Lecture 9

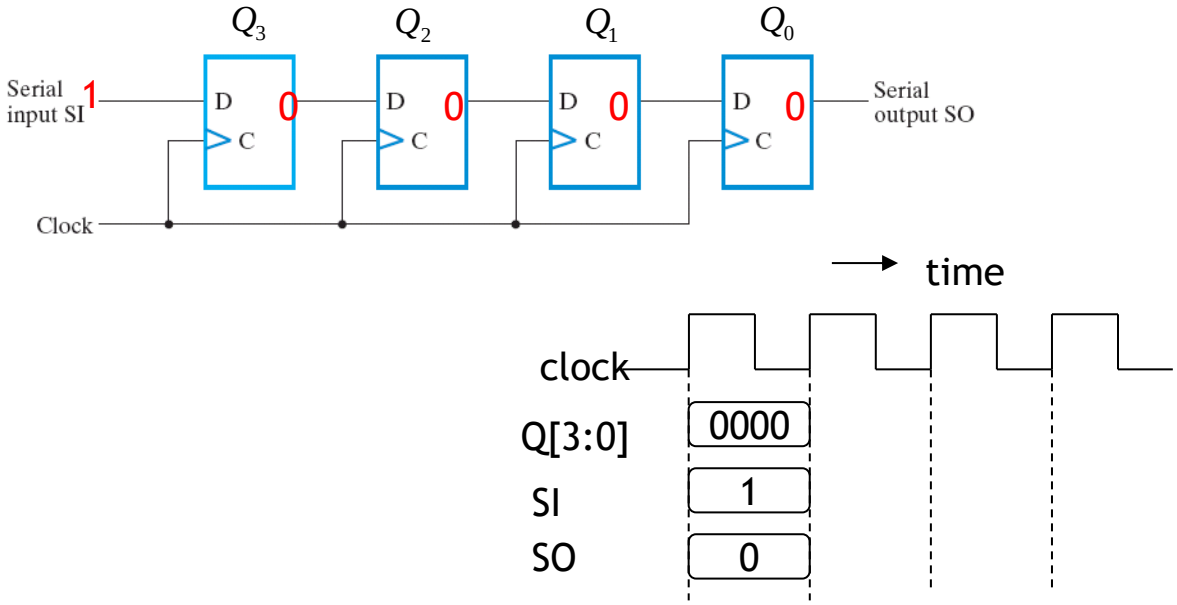
ECE 228

Mostafizur Rahman

UMKC

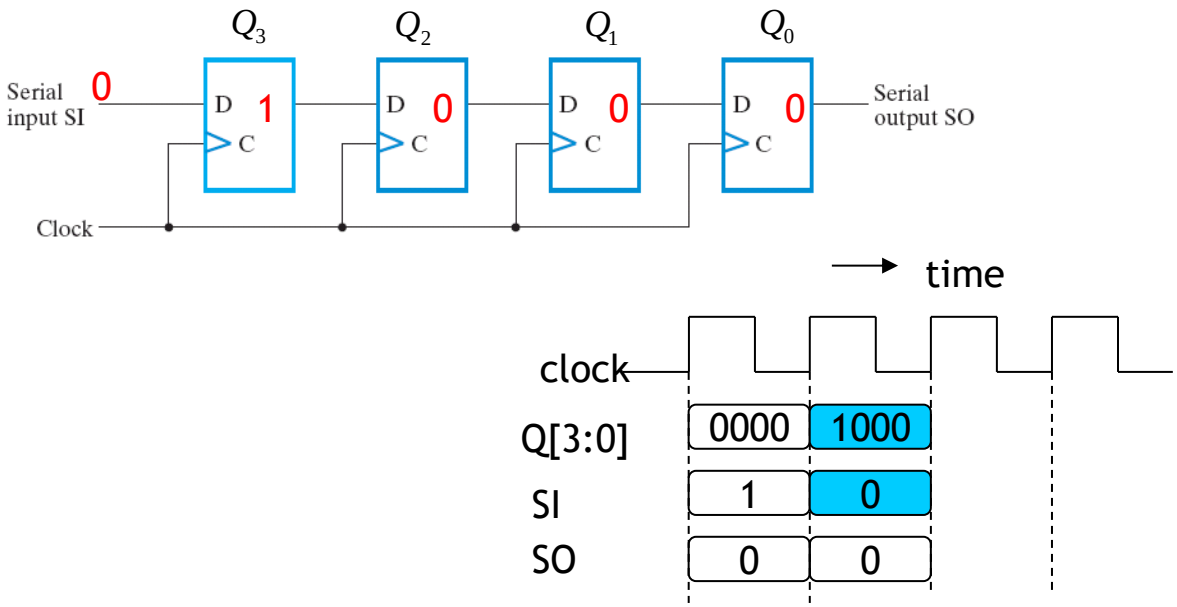
Function of the simple shift register

- initial state at cycle 0



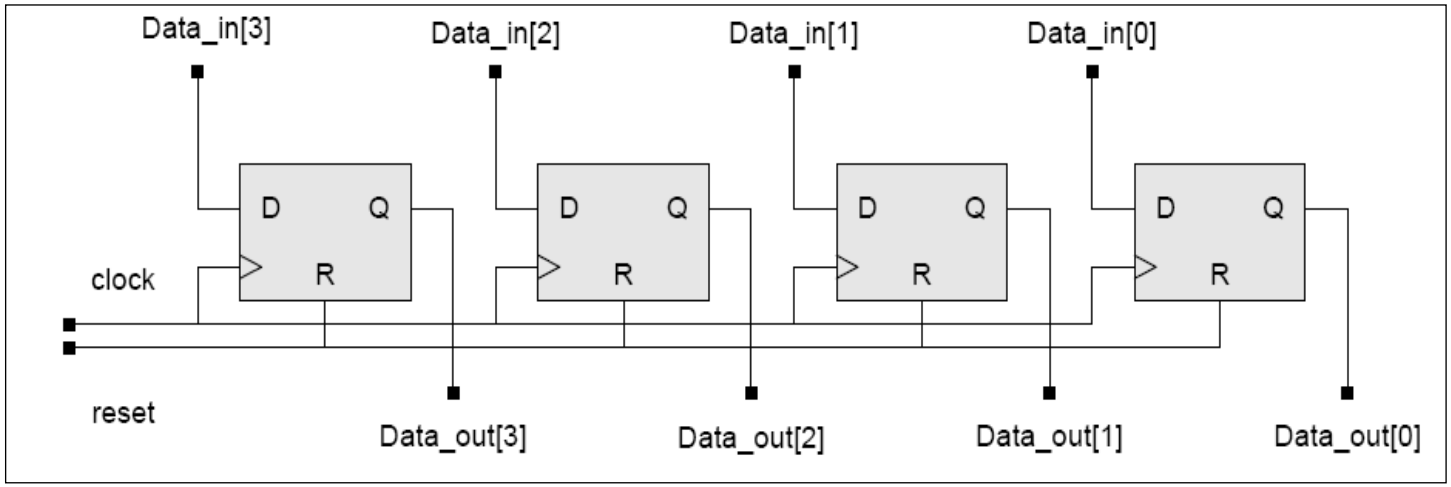
Function of the simple shift register

- initial state at cycle 1

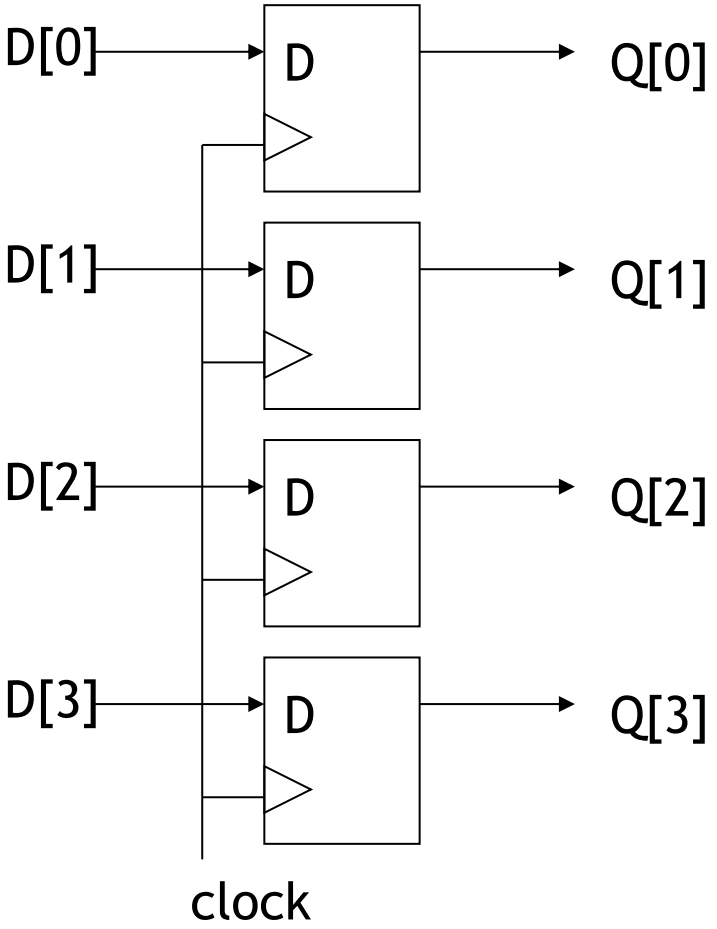


VERILOG: Synthesis - Sequential Logic

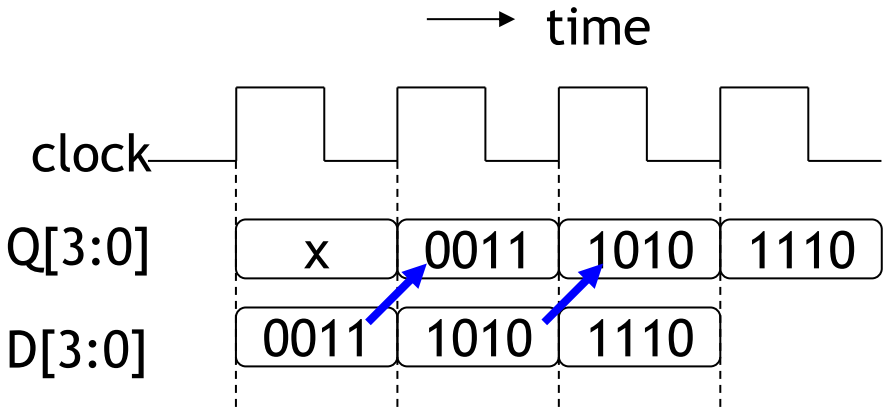
```
module D_reg4a (Data_in, clock, reset, Data_out);
  input [3:0] Data_in;
  input clock, reset;
  output [3:0] Data_out;
  reg [3:0] Data_out;
  always @ (posedge reset or posedge clock)
     if (reset == 1'b1) Data_out <= 4'b0;
    else Data_out <= Data_in;
endmodule
```



Timing behavior of the register



$$Q(t+1) = D(t)$$



Activity

Write a module for 4 bit register that has clk, clear, set and D as inputs. On every clock's negative edge, the output (Q) gets the value D. If *clear* signal is ON, the register is reset to all 0s, and if *set* is ON, register is set to all 1s.

