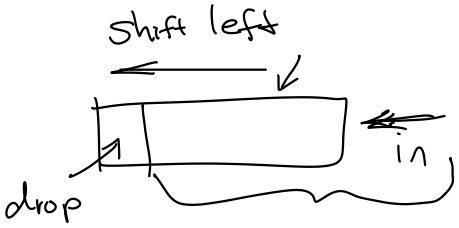
# Applications of State Machines




**Exercise 1**: The example above is an N-bit shift register that shifts the bits right. Draw a block diagram and write the Verilog for a 6-bit shift register that shifts left.

```
module shift;
logic [5:0] q;
logic in;
always_ff @(
    q <= {q[4:0], in};
6
q <= {q << 1, in};
```

left shift

6 1

7 bit

wrong - this would always shift in a zero!

```
q <= {in, q >> 1};
```

right shift

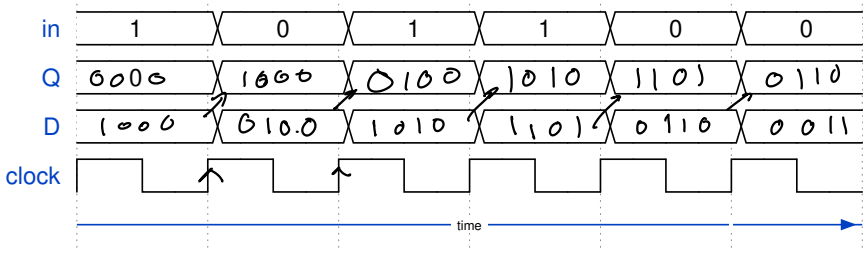
1 6

```
end module
```

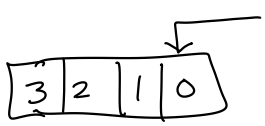
```
q <= {in, q[5:1]};
```

right shift

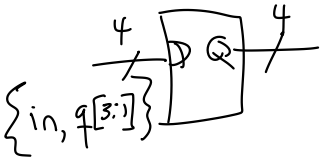
**Exercise 2**:



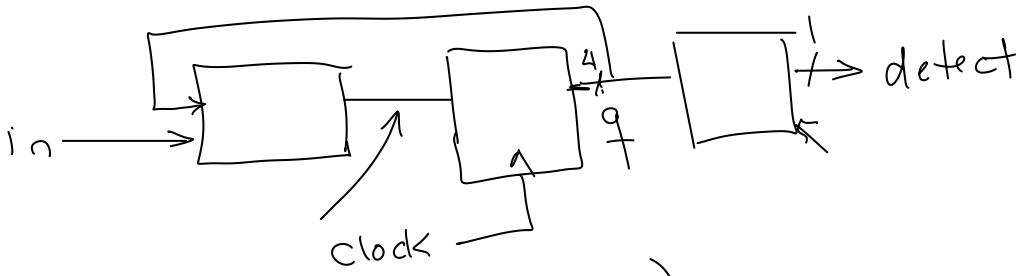

Fill in the diagram above for a 4-bit ($N = 4$) right-shift shift register. Assume the initial value is zero. Which bit is the oldest (first) value in the waveform? Which bit of the shift register holds the oldest value?




oldest ≡ first bit = bit 0



**Exercise 3**: Draw a block diagram and write the Verilog for a circuit that sets an output named **`detect`** high when the sequence of values 1, 1, 0, 1 has appeared on an input named **`in`** on successive rising edges of the clock.




```
always_ff @ (posedge clock)
    q <= {in, q[3:1]};

assign detect = q == 4'b1011;

OR
    = q[3] && !q[2] && q[1] && q[0];
```

**Exercise 4**: How could you modify the code so that **`digits`** is only updated when an **`enable`** input is asserted?

```
always_ff @(posedge clk) digits
   <= { digits[1:3], digit } ;
```

```
<= enable ? { digits[1:3], digit } :
    digits;
```

