SC1005
Digital Logic

Recap and Discussion

Lecture 19 and 20

Sequential Circuits in Verilog

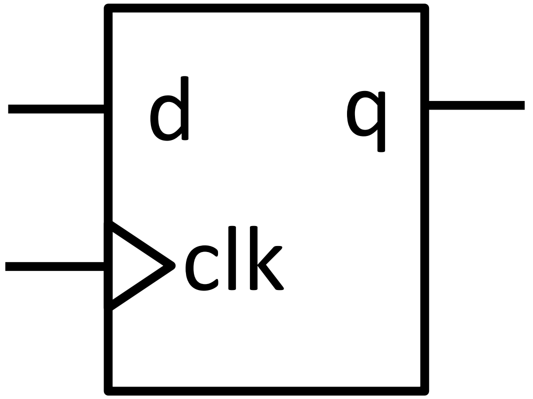
Summary of Lecture 19

- Sequential Circuits in Verilog
  - Registers in Verilog
  - Clock and Reset
  - Synchronous always block
  - Counters

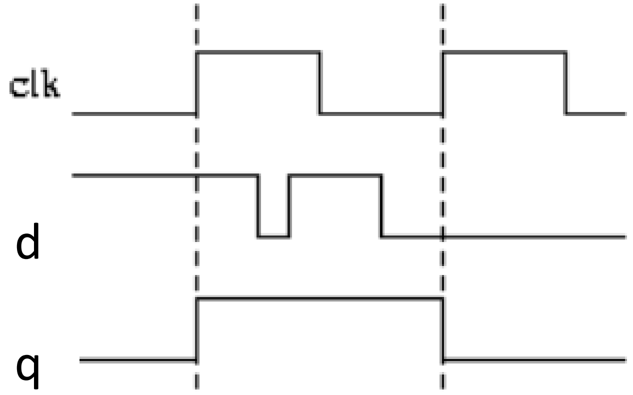
Summary of Lecture 20

- Sequential Circuits in Verilog
  - Shift Registers
  - Serial Data Transfer
  - Memory
  - What gets synthesized

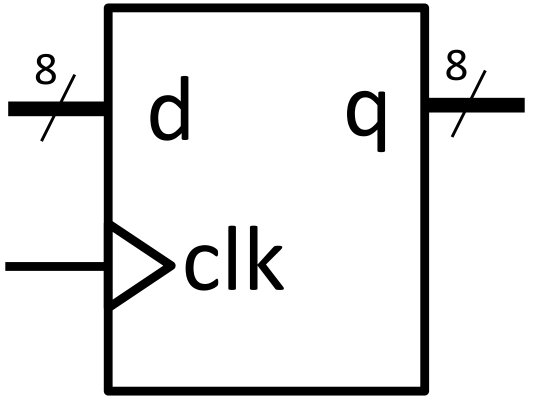
Recap: Registers in Verilog



```
module d_reg (input d, clk,
                output reg q);

always @ (posedge clk)
     q <= d;

endmodule
```

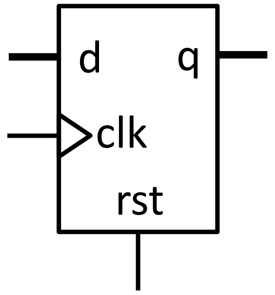


Note the use of the non-blocking assignment. **e.g. <=**
More on that later

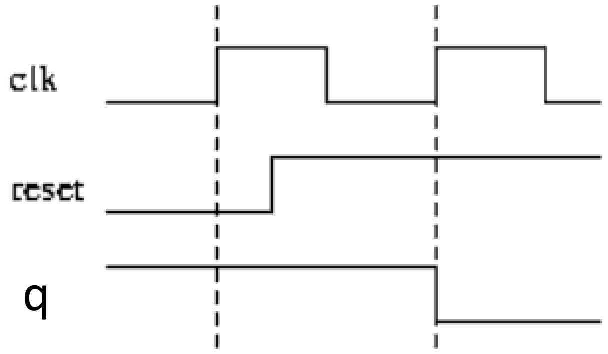


```
module d_reg (input clk, input [7:0] d,
                output reg [7:0] q);

always @ (posedge clk)
     q <= d;

endmodule
```

Recap: Synchronous Reset vs. Asynchronous Reset

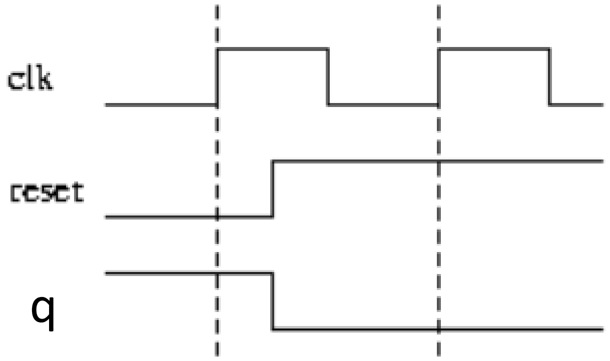


Synchronous Reset



```
module d_reg (input [7:0] d,
              input clk, rst,
              output reg [7:0] q);

always @ (posedge clk)
begin
    if (rst)       // same as (rst==1'b1)
        q <= 8'b0000_0000;
    else
        q <= d;
end

endmodule
```

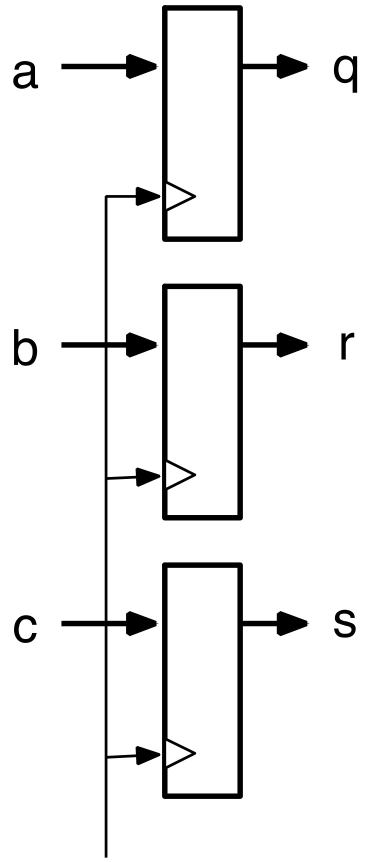
Asynchronous Reset



```
module d_reg (input [7:0] d,
              input clk, rst,
              output reg [7:0] q);

always @ (posedge clk or posedge rst)
begin
    if (rst)
        q <= 8'b0000_0000;
    else
        q <= d;
end

endmodule
```

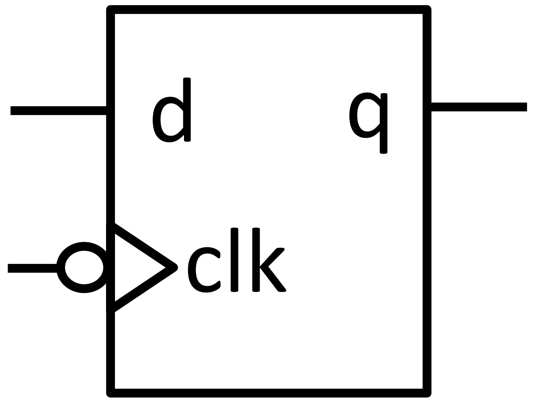
In FPGA Design we try to AVOID asynchronous reset.