Solution

```
module register (q,d,clk,clr_, set_) ;
output [7:0] q ;
input [7:0] d ;
input clk,clr_, set_ ;
reg [7:0] q ;
always @ (posedge clk or negedge clr_ or negedge set_)
   if (~clr_)
                q = 0 ;
     else if (~set_)
                q = 8'b1111_1111 ;
   else
                q = d ;
endmodule
```

Sequential Circuits

- How to design custom sequential circuits?

```
If(X)
        count = count +2
Else count = count +1
```

State Machine

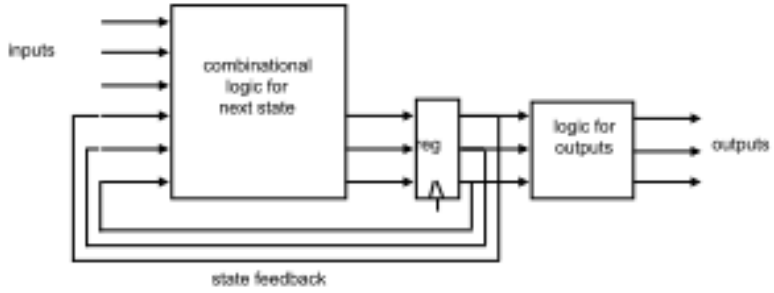
- We need register to hold
- the current state
  - always @(posedge clk) block

- We need next state function
  - Where do we go from each state given the inputs
  - state by state case analysis
    - next state determined by current state and inputs

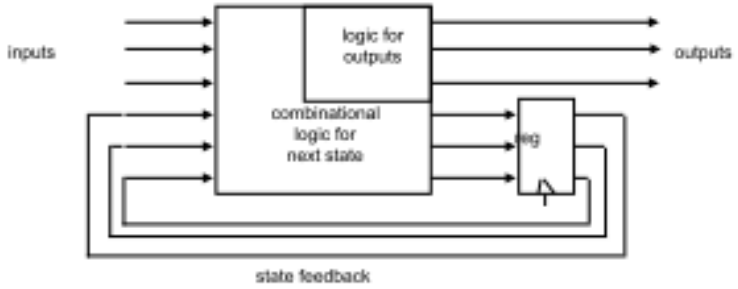
- We need the output function
  - State by state analysis
  - Moore: output determined by current state only
  - Mealy: output determined by current state and inputs

State Machine

- Moore

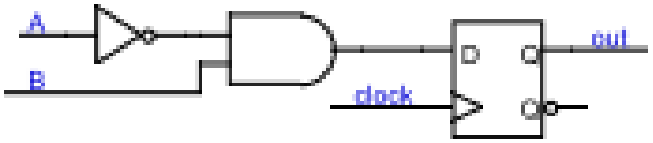


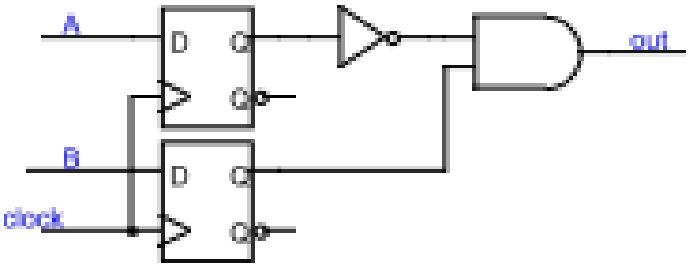
- Mealy



Activity

- Recognize A,B = 0,1
  - Mealy or Moore?





State Register

- Declare two values
  - state : current state – output of state register
  - nxtState : next state – input to state register
  - We rely on next state function to give us nxtState
- Declare symbols for states with state assignment

```
localparam IDLE=0, WAITFORB=1,
           DONE=2, ERROR=3;

reg [1:0] state,    // Current state
          nxtState; // Next state
```

State Machine- Verilog

- Simple code for register
  - Define reset state
  - Otherwise, just move to nxtState on clock edge

```
localparam IDLE=0, WAITFORB=1,
           DONE=2, ERROR=3;
reg [1:0] state,    // Current state
          nxtState; // Next state

always @(posedge clk) begin
  if (reset) begin
    state <= IDLE;   // Initial state
  end else begin
    state <= nxtState;
  end
end
```

State Machine-Verilog

- Combinational logic function
  - Inputs : state, inputs
  - Output : nxtState
- We could use assign statements
- We will use an `always @(*)` block instead
  - Allows us to use more complex statements
  - `if`
  - `case`

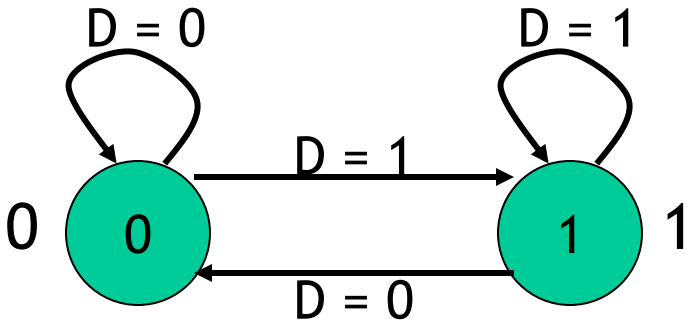
State Machine-Verilog

- Describe what happens in each state
- Case statement is natural for this

```
always @(*) begin
  nxtState = state;  // Default next state: don't move
  case (state)
    IDLE : begin
      if (B) nxtState = ERROR;
      else if (A) nxtState = WAITFORB;
    end
    WAITFORB : begin
      if (B) nxtState = DONE;
    end
    DONE : begin
    end
    ERROR : begin
    end
  endcase
end
```

State Diagrams

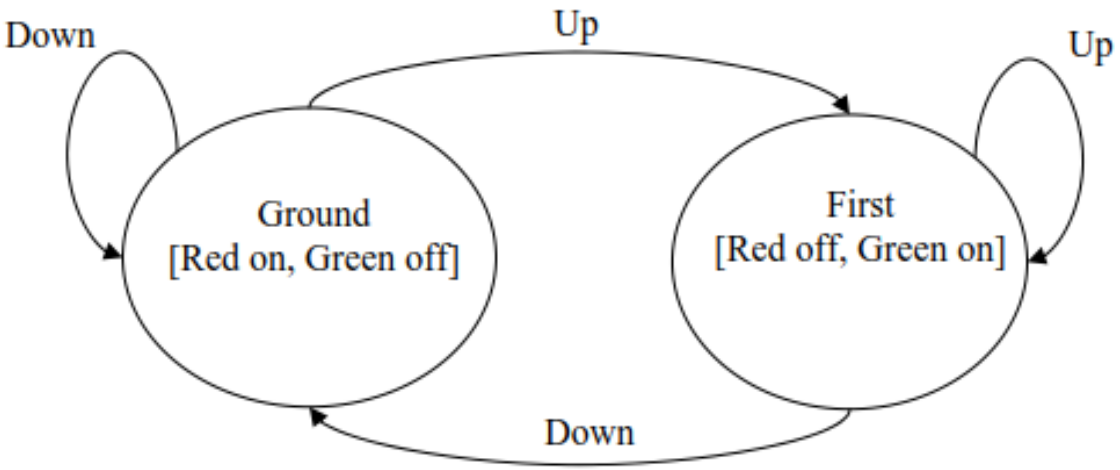
- Each state is shown with a circle, labeled with the state value – the contents of the circle are the outputs
- An arc represents a transition to a different state, with the inputs indicated on the label

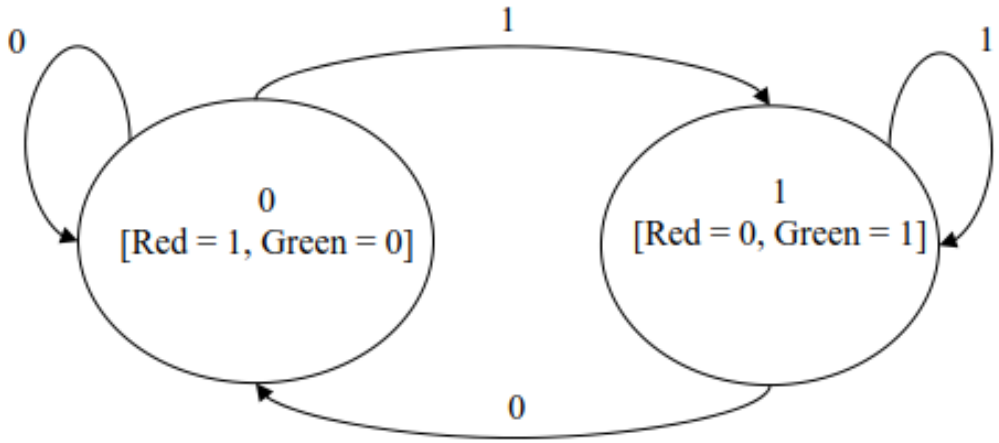


Example

In this example, we'll be designing a controller for an elevator. The elevator can be at one of two floors: Ground or First. There is one button that controls the elevator, and it has two values: Up or Down. Also, there are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First. At each time step, the controller checks the current floor and current input, changes floors and lights in the obvious way

