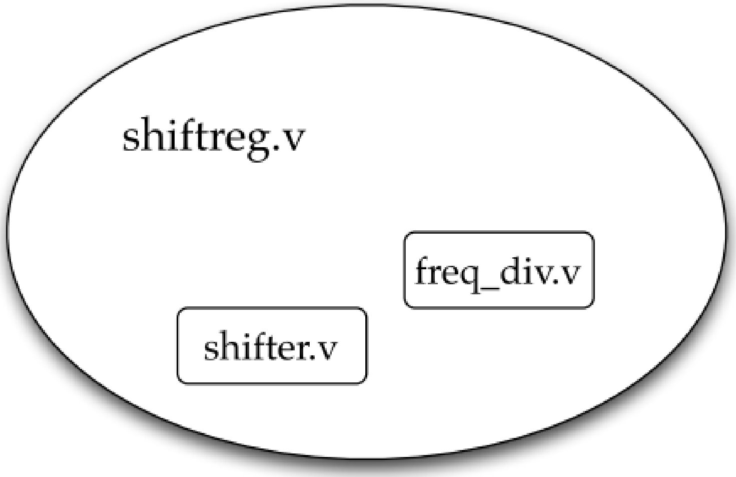
# Shifter

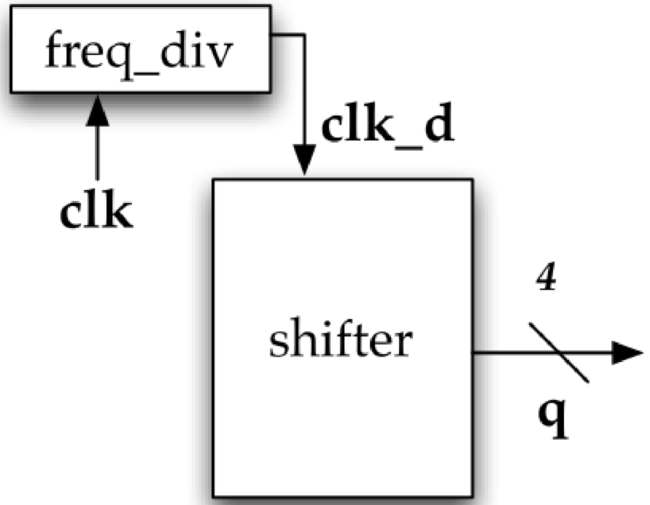
黃元豪
Yuan-Hao Huang

國立清華大學電機工程學系
Department of Electrical Engineering
National Tsing-Hua University

## Modularized Shift Register Design

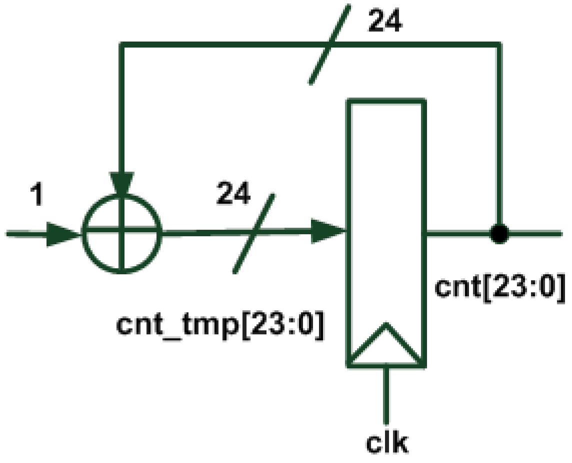
# Shift Register

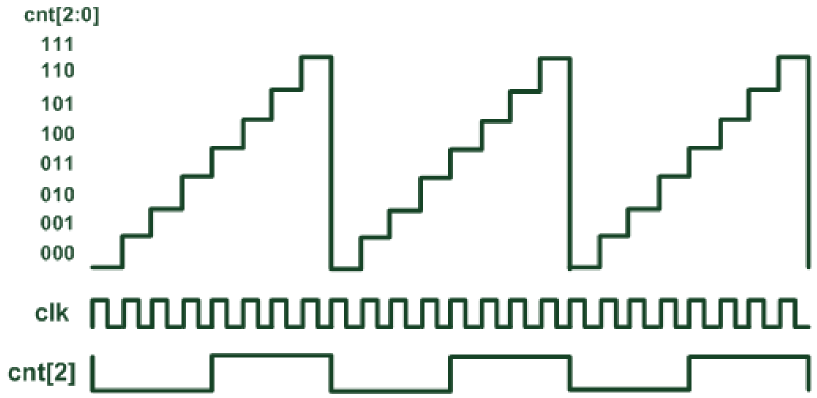




# Frequency Divider

```
`define FREQ_DIV_BIT 27
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;

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

// 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
```

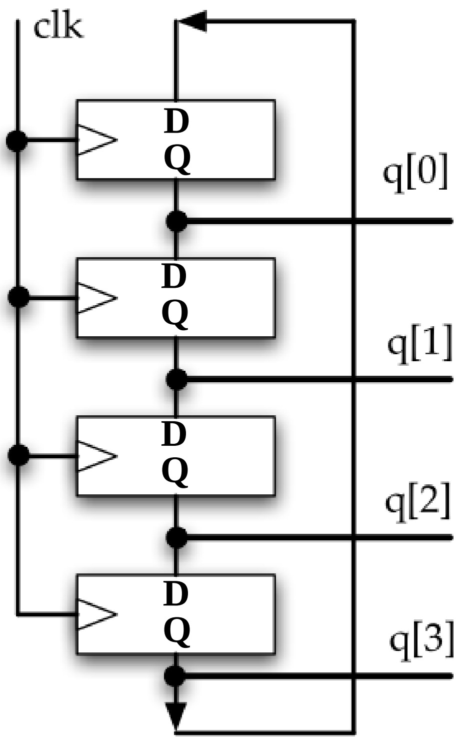




# shifter.v

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

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

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

// Sequential logics: Flip flops
always @(posedge clk or posedge rst)
  if (rst)
  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 q = 0101



# shiftreg.v

```
`define BIT_WIDTH 4
module shiftreg(
  q,  // LED output
  clk, // global clock
  rst  // active high reset
);

output [`BIT_WIDTH-1:0] q;  // LED output
input clk;  // global clock
input rst;  // active high 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(rst)  // active high reset
);

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

endmodule
```

# FAQ: Module Reuse

```
/// combinational circuits
new N0(.c(Q_temp), .a(A) , .b(B));

/// sequential flip-flops
always @(posedge clk or posedge rst)
If(rst)
Q<=0;
else
Q<=Q_temp;


module new(a,b,c);
input a,b;
output c;
……..
endmodule
```

```
/// combinational circuits

/// sequential flip-flops
always @(posedge clk or posedge rst)
If(rst)
Q<=0;
else
new N0(.c(Q), .a(A) , .b(B));  ✗


module new(a,b,c);
input a,b;
output c;
……..
endmodule
```

## FAQ: Black Box

- Your module is reported as “black box” if the module is redundant.
  - Example : a module without output port.

```
`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;

// Dividied 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
```