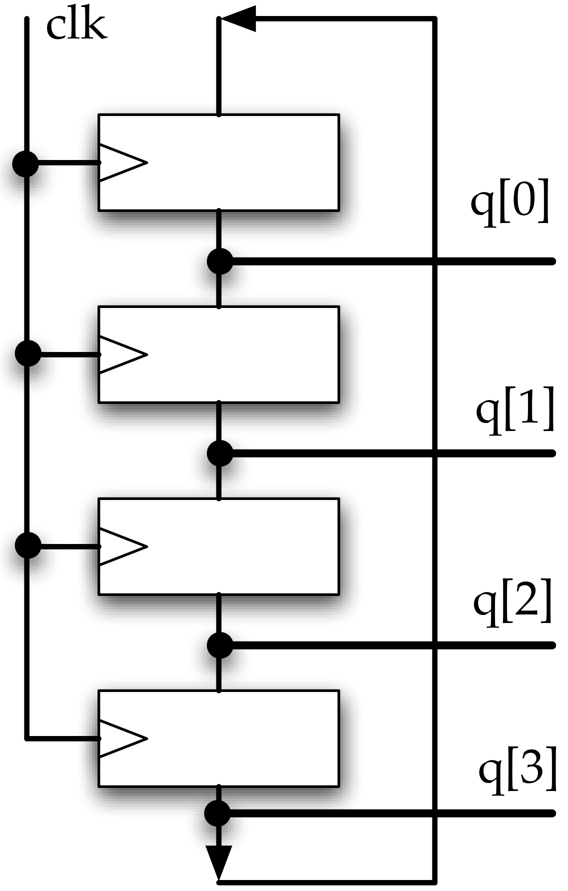
Shift Register





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'b0001;
  end                   initial value 0001
  else
 begin
    q[0]<=q[3];
    q[1]<=q[0];
    q[2]<=q[1];
    q[3]<=q[2];
  end
endmodule
```



Top Module (shift_reg.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)
freqdiv 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
```

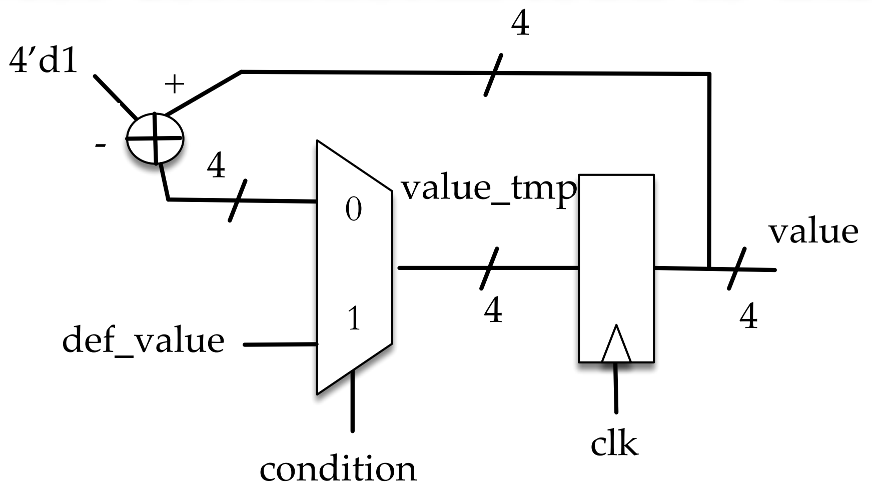
Modularized BCD Counter

Load Default Value for DFFs

- at reset of DFFs

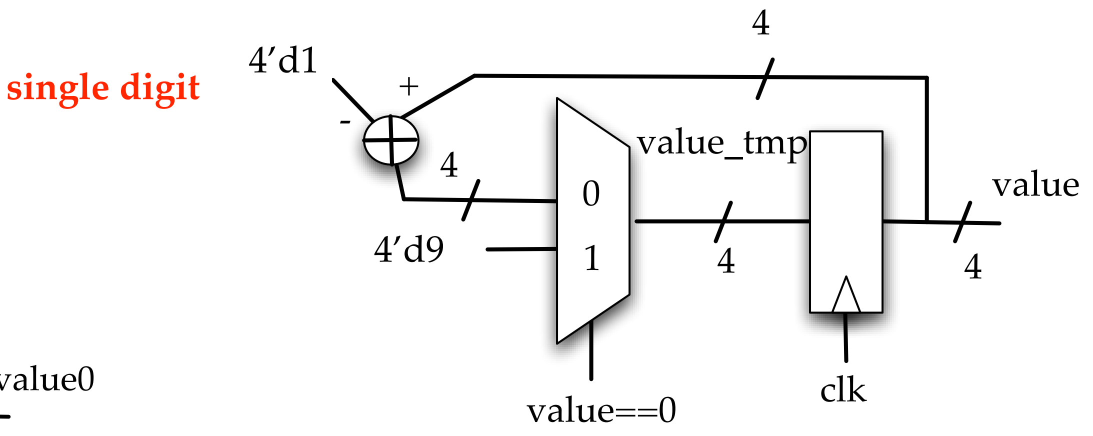
```
always @(posedge clk or negedge rst_n)
  if (~rst_n)
    q <= 0;
  else
    q <= d;
```