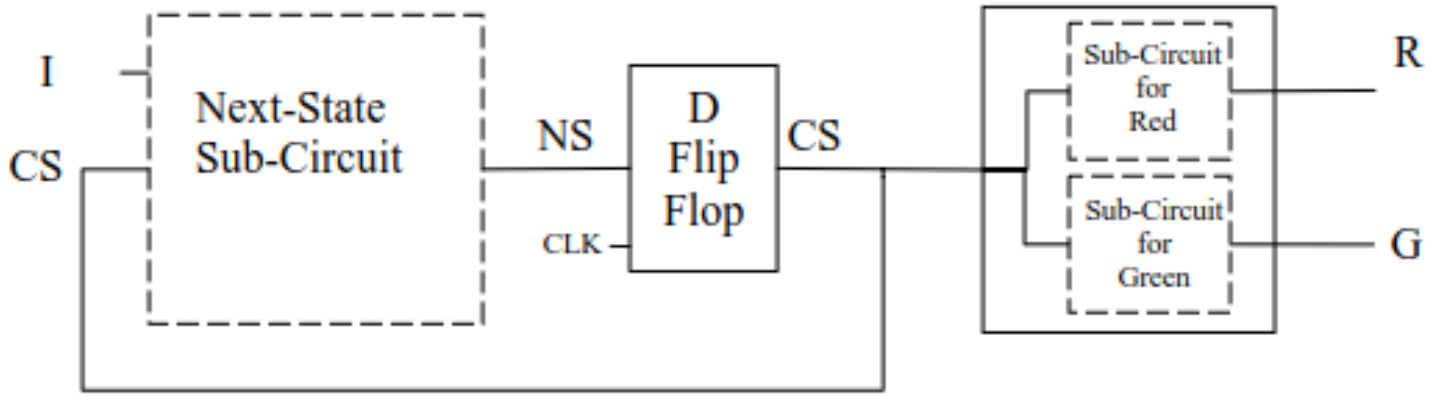
Example


Down
Up
Up
Ground
[Red on, Green off]
First
[Red off, Green on]
Down

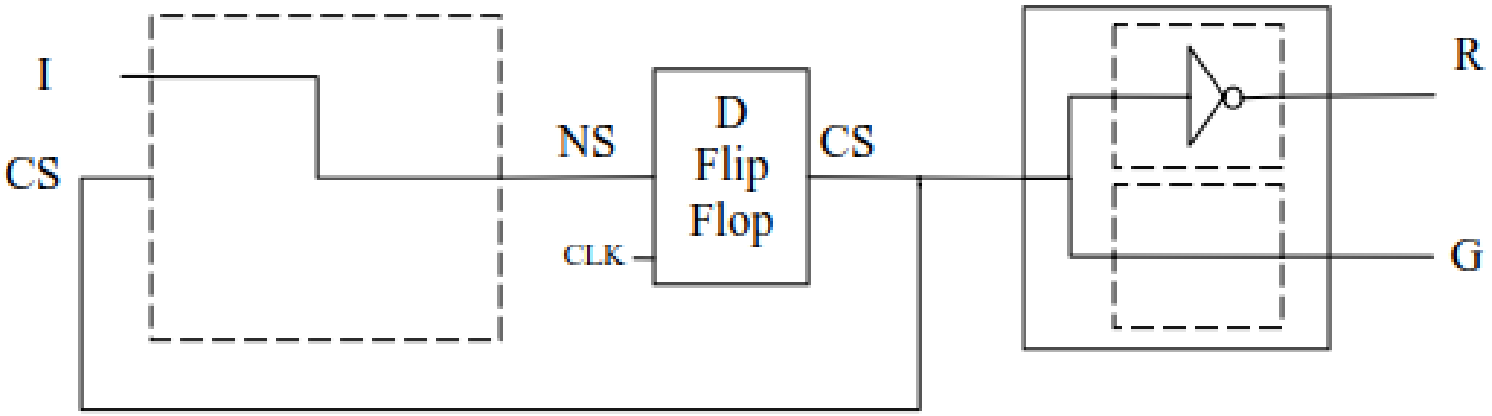

0
1
1
0
[Red = 1, Green = 0]
1
[Red = 0, Green = 1]
0

Example

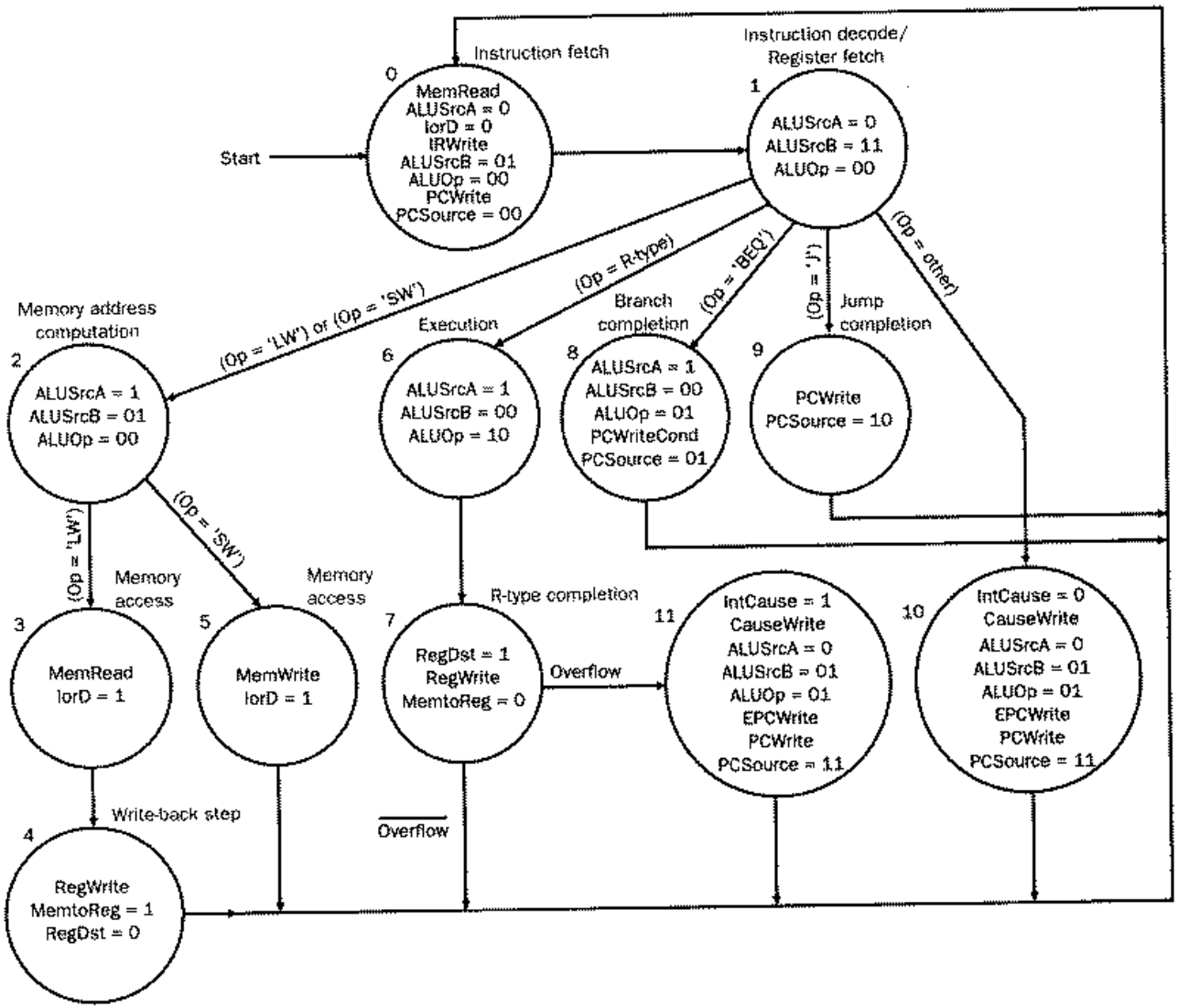
CurrentState	Input	NextState	Red	Green
0	0	0	1	0
0	1	1	1	0
1	0	0	0	1
1	1	1	0	1



Example

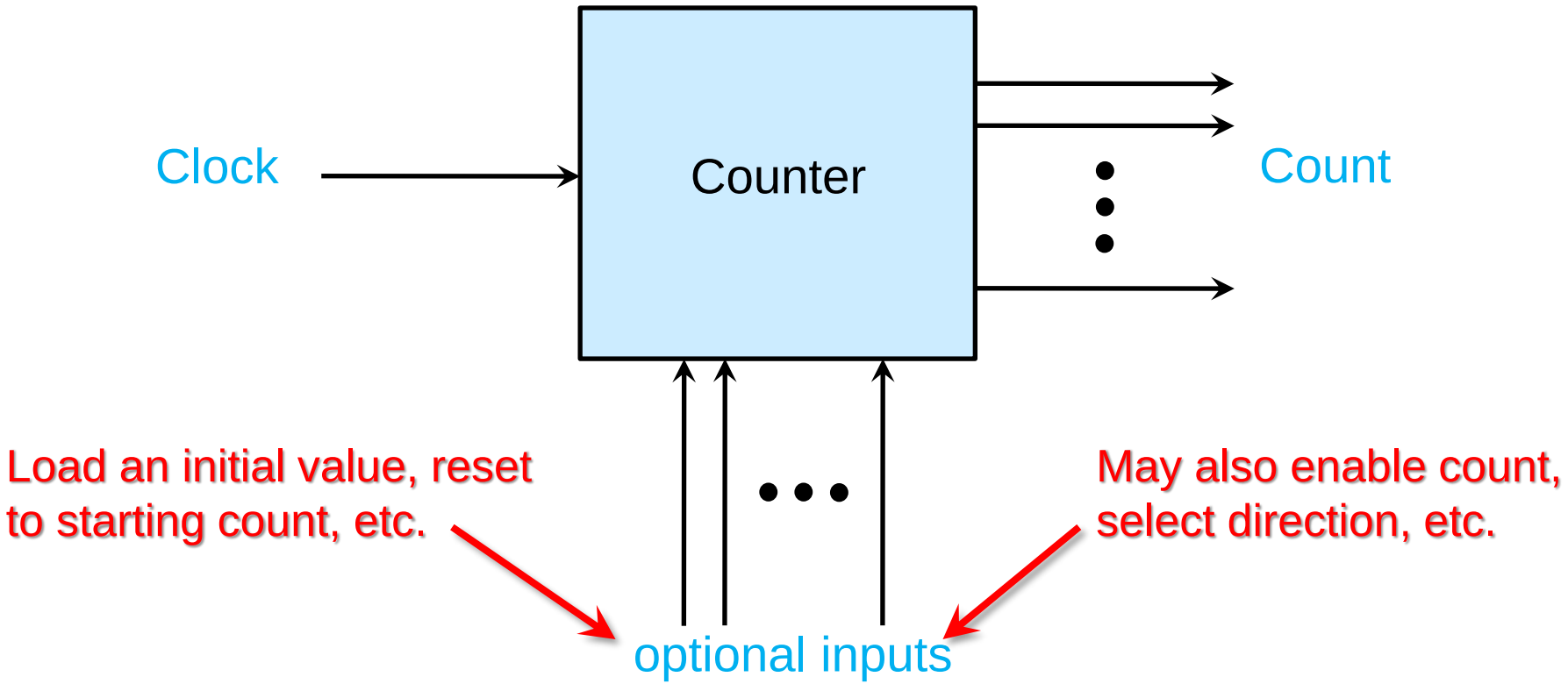


Simple Processor's State Machine



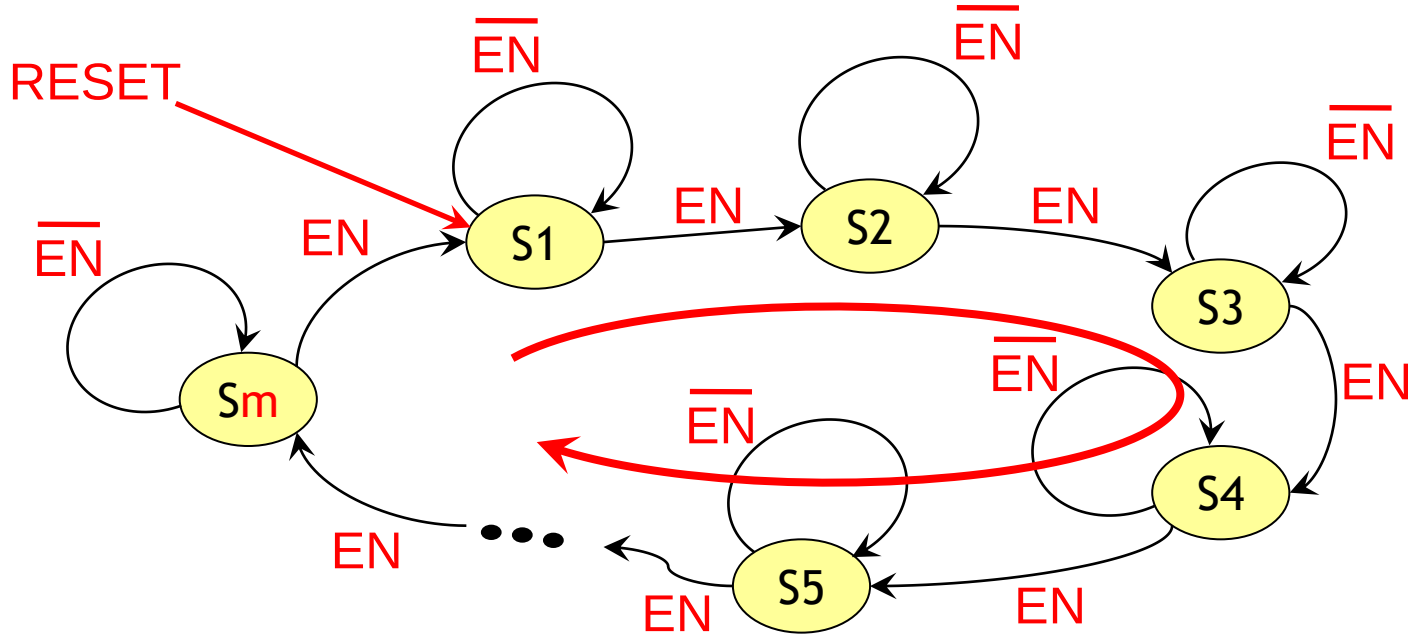
Counters

- A counter is a circuit that produces a numeric count each time an input clock pulse makes an active transition