Recap



```
module multireg (input [7:0] a, b, c,
                 input clk, rst,
                 output reg [7:0] q, r, s);

always@(posedge clk)
begin
    if(rst) begin
        q <= 8'b0000_0000;
        r <= 8'b0000_0000;
        s <= 8'b0000_0000;
    end else begin
        q <= a;
        r <= b;
        s <= c;
   end
end
endmodule
```

- Each *assignment* in a synchronous always block results in a *register*
- All **synchronous always blocks** should use the same clock signal
- For falling-edge triggered, use always@(**negedge** clk)

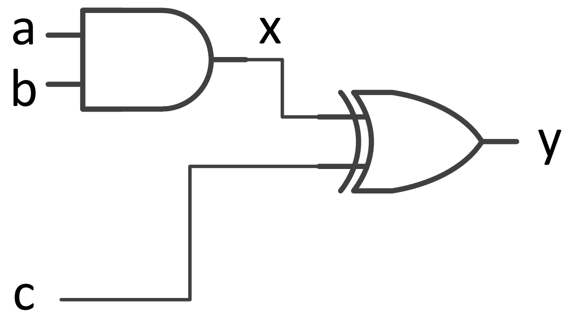


Recap: Assignments in Always Blocks

- For ***combinational always blocks***, we *always* use a blocking assignment (=), and **order matters**
- For ***synchronous always blocks***, we *always* use non-blocking assignments (<=), and **order does not matter**

Combinational always block

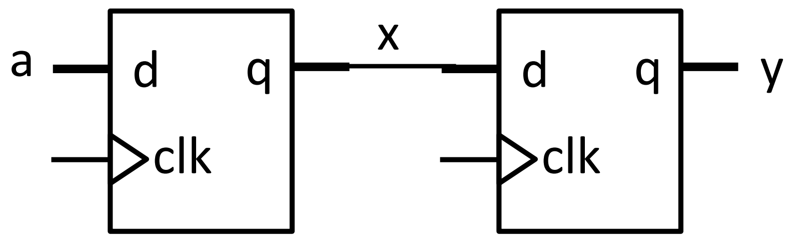
```
reg x, y;

always @ *
begin
    x = a & b;
    y = x ^ c;
end
```



Synchronous always block

```
reg x, y;

always@(posedge clk)
begin
    x <= a;
    y <= x;
end
```

Each non-blocking *assignment* in a synchronous always block results in a *register*



Exercise 1

For the Verilog code shown, if Rst=0 and the current circuit state is *ABC* = 001, what is the next state?

A. 111
B. 001
C. 100
D. 010

```
module Ex1 (input  Clk, Rst,
            output reg A, B, C);

  always@(posedge Clk)
  begin
    if (Rst) begin
      C <= 1'b1;
      B <= 1'b0;
      A <= 1'b0;
    end
    else begin
      C <= B;
      B <= A;
      A <= A ^ B ^ C;
    end
  end
endmodule
```

Ans: C. 100
A is shifted to B, B is shifted to C,
and the new A is A XOR B XOR C = 1.

Sequential Verilog

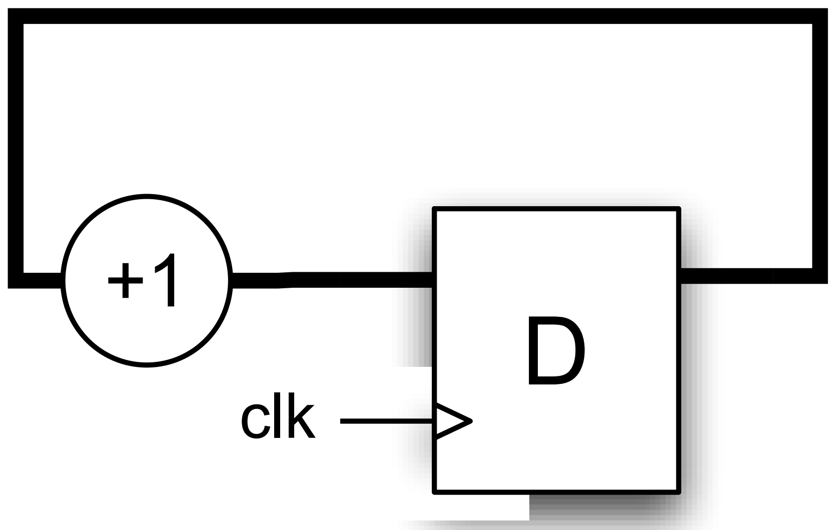
- While it is possible to describe general sequential blocks in Verilog, it is generally used to describe ***synchronous*** circuits, i.e. **edge-triggered components**
  - Counter
  - Shift Register
  - Serial to parallel converter
  - Parallel to serial converter
  - First-In-First-Out buffers
  - …
- The Vivado Synthesis tool will generally convert designs to D-type flip-flops/registers

Synchronous Counters: Binary Counters

- At each rising edge, we pass through the incremented value of the current count

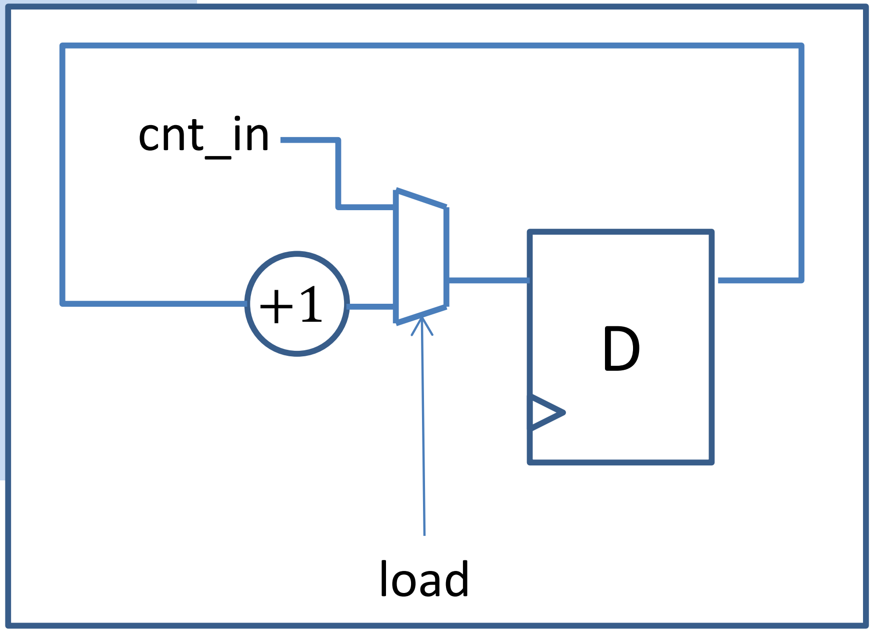
```
0 0 0   ⇐ Reset
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
0 0 0
0 0 1
  ⋮
```

```
module simplecnt (input clk, rst,
                  output reg [2:0] q);

always @ (posedge clk)
begin
    if (rst)
        q <= 3'b000;
    else
        q <= q + 1'b1;
end

endmodule
```

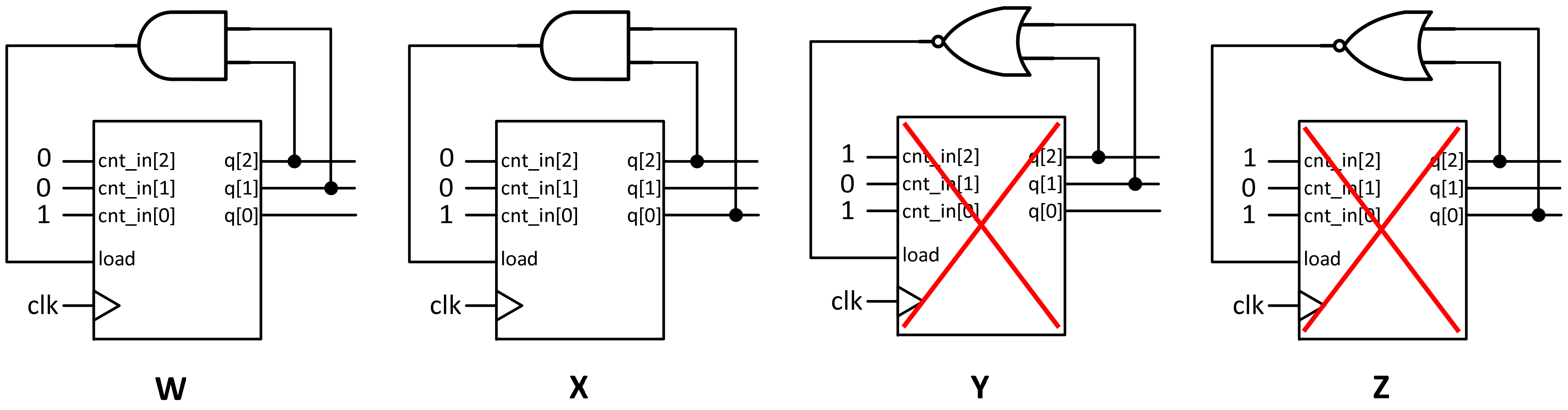


Synchronous Counters: Counter with load

```
module simplecnt (input clk, rst, load,
                  input [3:0] cnt_in,
                  output reg [3:0] q);

always @ (posedge clk)
begin
    if (rst)
        q <= 4'b0000;
    else
        if (load)
            q <= cnt_in;
        else
             q <= q + 1'b1;
end
endmodule
```



Exercise 2



- Using an up-counter with load capabilities, which circuit will generate a counting sequence: 1, 2, 3, 4, 5, 6, 1, 2 …?