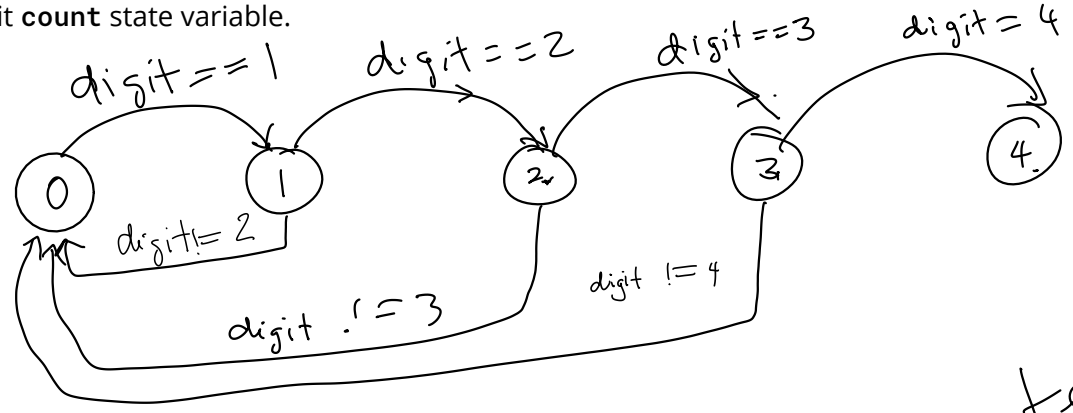
**Exercise 5**: How many states can this state machine have?

$16 \times 16 \times 16 \times 16 = (2^4)^4 = 2^{16}$

$= 65536$

1, 2, 3, 4

**Exercise 6**: Draw the state transition diagram for this simpler implementation. How many states are there? Write the Verilog using a 3-bit **count** state variable.




template

always_ff @ (posedge clk)

state <= // <start state> && <condition> ? <end state> :

state == 0 && digit == 1 ? 1 :

state == 1 && digit == 2 ? 2 :

state == 2 && digit == 3 ? 3 :

state == 3 && digit == 4 ? 4 :

state == 1 && digit != 2 ? 0 :

state == 2 && digit != 3 ? 0 :

state == 3 && digit != 4 ? 0 :

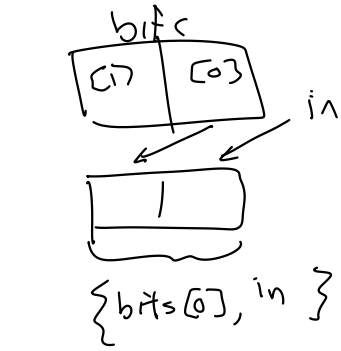
State ;

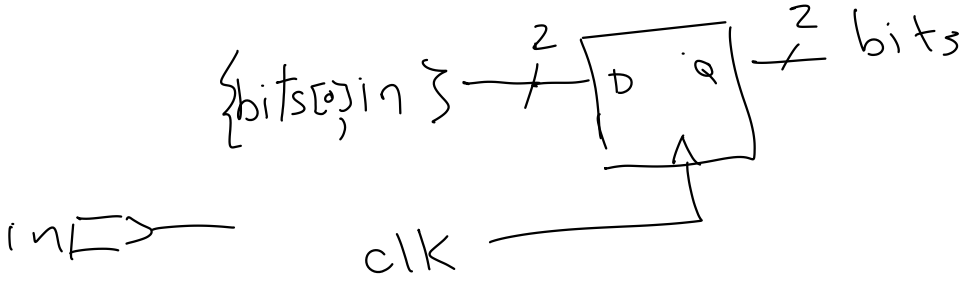
**Exercise 7**: For which states would a **`fell`** output be asserted? A **`rose`** output? Draw the schematic and write the Verilog for this state machine. Assume an input **`in`** and a 2-bit register **`bits`** that holds the two most recent input values.