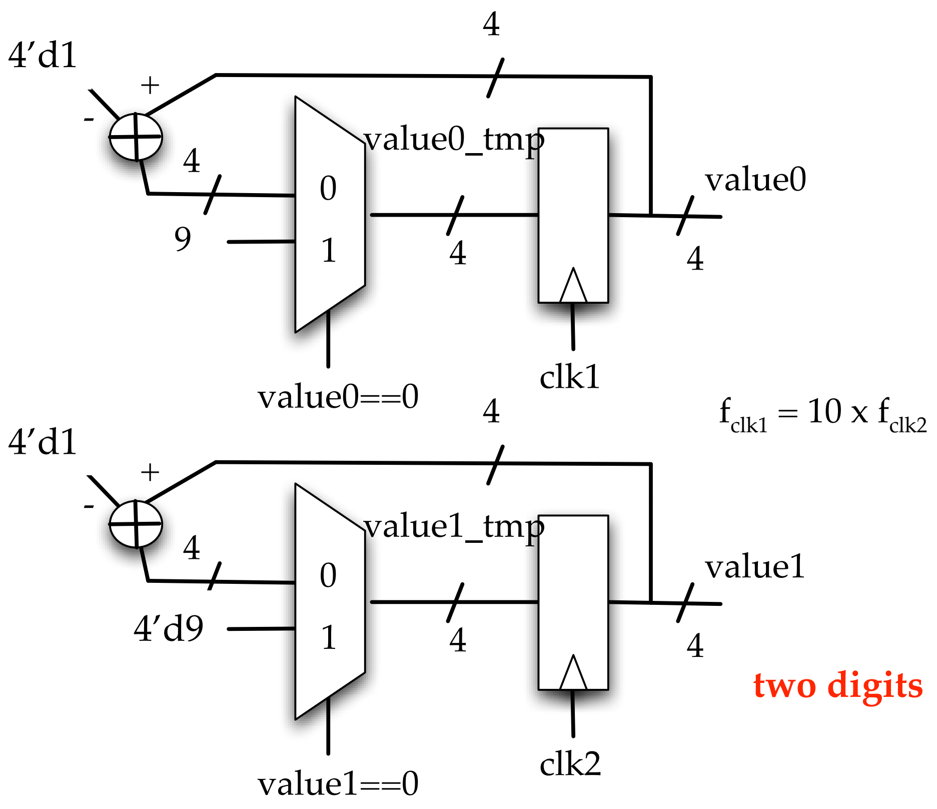
- Use MUX for conditional load to the DFFs