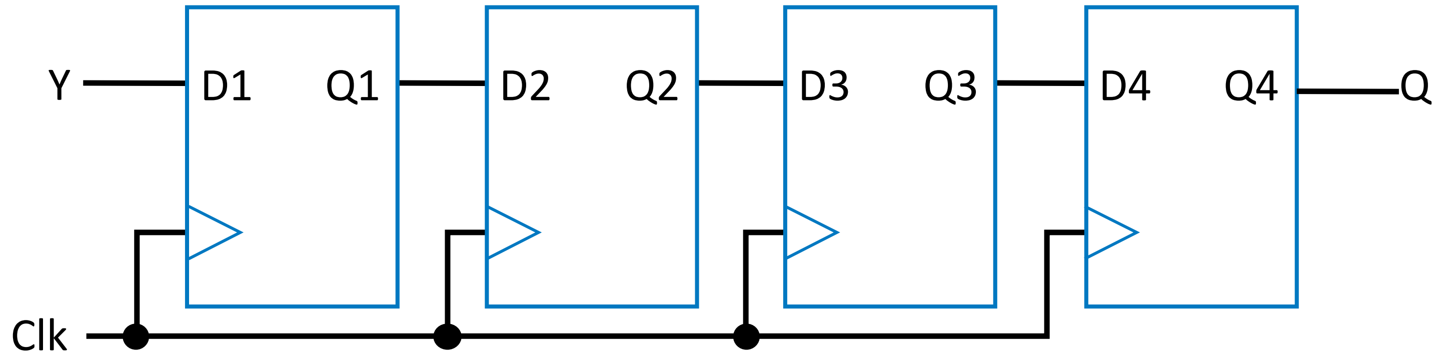
  You can assume that the initial state of the counter is 001_2.

[✓ A. W]
B. X
C. Y
D. Z
E. None of the above

Ans: Circuit W
A binary counter with initial state equal to 001 and with *load* = 0, will naturally count from 1 to 6.
At 6, the counter needs to roll over back to 1 so that the sequence will repeat. To achieve this we need to detect 6 and use it to load 1. Circuit W will do this by using q[2]=q[1]=1 to generate the *load* signal which will load 001 back into the counter on the next *clk* edge.

Recap: Shift Registers in Verilog

- Shift registers pass a single input through a chain of flip-flops



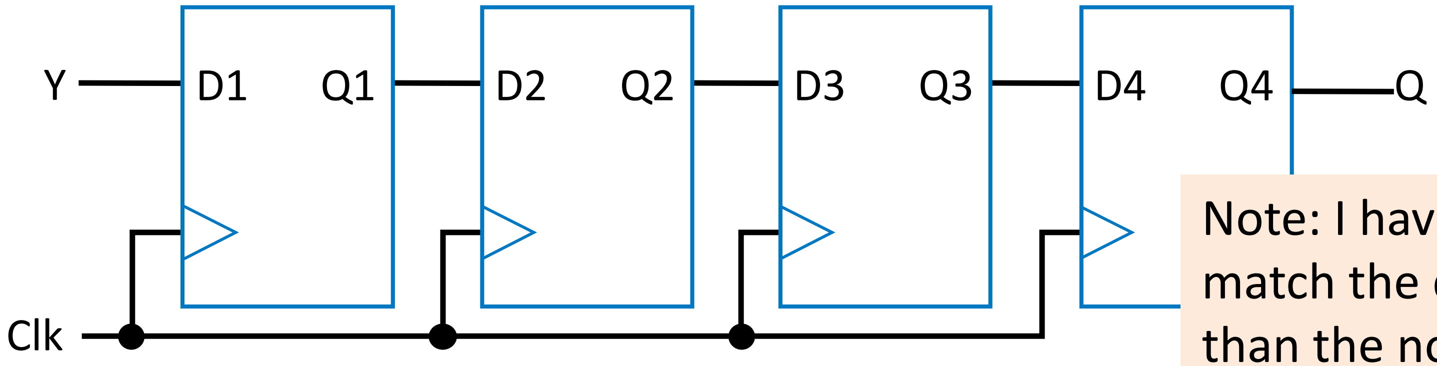
```
module shiftreg (input clk, y,
                 output reg q);

   reg q1, q2, q3;

   always@(posedge clk)
   begin
      q1 <= y;
      q2 <= q1;
      q3 <= q2;
      q  <= q3;
   end

endmodule
```

Order does not matter

```
module shiftreg (input  clk, y,
                 output reg q_out);

   reg [3:1] q;

   always@(posedge clk)
   begin
      q[1]   <= y;
      q[3:2] <= q[2:1];
      q_out  <= q[3];
   end

endmodule
```

Recap: Shift Registers in Verilog

- Shift registers pass a single input through a chain of flip-flops



Note: I have defined ***q*** to match the diagram, rather than the normal **[3:0] q;**.

```
module shiftreg (input clk, y,
                 output reg q);

   reg q1, q2, q3;

   always@(posedge clk)
   begin
      q1 <= y;
      q2 <= q1;
      q3 <= q2;
      q  <= q3;
   end

endmodule
```

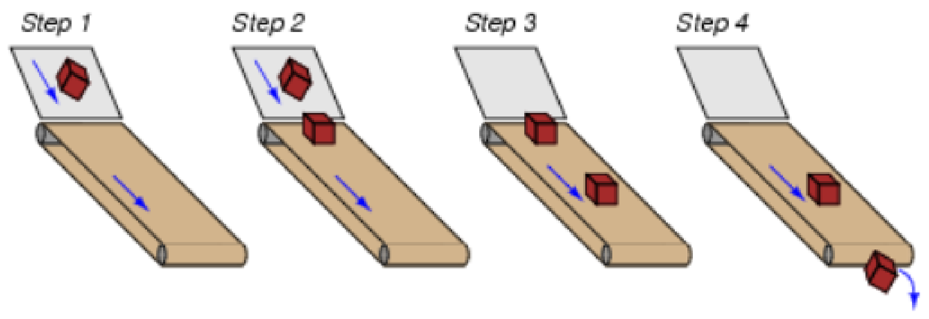
Order does not matter

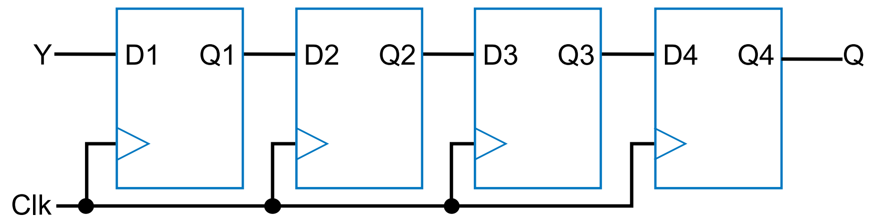
```
module shiftreg (input  clk, y,
                 output q_out);

   reg [4:1] q;

   assign q_out = q[4];

   always @ (posedge clk)
   begin
      q[4:1] <= {q[3:1], y};
   end

endmodule
```