Counter

- From another viewpoint, a counter is any sequential circuit whose state diagram is a single cycle
  - in other words, counters are a special case of a finite state machine
- Output is usually the state value, Moore machine



Counters

- Counters differ by a number of basic characteristics, including:

Characteristic	Description
Modulus	Length of sequence
Coding	Count sequence
Direction	Up or down
Resetable	Reset to zero
Loadable	Load a specific value

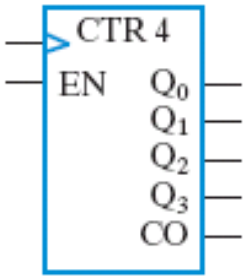
Two Common Types of Counters

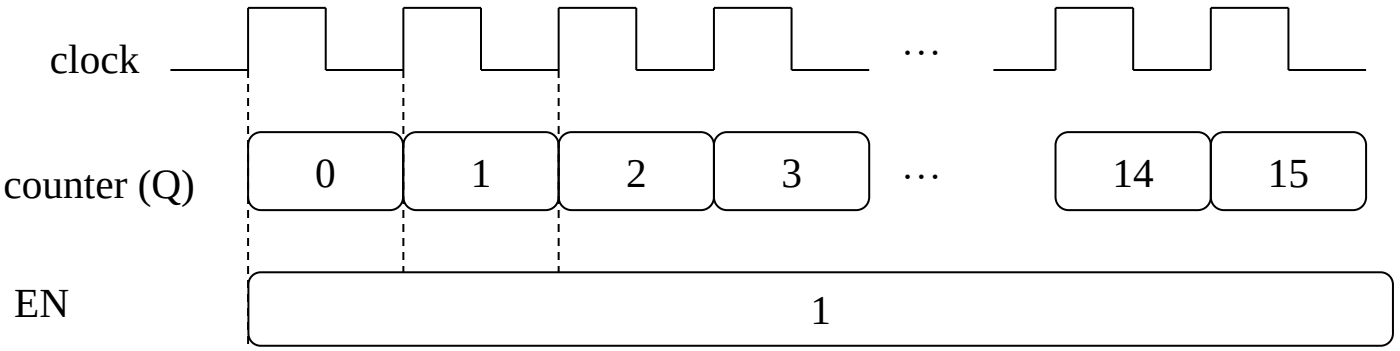
- Synchronous Counters
  - Clock is directly connected to the flip-flop clock inputs
  - Logic is used to implement the desired state sequencing

- Asynchronous Counters
  - This counter is called *asynchronous* because not all flip flops are hooked to the same clock.
  - Clock connected to the flip-flop clock input on the LSB bit flip-flop
  - For all other bits, a flip-flop output is connected to the clock input, thus circuit is not truly synchronous!
  - slow, limited by propagation delays