- **Do not** use *initial*

BCD Down-Counter

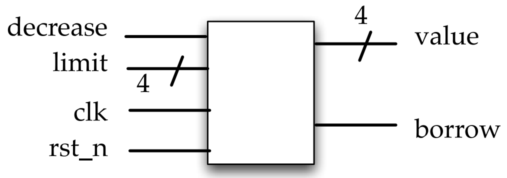




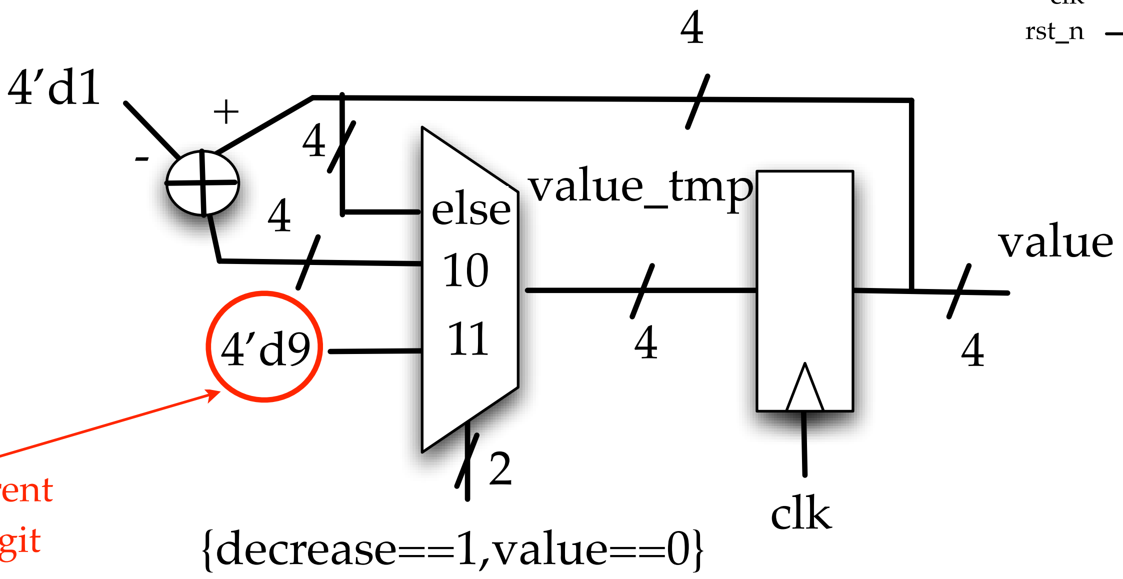
```
// Combinational logics
always @*
  if (value==`BCD_ZERO)
    value_tmp = `BCD_NINE;
  else
    value_tmp = value - `INCREMENT;

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

clk1 and clk2 needed to be synchronized in frequency and phase!!

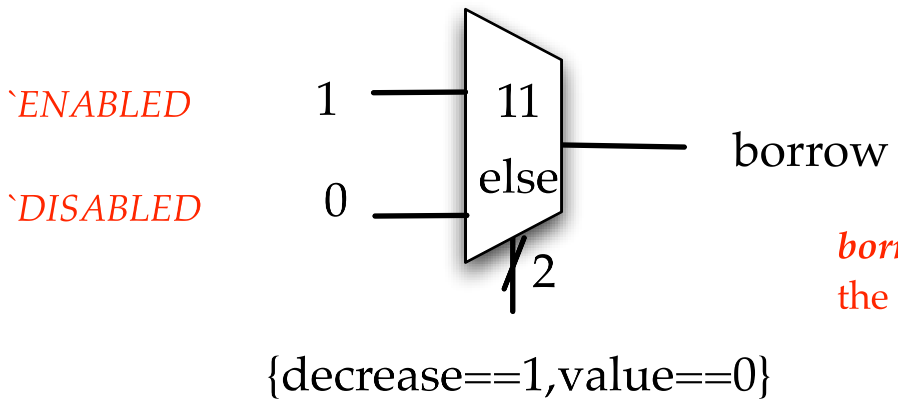
BCD Down-Counter



How to use one single clock for different digit counting?



Use *limit* as different ceiling for each digit



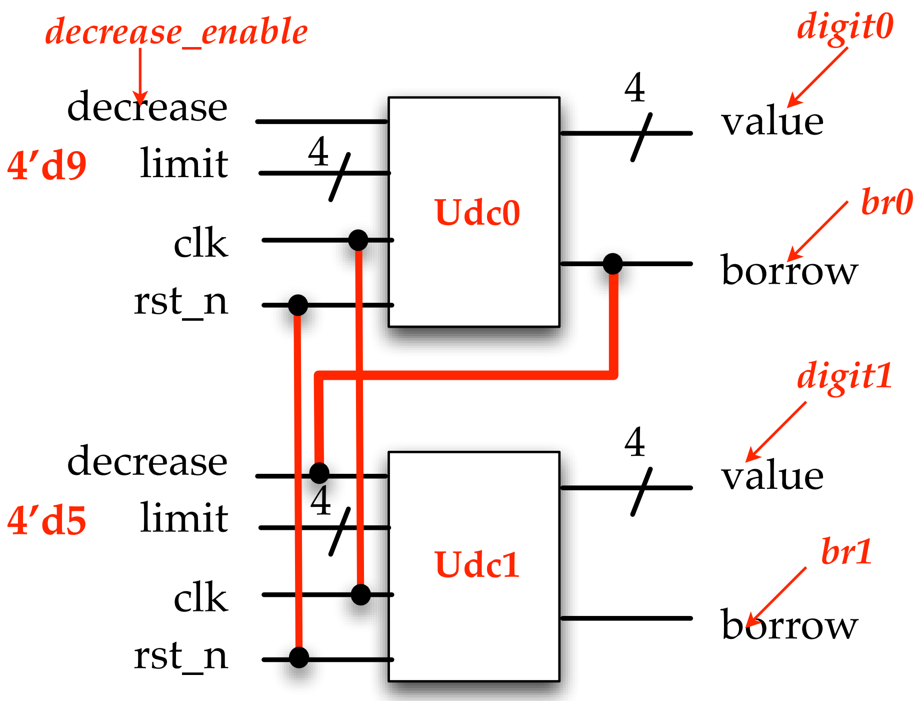
borrow signal in lower digit is the *decrease* control in the upper digit

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
```

2-digit 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
);
```

counting from 59 to 0