An even better way

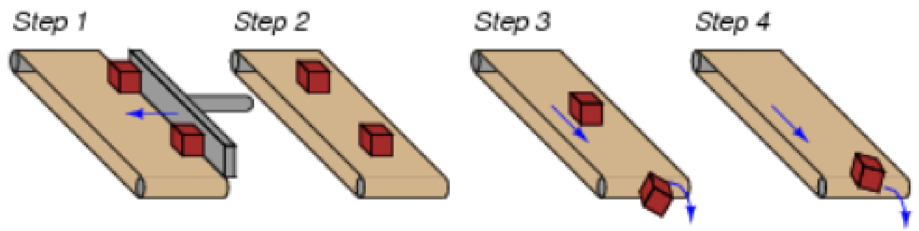
Shift Registers (Analogy)

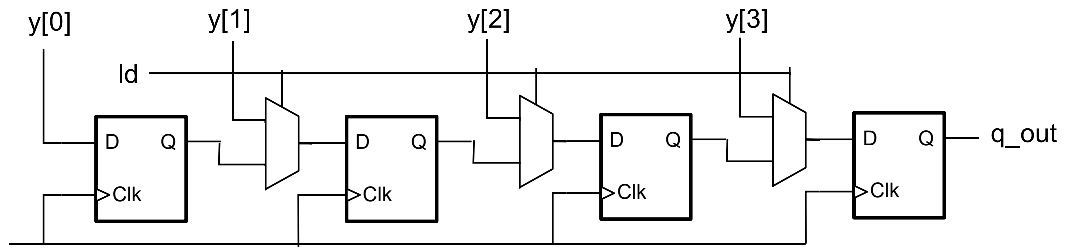
Serial-In Serial-Out



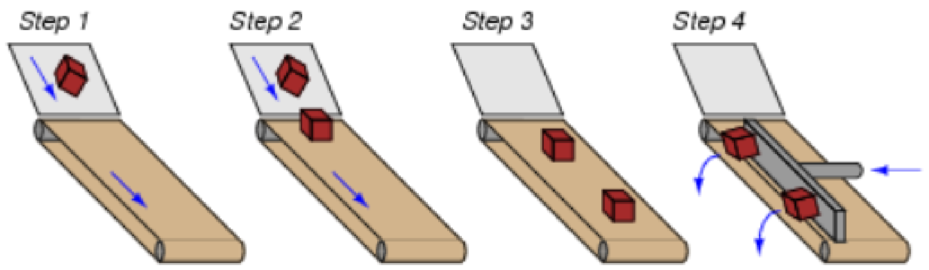


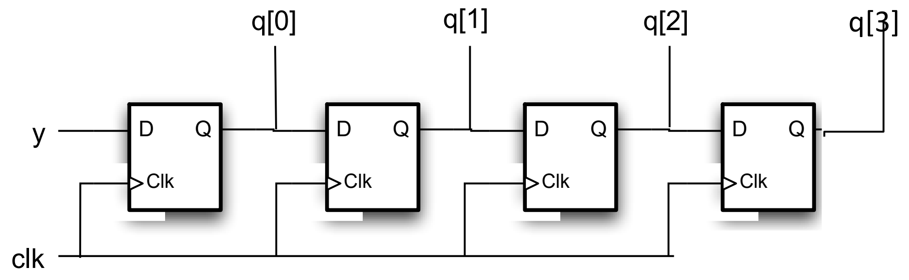
Parallel-In Serial-Out





Serial-In Parallel-Out



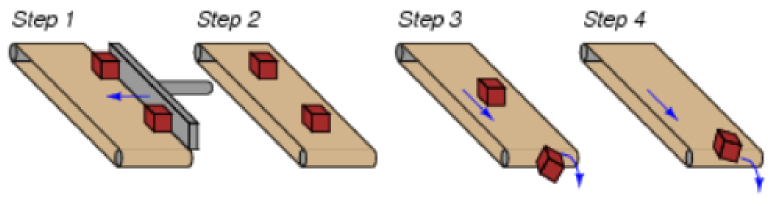


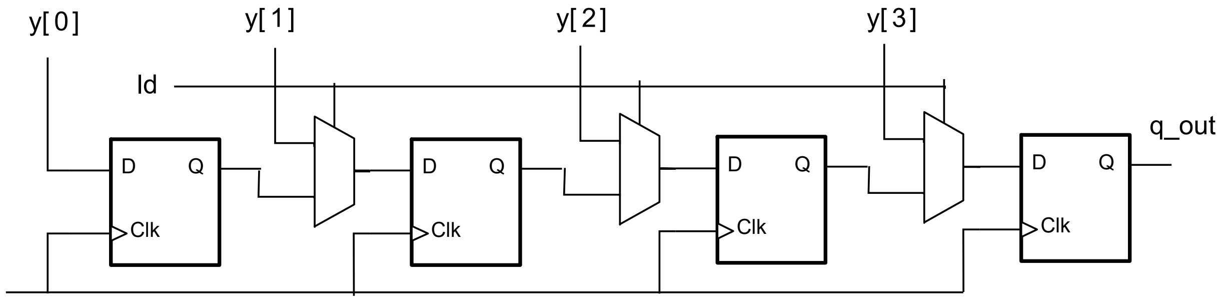
Source: http://www.allaboutcircuits.com/worksheets/shift-registers/

Shift Registers (Parallel-In Serial-Out)

- Whenever *ld* is high, the shift register is loaded with the value on *y*
- Can also be used as a conventional shift register when *ld* is always low.

```
module shiftreg ( input clk, ld,
                  input [3:0] y,
                  output q_out);
   reg [3:0] q;

   always @ (posedge clk)
   begin
      if (ld)
         q <= y;
      else
         q[3:0] <= {q[2:0], y[0]};
   end
   assign q_out = q[3];

endmodule
```

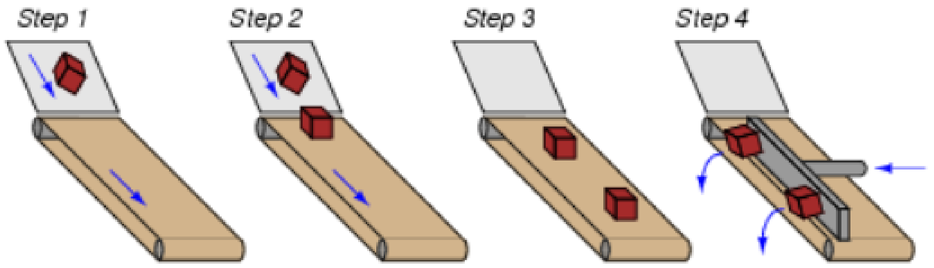


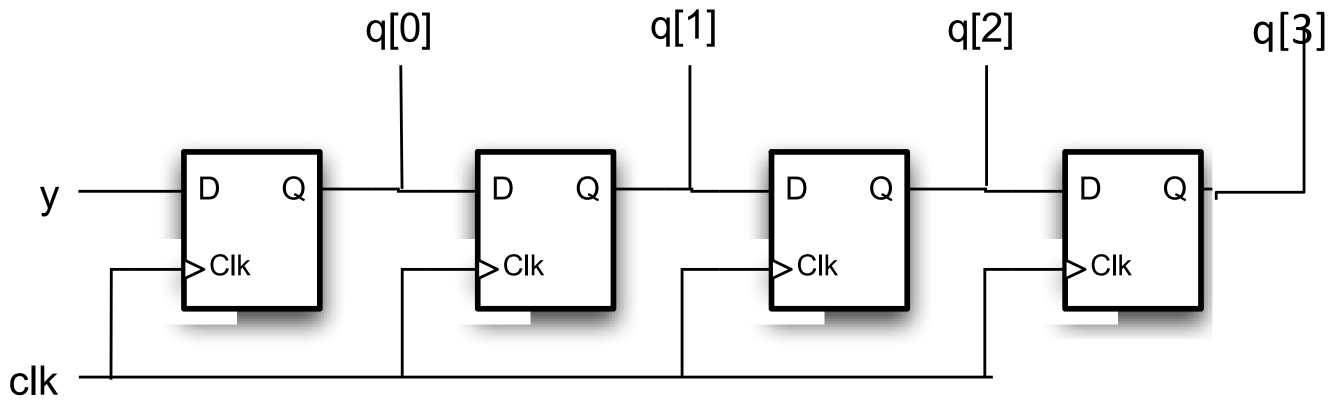


Shift Registers (Serial-In Parallel-Out)