fell is asserted for state 10

rose is asserted for state 01

Shift left



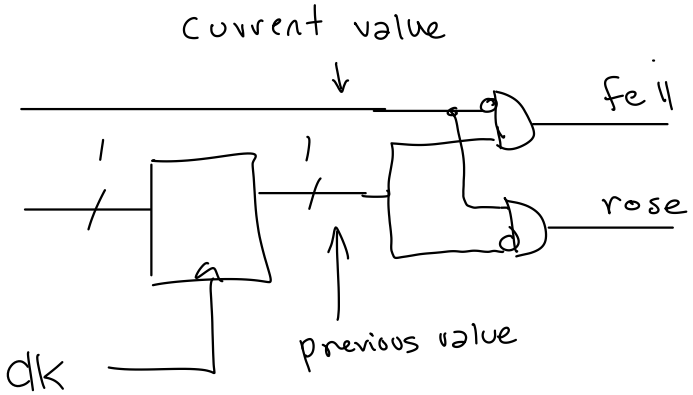



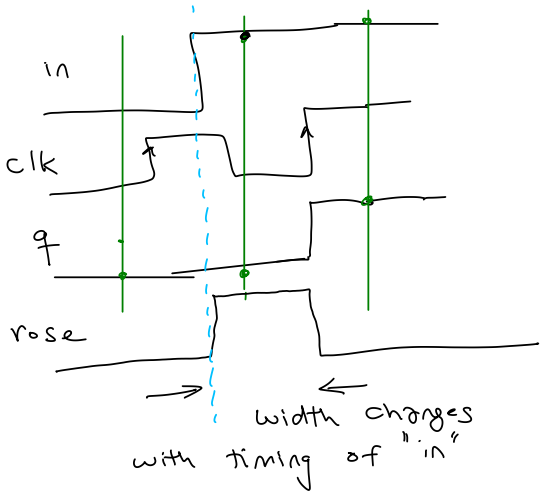

bits == 2'b10 → fell

bits == 2'b01 → rose

```
module edge_detect (
  input logic in, clk,
  output logic rose, fell);

  logic [1:0] bits;

  always_ff @(posedge clk) bits <= {bits[0], in};
  assign rose = bits == 2'b01;
  assign fell = bits == 2'b10;

endmodule
```

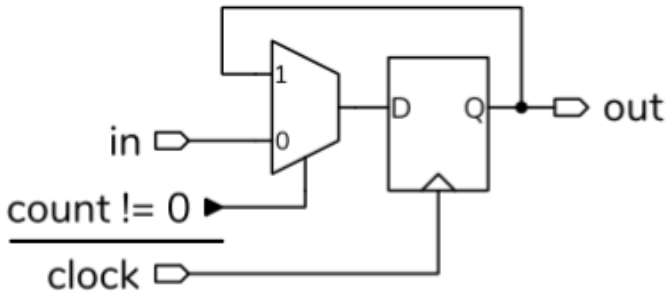
**Exercise 8**: Can you design an edge detector that uses only one bit? Is this a Mealy or a Moore state machine?



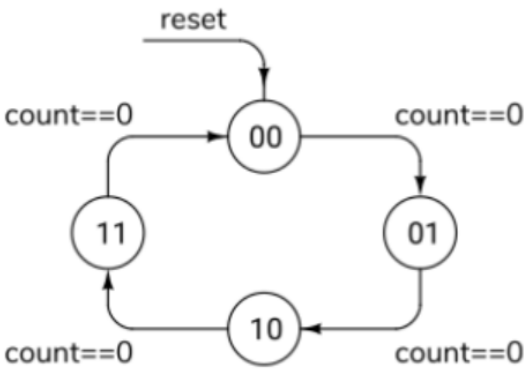
**Exercise 9**: Write **`always_ff`** statements that implement these state machines.

| count | in == out | next count |
|---|---|---|
| x | 1 | $N-1$ |
| 0 | x | $N-1$ |
| $n$ | 0 | $n-1$ |




```
logic [ : ] count;

always_ff @(posedge clk)
  count <=
    in == out ? N-1 :
    !count ? N-1 :
    in != out ? count-1:
    count;

always_ff @(posedge clk)
  out <=
    count != 0 ? out : in;
```

**Exercise 10**: Write the state transition table for this state machine.




| count | reset | next count |
|---|---|---|
| X | 1 | 00 |
| 00 | 0 | 01 |
| 01 | 0 | 10 |
| 10 | 0 | 11 |
| 11 | 0 | 00 |