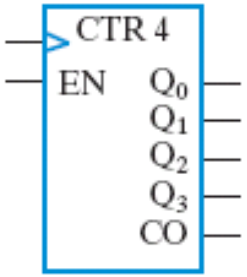
Example – Synchronous binary counter

- Function of the counter



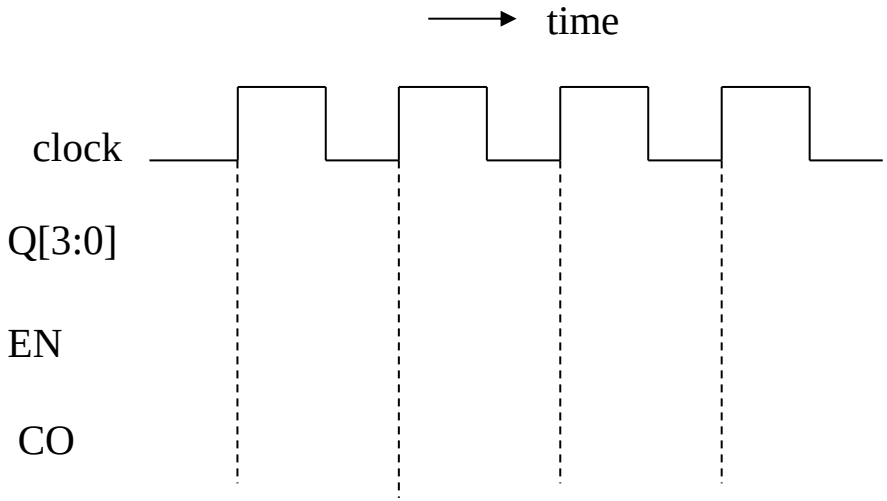


Example - Synchronous binary counter

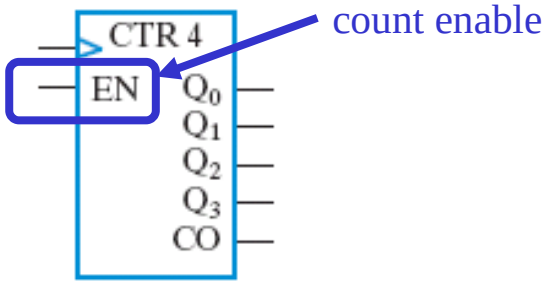


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

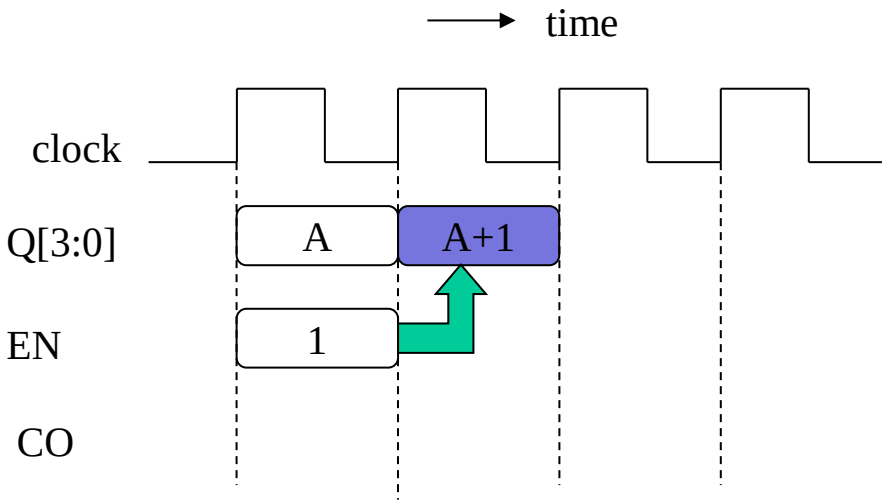


Example - Synchronous binary counter

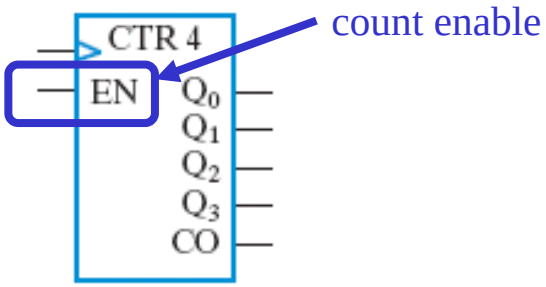


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

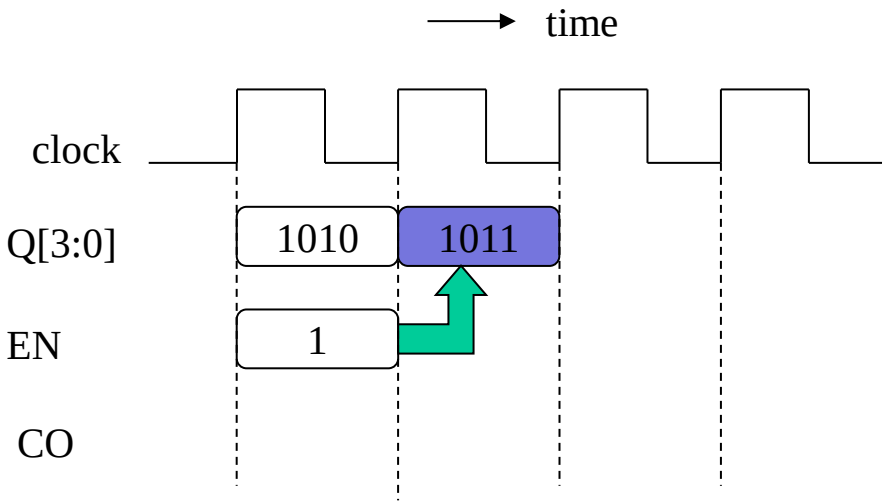


Example - Synchronous binary counter

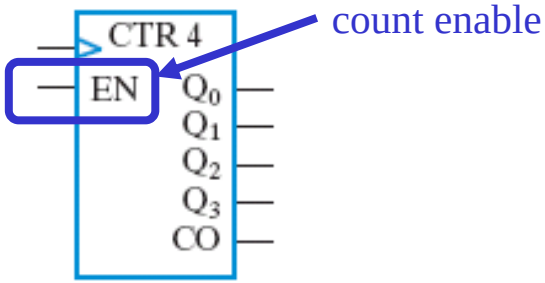


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

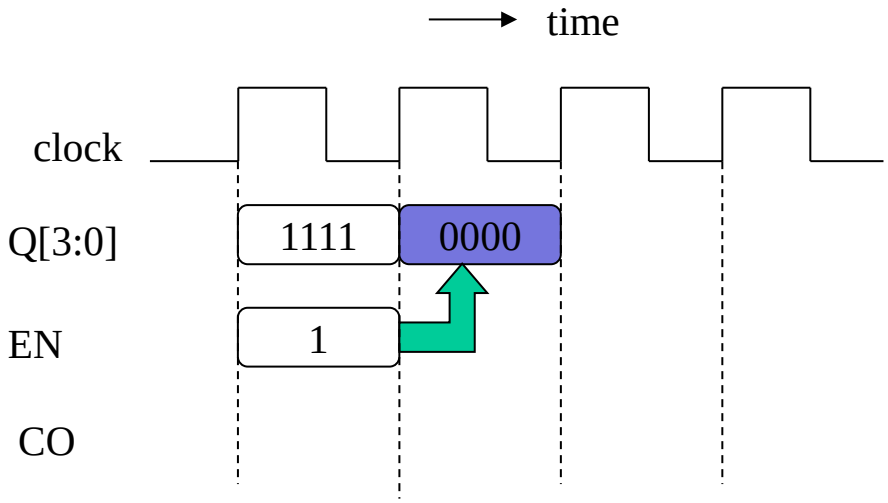


Example - Synchronous binary counter

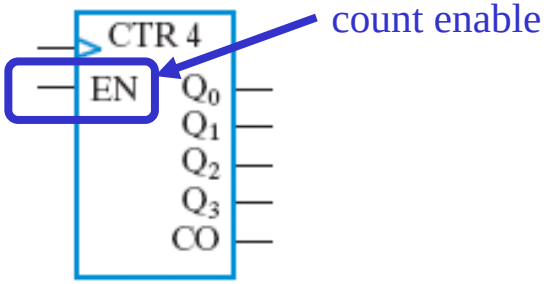


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

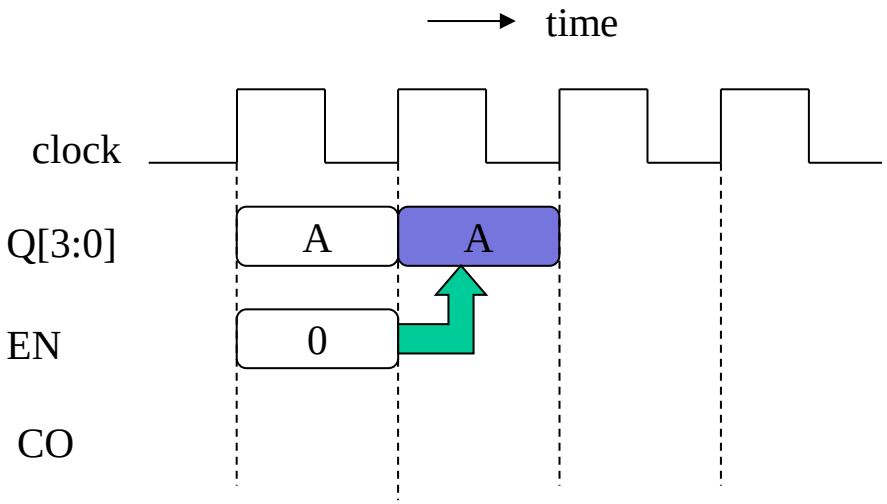


Example - Synchronous binary counter

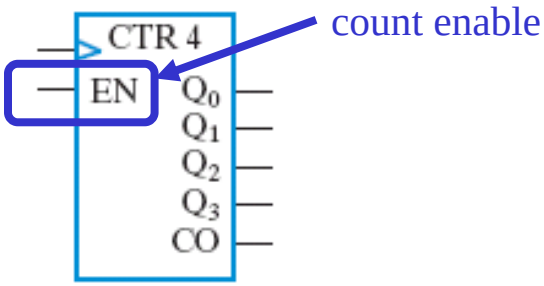


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

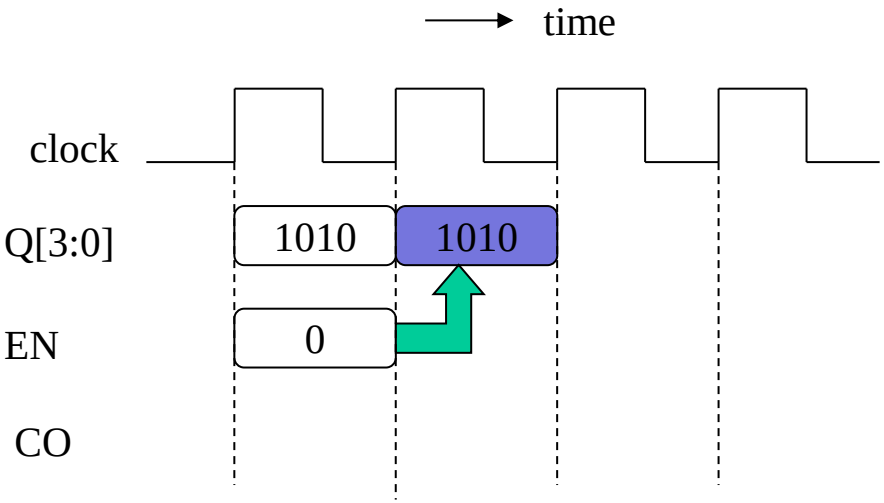


Example - Synchronous binary counter

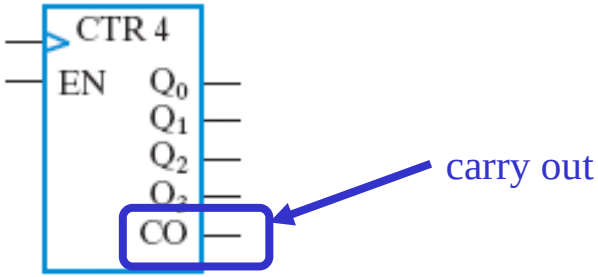


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

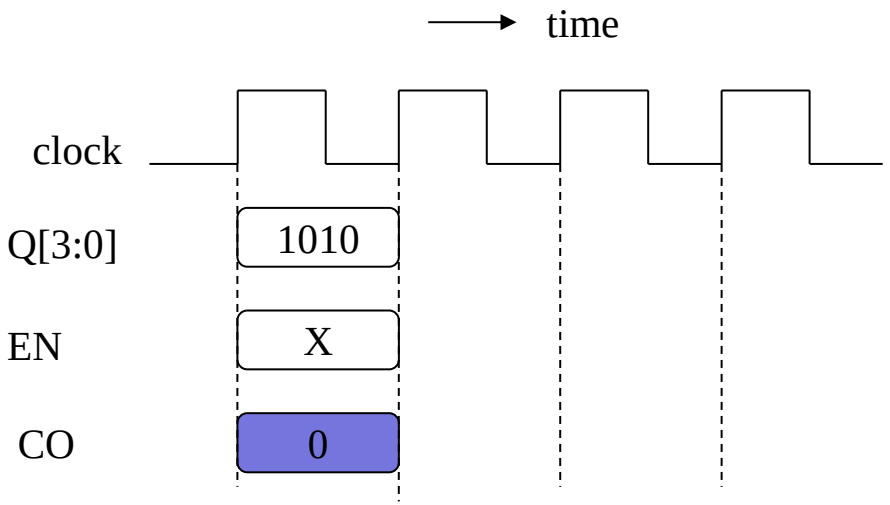


The Spec

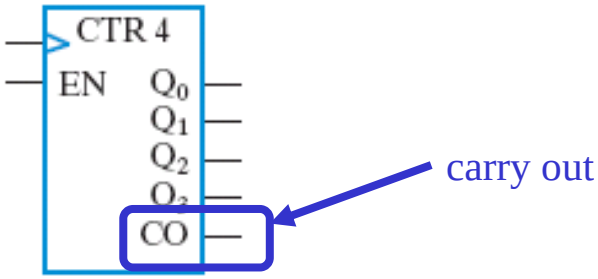


$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$

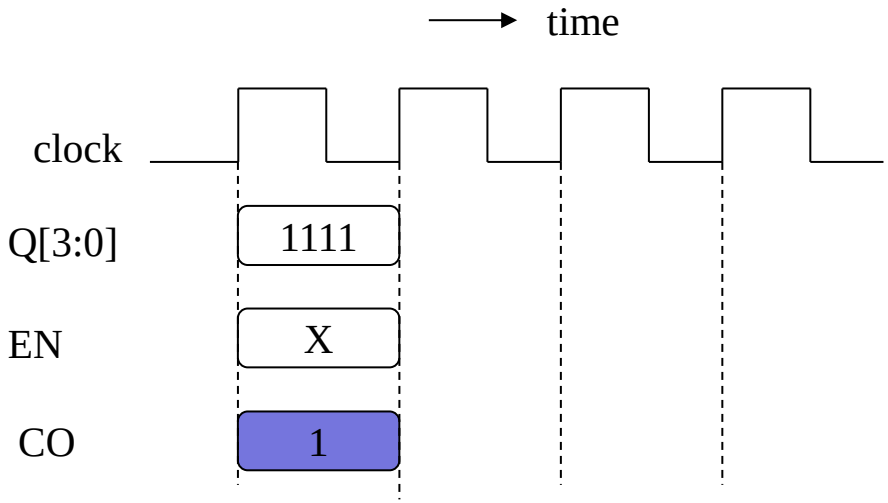


Example - Synchronous binary counter



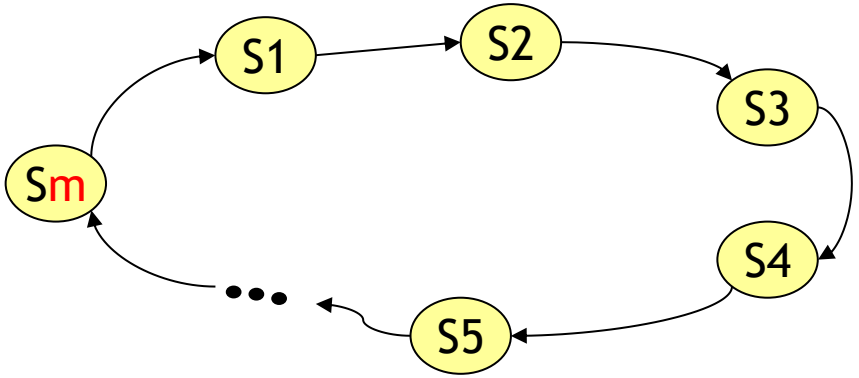
$$Q(t+1) = \begin{cases} Q(t)+1 \text{ if } EN(t)=1 \\ Q(t) \text{ if } EN(t)=0 \end{cases}$$

$$CO(t) = \begin{cases} 1 \text{ if } Q(t) = (1111)_2 \\ 0 \text{ if } Q(t) \neq (1111)_2 \end{cases}$$



Modulus Counters

- Modulus
  - number of states in a counter's cycle
- Given m states
  - modulo-m counter or divide-by-m counter
  - Count is remainder of division by *m*; *m* may not be a power of 2
- Power-of-2 counters use all states
- Non-power-of-2 counters have extra, unused states



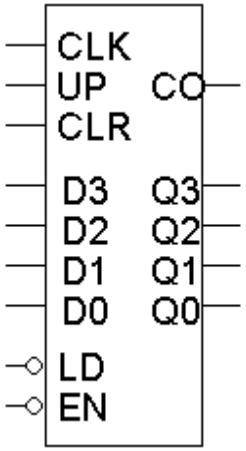
- Example: 4-bit Binary / Hex / Mod-16 Counter

  0000, 0001, 0010, … 1110, 1111, 0000, 0001, …
  - All states used
- 4-bit BCD / Decade / Mod-10 Counter

  0000, 0001, 0010, … 1000, 1001, 0000, 0001, …
  - 6 unused states

More complex counters

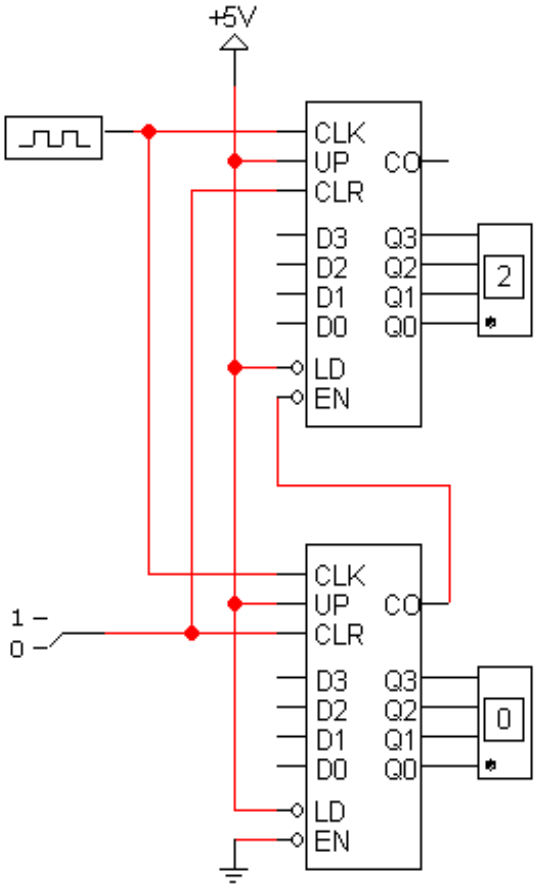
- An example of full-featured 4-Bit Synchronous counter:
  - It can increment or decrement, by setting the UP input to 1 or 0.
  - You can immediately (asynchronously) clear the counter to 0000 by setting CLR = 1.
  - You can specify the counter's next output by setting D_3-D_0 to any four-bit value and clearing LD.
  - The active-low EN input enables or disables the counter.
    - When the counter is disabled, it continues to output the same value without incrementing, decrementing, loading, or clearing.
  - The "counter out" CO is normally 1, but becomes 0 when the counter reaches its maximum value, 1111.