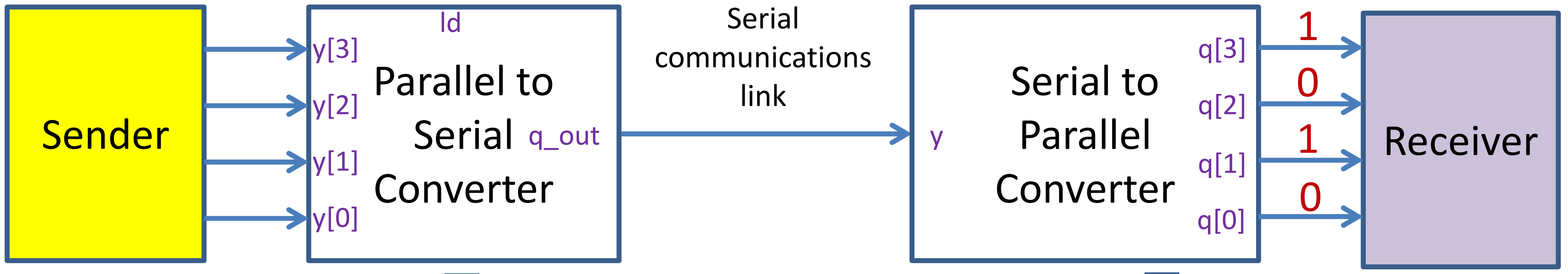
- We can also access the outputs of the flip flop all at once

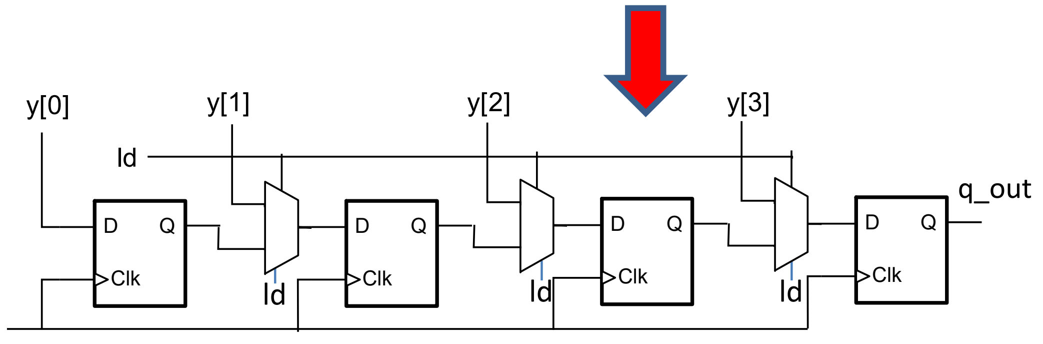
```
module shiftreg ( input clk, y,
                  output reg [3:0] q);

   always@(posedge clk)
   begin
      q[3:0] <= {q[2:0], y};
   end

endmodule
```

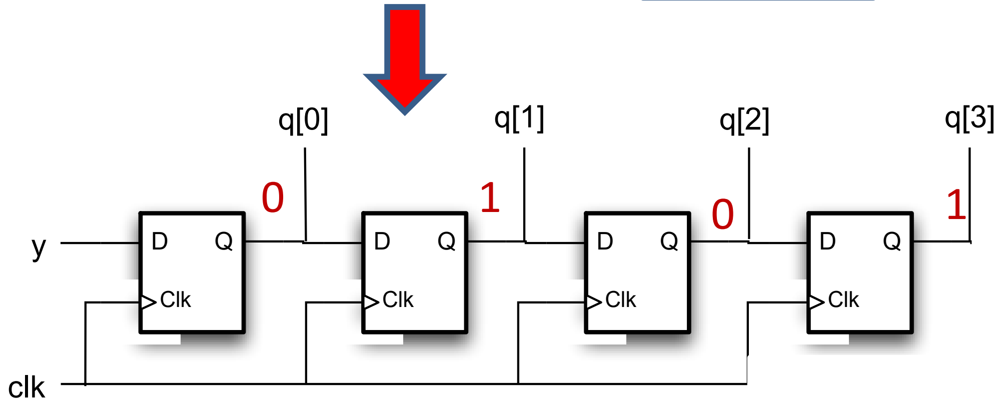




Serial Data Transfer


Sender
ld
y[3]
y[2]
y[1]
y[0]
Parallel to Serial Converter
q_out
Serial communications link
y
Serial to Parallel Converter
q[3]
q[2]
q[1]
q[0]
1
0
1
0
Receiver



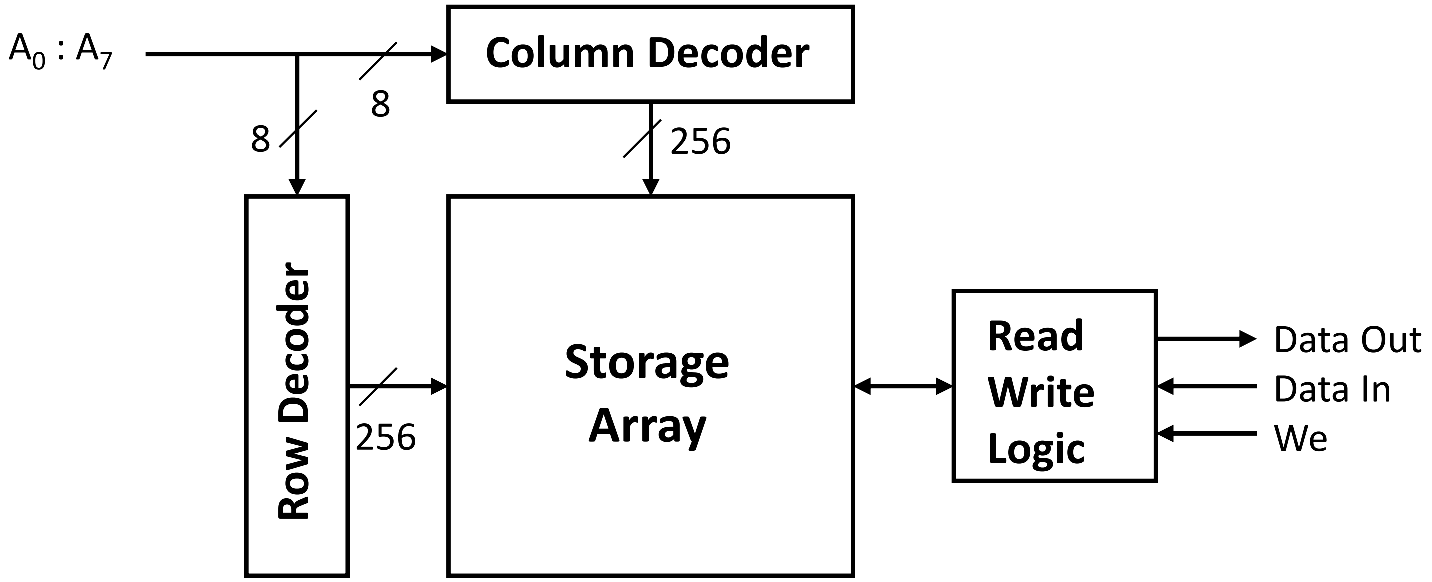


```
module piso4 (input clk, ld, input [3:0] y,
              output q_out);
   reg [3:0] q;

   always@(posedge clk)
   begin
       if (ld)
           q <= y;
       else
           q[3:0] <= {q[2:0], y[0]};
   end
   assign q_out = q[3];

endmodule
```

```
module sipo4 (input clk, y,
              output reg [3:0] q);

   always@(posedge clk)
   begin
       q[3:0] <= {q[2:0], y};
   end

endmodule
```

Memory

- Memory is an array of registers
  - Can access a single word
    - Need an *Address* signal
  - Can read from and write to
    - Need *Data in*, *Data out* & *Write enable*
    - May have other control signals



Memories in Verilog

- Memories are defined as an array of type ***reg***.

An 8-bit x 1024 memory array

```
    reg [7:0] mem_array [0:1023];
```

A 32-bit x 8K memory array

```
    reg [31:0] mem_array [0:8191];
```

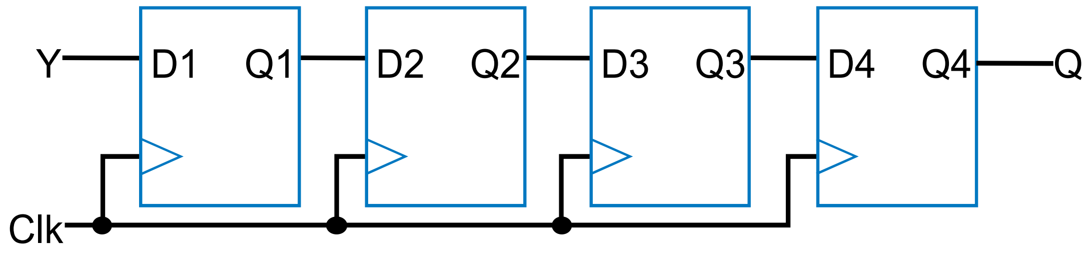
- A simple 8-bit x 1024 single port RAM module in Verilog

```
module dmem #(parameter WORD_SIZE = 8, ADD_SIZE = 10) (input clk, we,
          input [WORD_SIZE-1:0] din, input [ADD_SIZE-1:0] add,
          output [WORD_SIZE-1:0] dout);

   reg [WORD_SIZE-1:0] RAM [0:(1<<ADD_SIZE)-1];    // Define mem array

   assign dout = RAM[add];                     // Combinational read
   always @ (posedge clk)
      if (we) RAM[add] <= din;                 // Synchronous write

endmodule
```

What does 1<<ADD_SIZE produce?
Ans: 1<<10 = 100_0000_0000 = 1024

What Gets Synthesized?

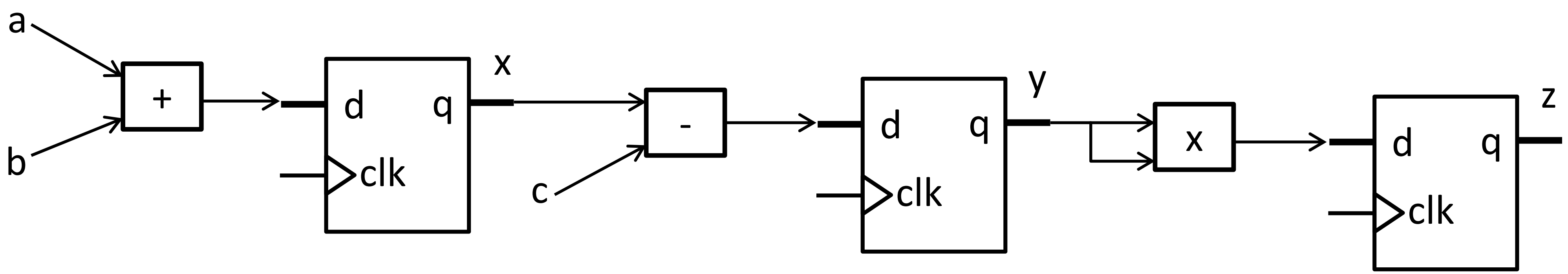
- Important to understand what is synthesized from a synchronous always block
- Each (non-blocking) assignment results in a register, with its input connected to a circuit based on the right hand side, and the output connected to the signal the left hand side
- **Every assignment is a register (i.e. synchronous)**



```
module shiftreg (input clk, y,
                 output reg q);
   reg q1, q2, q3;
   always@(posedge clk) begin
      q1 <= y;
      q2 <= q1;
      q3 <= q2;
      q  <= q3;
   end
endmodule
```

What Gets Synthesized?

```
always@(posedge clk)
begin
    x <= a + b;
    y <= x – c;
    z <= y * y;
end
```

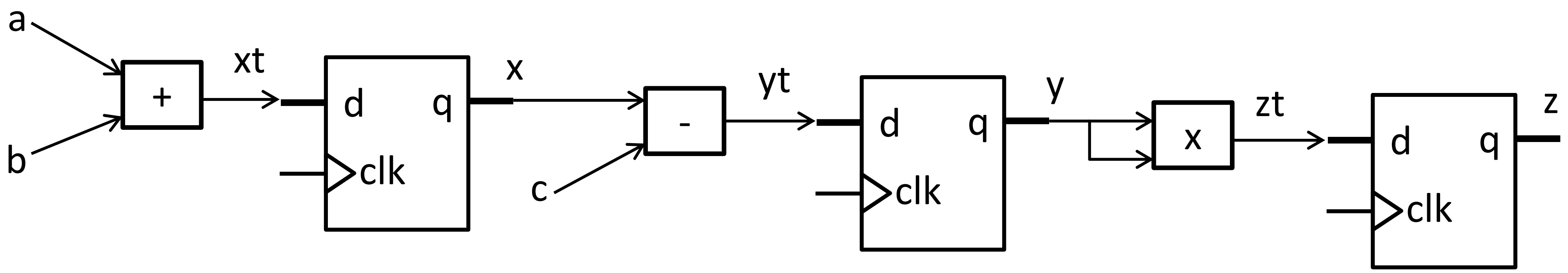


What Gets Synthesized?

- It is also possible to separate the combinational and synchronous parts

```
always @ (posedge clk)
begin
    x <= a + b;
    y <= x – c;
    z <= y * y;
end
```

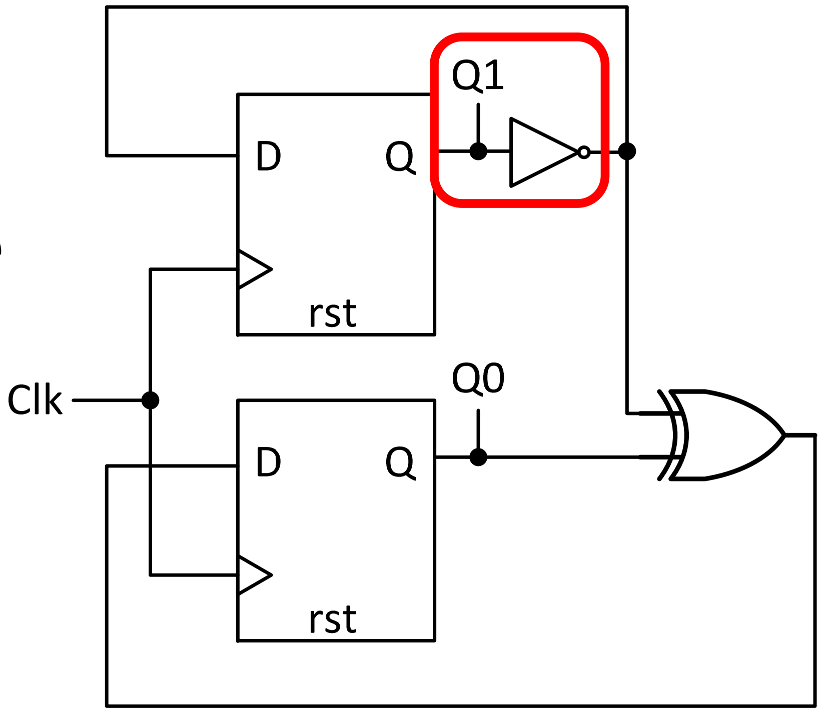
```
always @ (posedge clk)
begin
    x <= xt;
    y <= yt;
    z <= zt;
end

always@*
begin
    xt = a + b;
    yt = x – c;
    zt = y * y;
end
```