An 8-bit counter

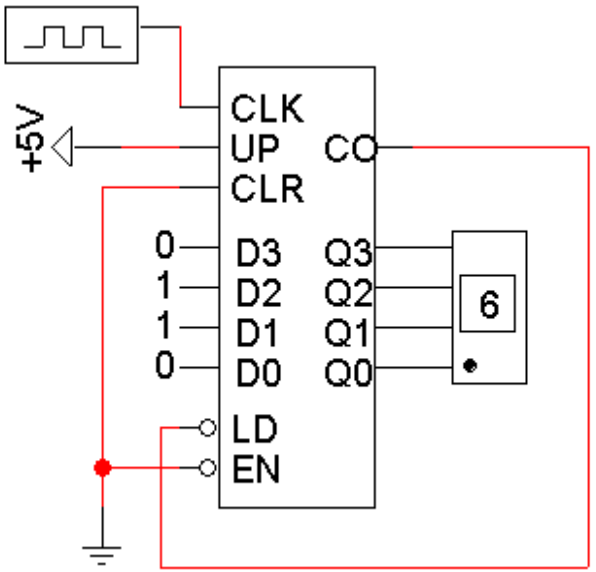
- As you might expect by now, we can use these general counters to build other counters.

- Here is an 8-bit counter made from two 4-bit counters.
  - The bottom counter represents the least significant four bits, while the top counter represents the most significant four bits.
  - When the bottom counter reaches 1111 (i.e., when CO = 0), it enables the top counter for one cycle.
  - The counters share clock and clear signals.



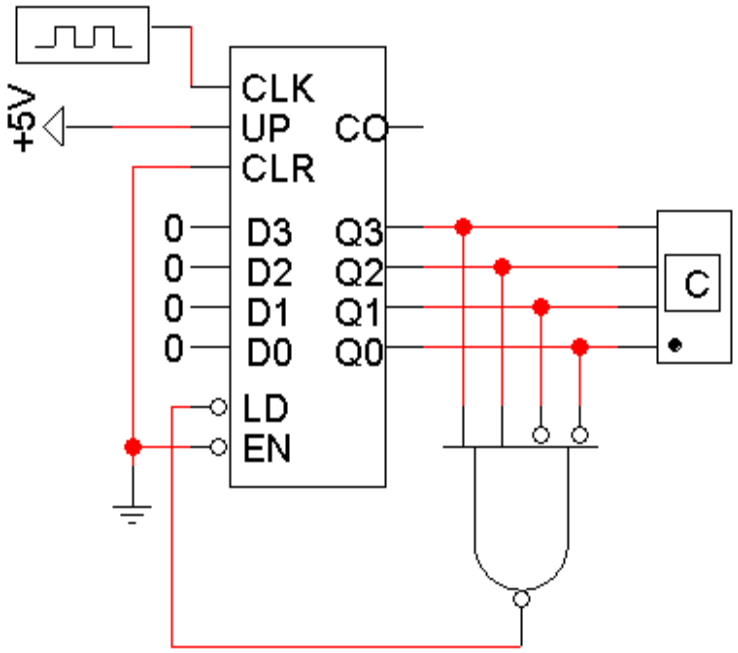
A restricted 4-bit counter

- We can also make a counter that “starts” at some value besides 0000.
- In the diagram below, when CO=0 the LD signal forces the next state to be loaded from D_3-D_0.
- The result is this counter wraps from 1111 to 0110 (instead of 0000).

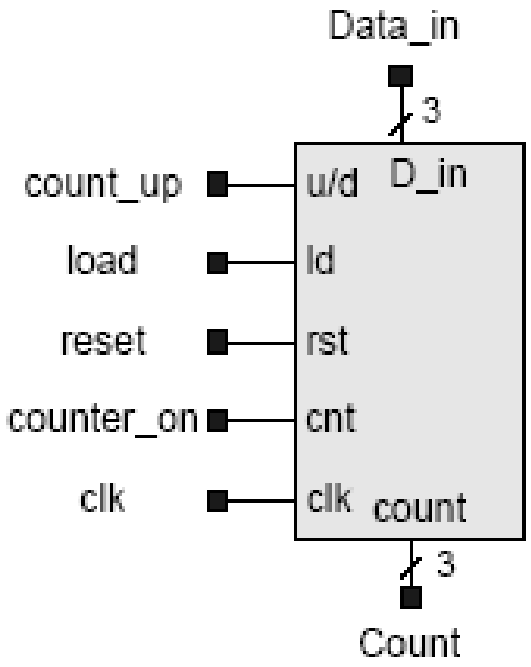


Another restricted counter

- We can also make a circuit that counts up to only 1100, instead of 1111.
- Here, when the counter value reaches 1100, the NAND gate forces the counter to load, so the next state becomes 0000.



3-bit Verilog Up/Down Counter w/ parallel load

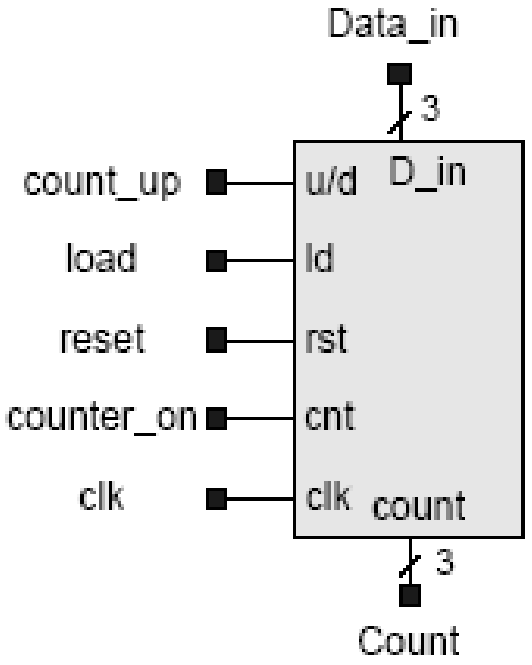


Functional Specs.

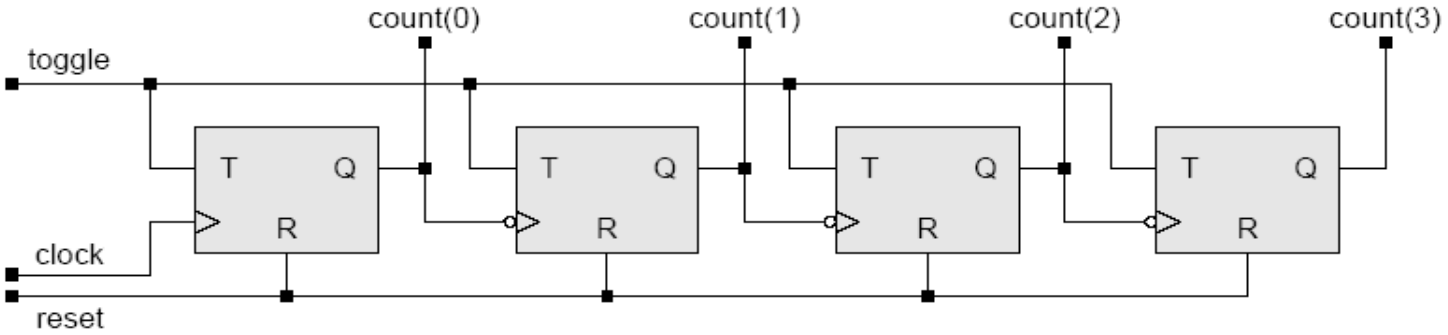
- Load counter with Data_in when load = 1
- Counter counts when counter_on = 1
  - counts-up when count_up = 1
  - Counts-down when count_up = 0

Verilog Up/Down Counter (cont.)

```
module up_down_counter (clk, reset, load, count_up, counter_on, Data_in, Count);
  input clk, reset, load, count_up, counter_on;
  input [2:0] Data_in;
  output [2:0] Count;
  reg [2:0] Count;

  always @ (posedge reset or posedge clk)
    if (reset == 1'b1)
       Count = 3'b0;
    else if (load == 1'b1)
       Count = Data_in;
    else if (counter_on == 1'b1) begin
        if (count_up == 1'b1)
           Count = Count +1;
        else Count = Count -1;
    end
endmodule
```