Exercise 3

Which of the following is the correct Verilog description of the circuit?

a. X
b. Y
c. Z



X

```
  always@(posedge Clk)
  begin
    if (rst) begin
      Q1 <= 1'b0;
      Q0 <= 1'b0;
    end
    else begin
      Q1 <= ~(Q1);
      Q0 <= ~(Q0) ^ Q1;
     end
   end
endmodule
```

Y

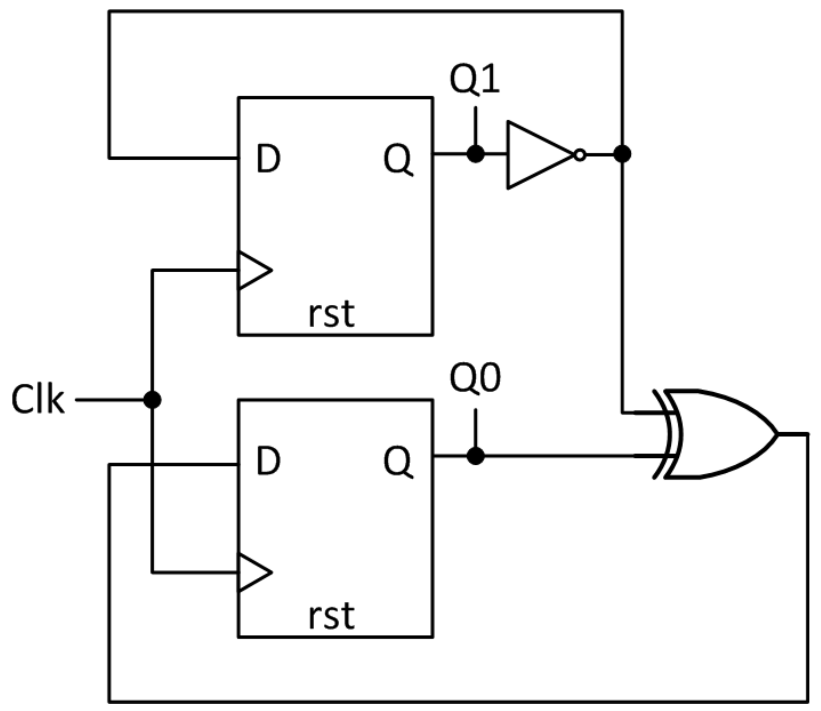
```
  always@(posedge Clk)
  begin
    if (rst) begin
      Q1 <= 1'b0;
      Q0 <= 1'b0;
    end
    else begin
      Q1 <= ~(Q1);
      Q0 <= ~(Q1) ^ Q0;
     end
   end
endmodule
```

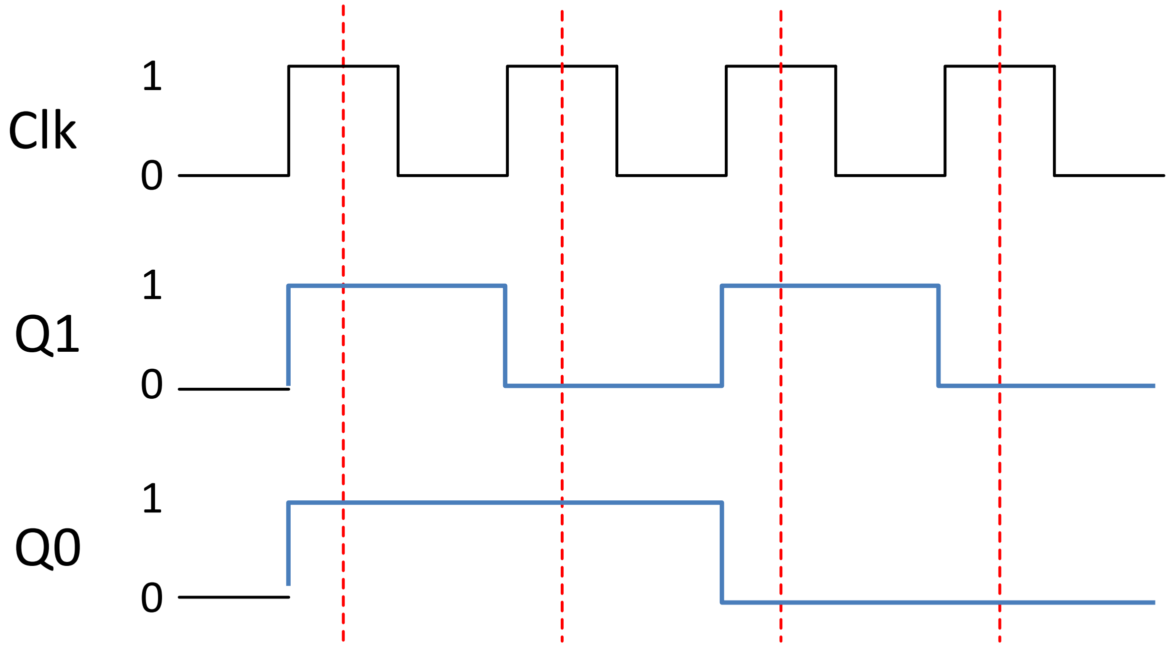
Z

```
  always@(posedge Clk)
  begin
    if (rst) begin
      Q1 <= 1'b0;
      Q0 <= 1'b0;
    end
    else begin
      Q1 <= ~(Q0);
      Q0 <= ~(Q0) ^ Q1;
     end
   end
endmodule
```

Exercise 4

What is the correct output sequence of Q1 and Q0 observed at the dash lines?





A. Q1: 1010; Q0: 0011
B. Q1: 0101; Q0: 1010
C. Q1: 1010; Q0: 1100
D. None of the above

Ans: C
Q1 will just toggle

Selected Past Exam Questions

CE/CZ1005 2018-2019 Semester 1 (Nov/Dec 2018)

Q4(a) Write the Verilog code for an 8-bit up/down counter (controlled by a single bit *up* input). Use a ***parameter*** statement to define the counter width (e.g. 8 bits). (8 marks)

ANS:

```
module cnt8bit #(parameter SIZE=8) (input clk, rst,
                up, output reg [SIZE-1:0] count);

   always @ (posedge clk)
   begin
      if (rst)
         count <= 0;              //or 8'h00;
      else if (up)
         count <= count + 1'b1;   //or count+1;
      else
         count <= count - 1'b1;
   end
endmodule
```

Selected Past Exam Questions

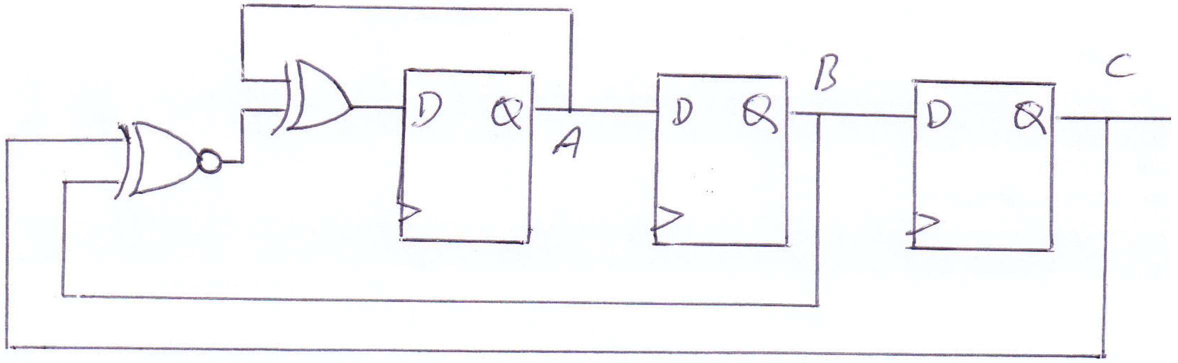
CE/CZ1005 2018-2019 Semester 2 (Apr/May 2019)

Q4(a) Fig Q4a shows a sequential Verilog module.