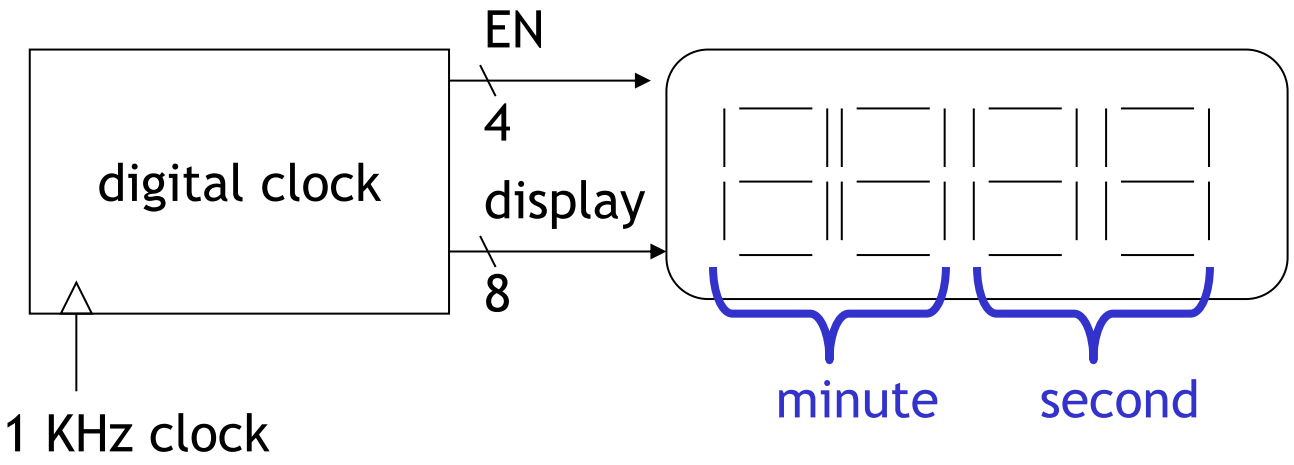
Verilog 4-bit Ripple Counter



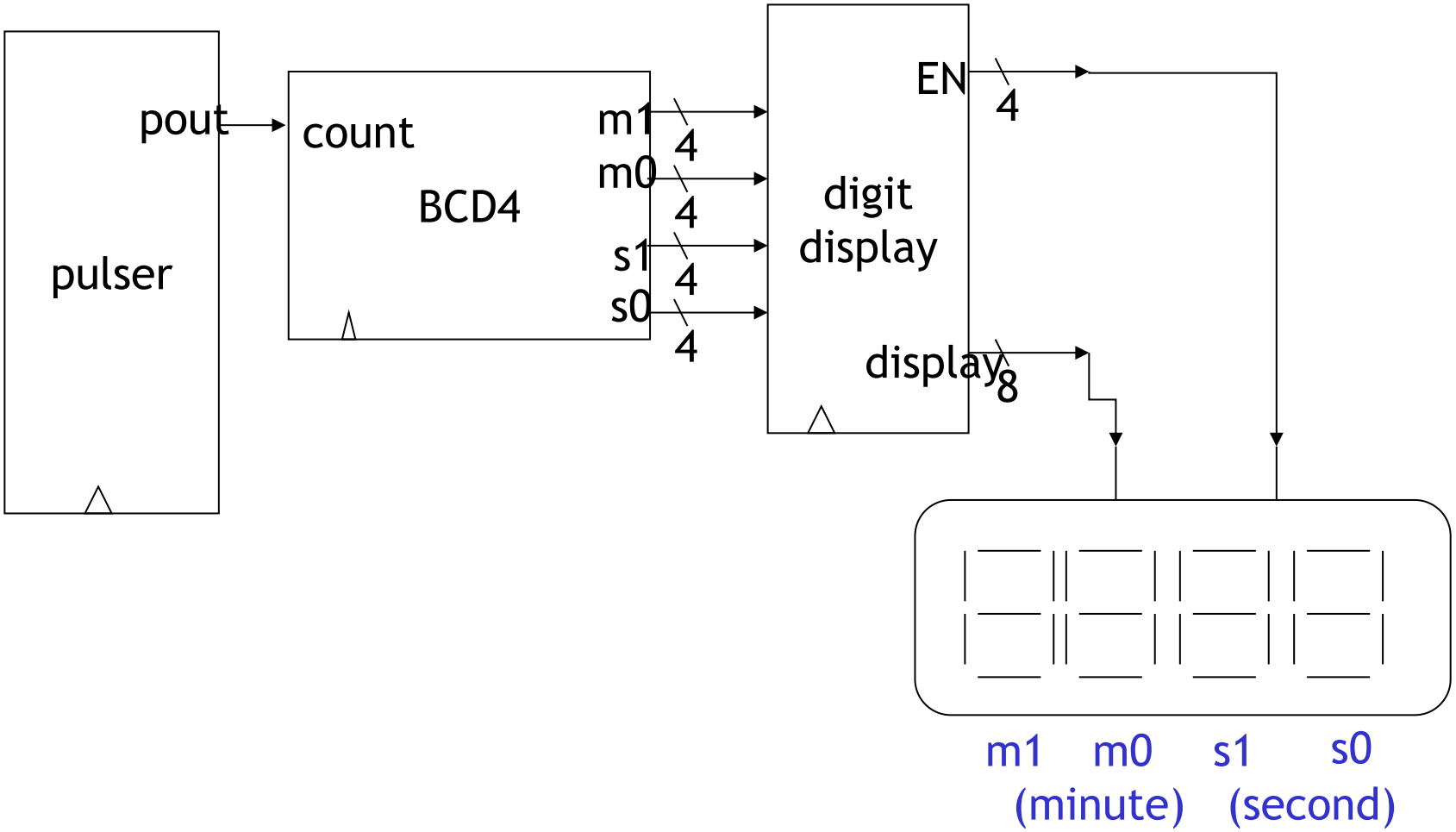

```
module ripple_counter (clock, toggle, reset, count);
  input clock, toggle, reset;
  output [3:0] count;
  reg [3:0] count;
  wire c0, c1, c2;
  assign c0 = count[0], c1 = count[1], c2 = count[2];

   always @ (posedge reset or posedge clock)
     if (reset == 1'b1) count[0] <= 1'b0;
    else if (toggle == 1'b1) count[0] <= ~count[0];

  always @ (posedge reset or negedge c0)
     if (reset == 1'b1) count[1] <= 1'b0;
     else if (toggle == 1'b1) count[1] <= ~count[1];

  always @ (posedge reset or negedge c1)
     if (reset == 1'b1) count[2] <= 1'b0;
     else if (toggle == 1'b1) count[2] <= ~count[2];

  always @ (posedge reset or negedge c2)
     if (reset == 1'b1) count[3] <= 1'b0;
     else if (toggle == 1'b1) count[3] <= ~count[3];
endmodule
```

Digital clock

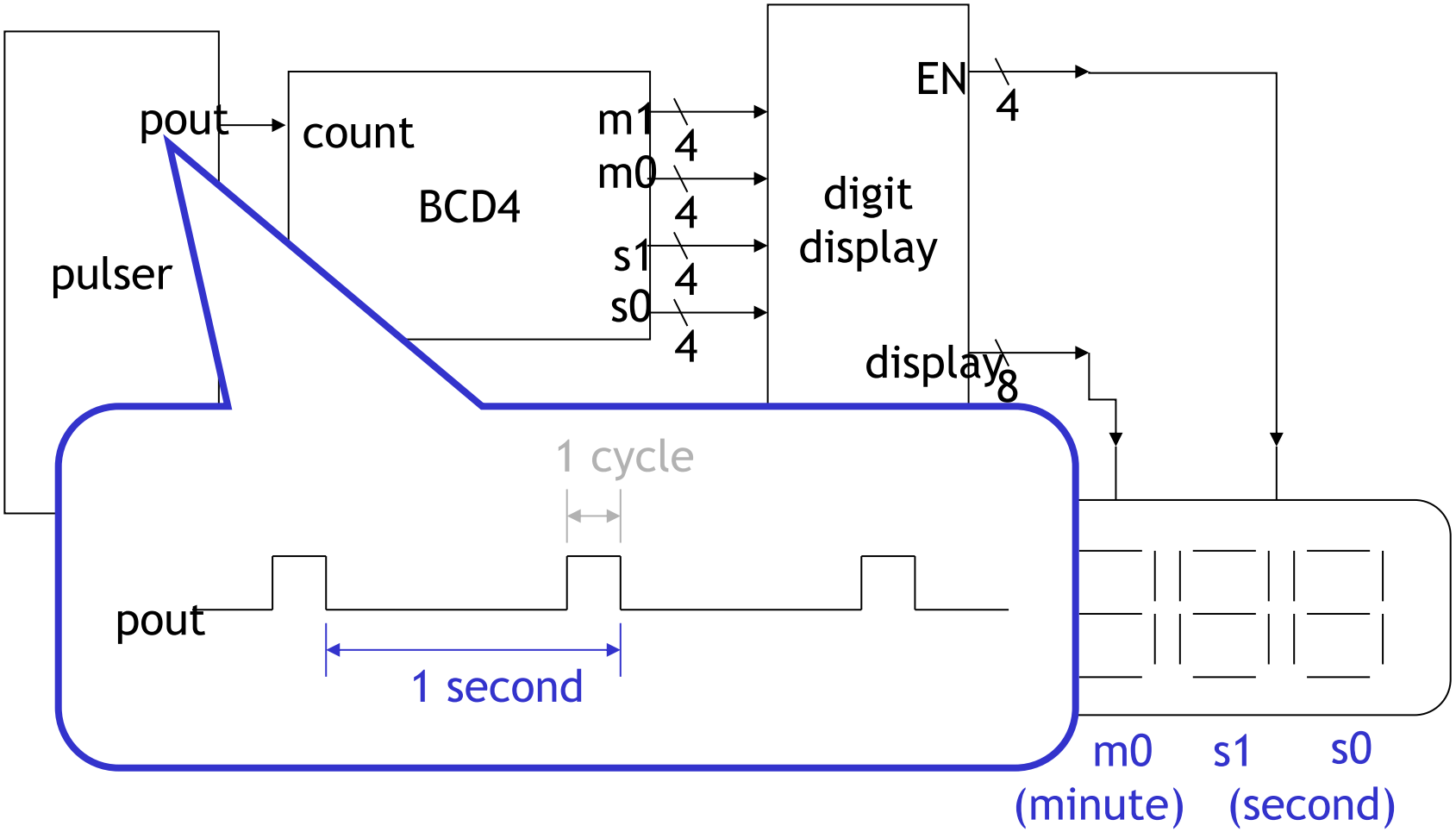
- refresh the time every one second
- 00:59 -> 01:00