(i) Draw the circuit described by the Verilog module in Figure Q4a with D-type flip-flops and logic gates. 5 marks

```
module mod1 (input clk, rst, output reg A, B, C);

  always@(posedge clk)
    begin
      if (rst) begin
        A <= 1'b0;
        B <= 1'b0;
        C <= 1'b0;
      end
      else begin
        C <= B;
        B <= A;
        A <= A ^ ~(B ^ C);
      end
    end
endmodule
```

Figure Q4a

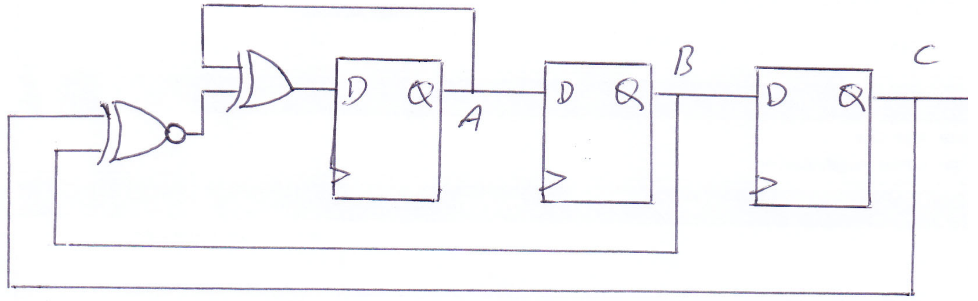


Selected Past Exam Questions

CE/CZ1005 2018-2019 Semester 2 (Apr/May 2019)

Q4(a) Fig Q4a shows a sequential Verilog module. 8 marks

(ii) Complete the timing diagram shown in Figure Q4b for the Verilog module in Figure Q4a by drawing the waveforms for *A*, *B* and *C*. Assume that *rst* = 0.



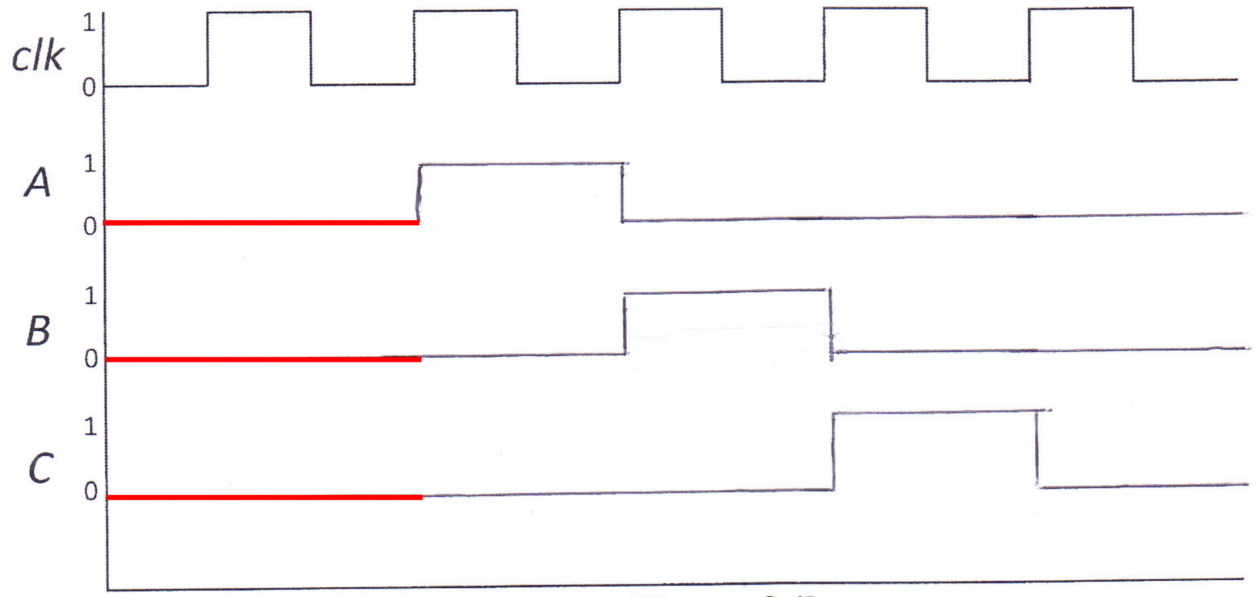


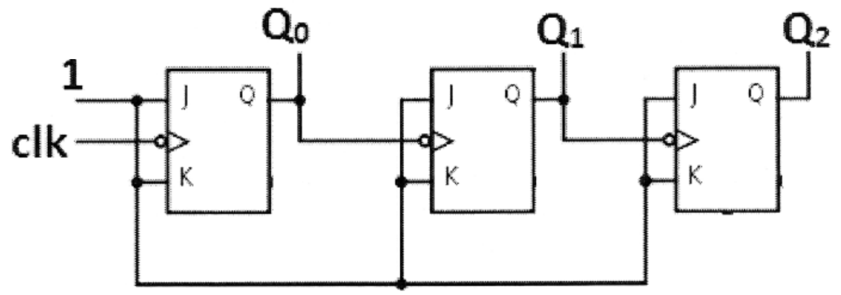
Figure Q4b

Selected Past Exam Questions

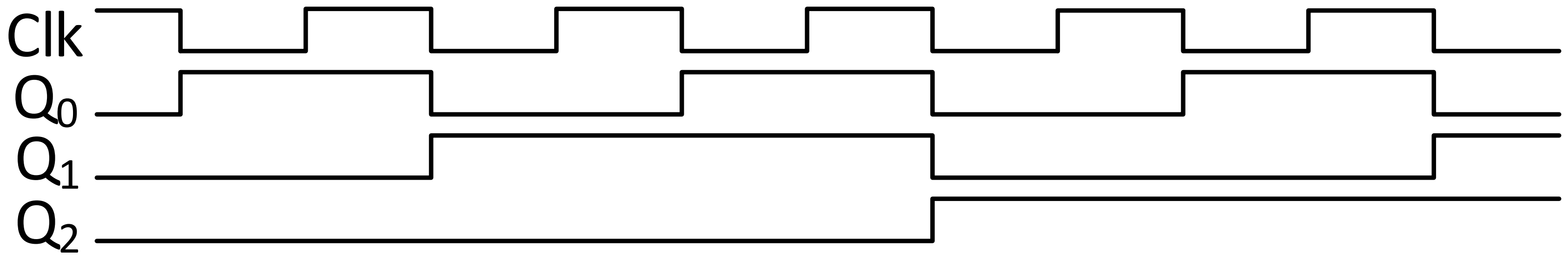
CE/CZ1005 2017-2018 Semester 1 (Nov/Dec 2017)

Q4(b) Figure Q4a shows a binary ripple counter. Sketch the timing diagram of each of the outputs Q_0, Q_1 and Q_2 for six clock cycles. Assume the initial state of $Q_2 Q_1 Q_0 = 000$.

(7 marks)



ANS: Firstly, the counter is negative edge triggered. Second, for a JK FF, when J=K=1, the Q output toggles.



Selected Past Exam Questions

CE/CZ1005 2016-2017 Semester 2 (Apr/May 2017)

Q4(b) Write the Verilog module *PISO* which is a parallel-in/serial-out register. The inputs to the module are *IN[3:0]*, *CLK* and *LOAD*. When *LOAD* is high, the register is loaded with the value of *IN*. The output of this module is *OUT*.

(7 marks)

YUK. Upper case!!!!

ANS:

```
module PISO (input CLK, LOAD, input [3:0] IN,
                 output OUT);
   reg [3:0] q;

   always @ (posedge CLK)
   begin
      if (LOAD)
         q <= IN;
      else
         q[3:0] <= {q[2:0], 1’b0};
   end
   assign OUT = q[3];

endmodule
```

Also Note that there is no real need for reset. Adding a reset is OK.

Note: There is no need to feed IN[0] into the LSB. Zero is fine.