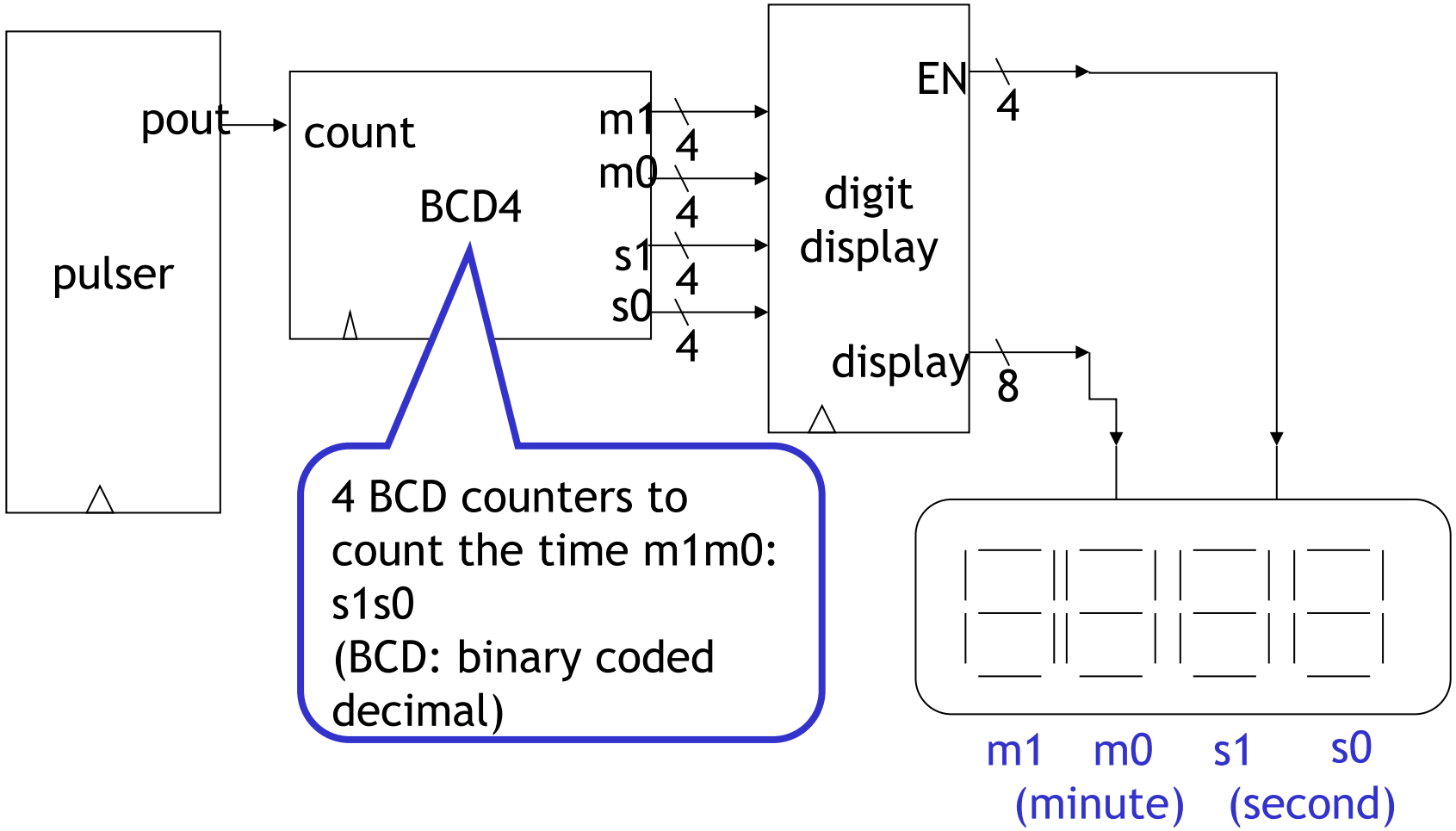
Architecture



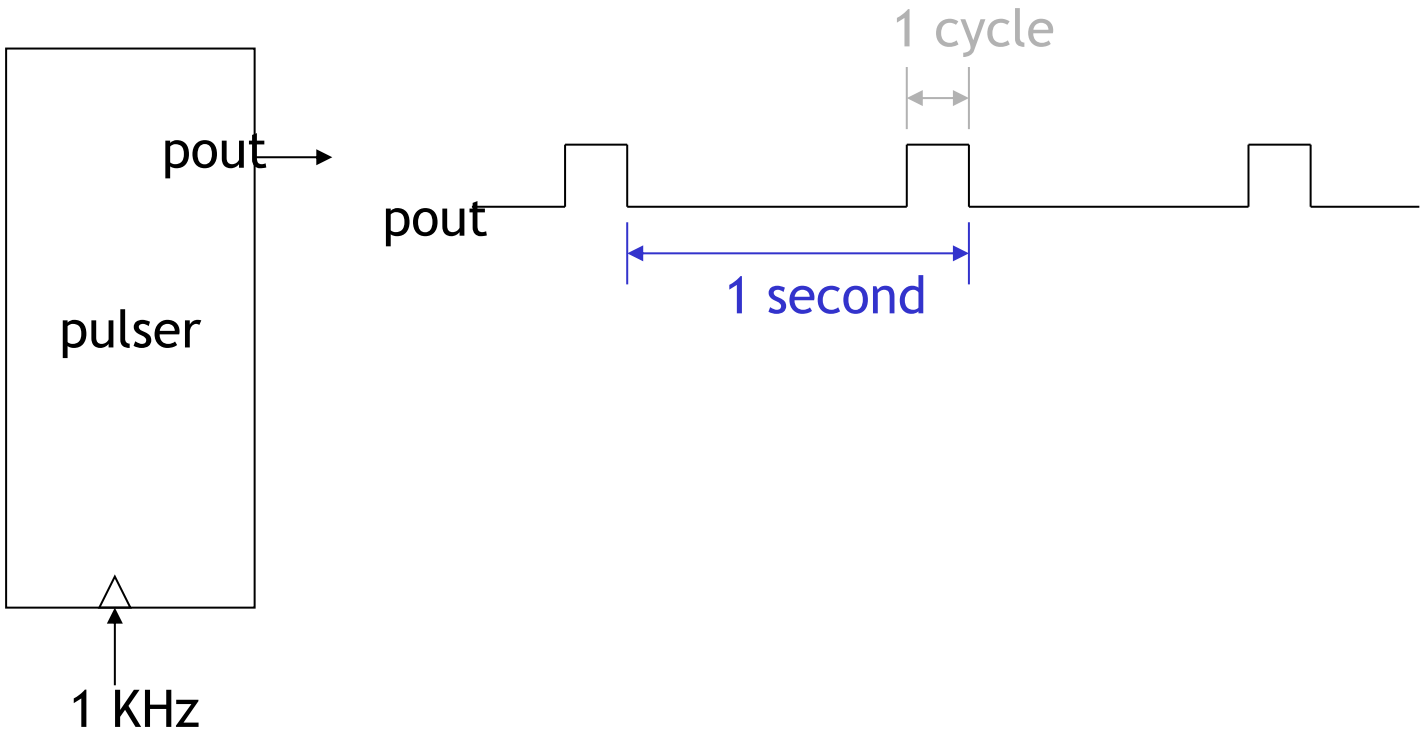
Architecture



Architecture

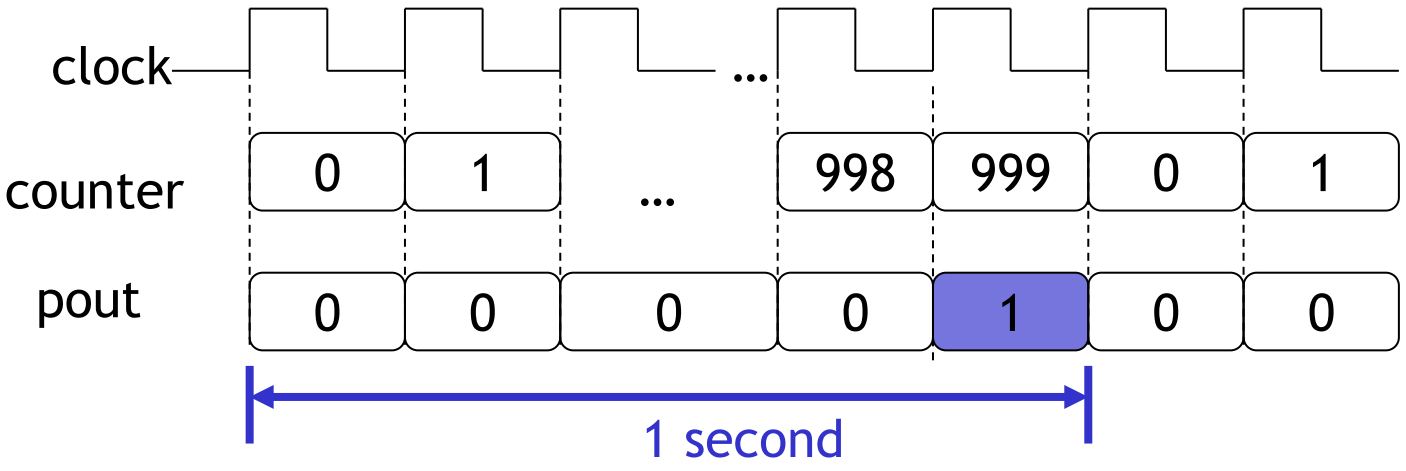


The pulser

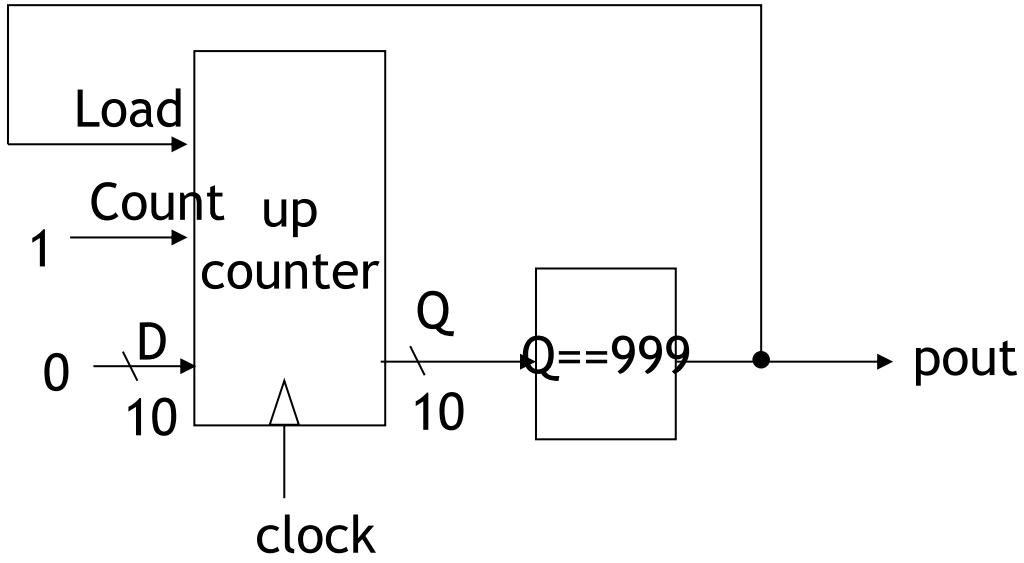


Design the pulser

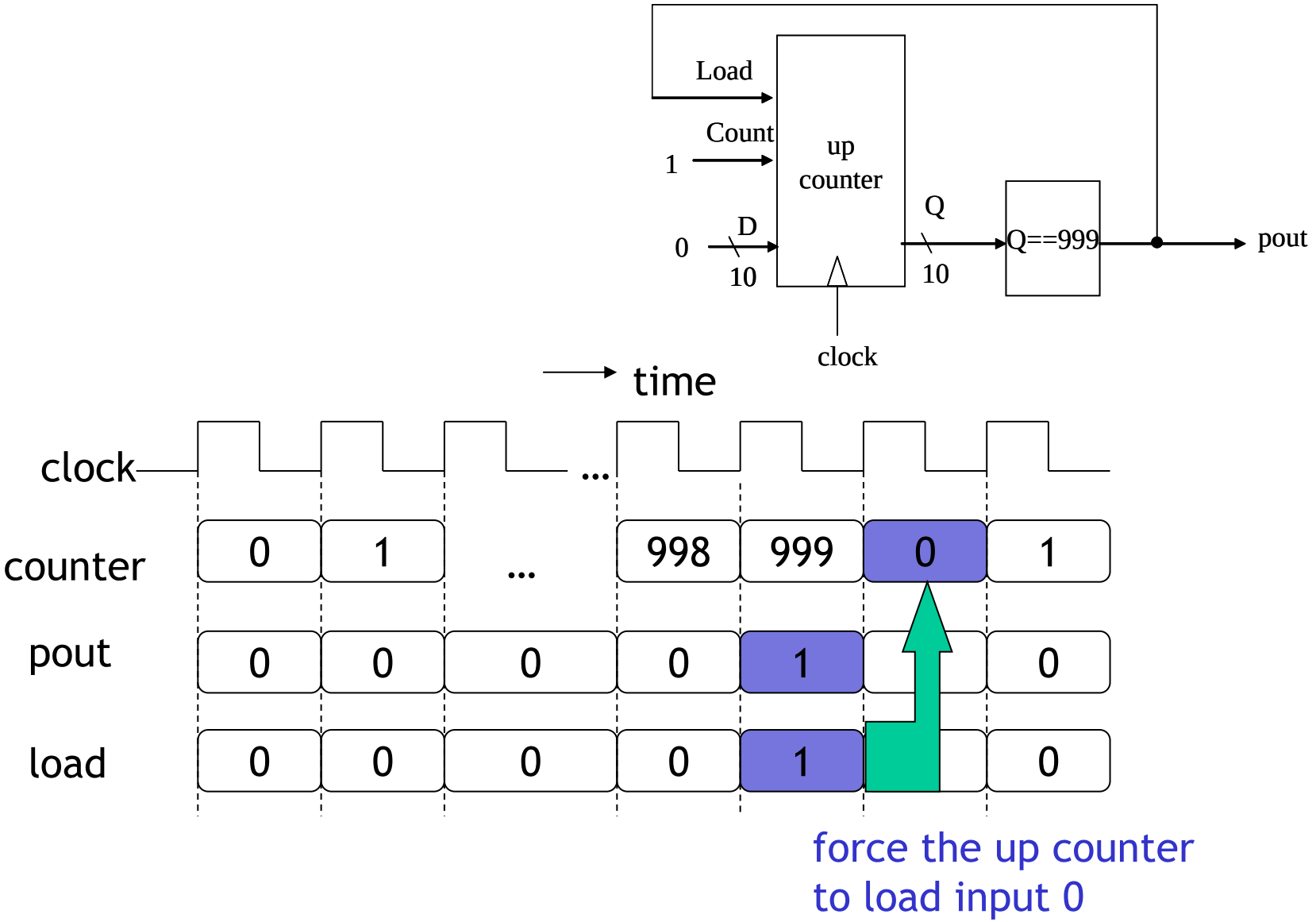
- up-counter within the pulser
- send out a pulse when counter==999



Pulser design

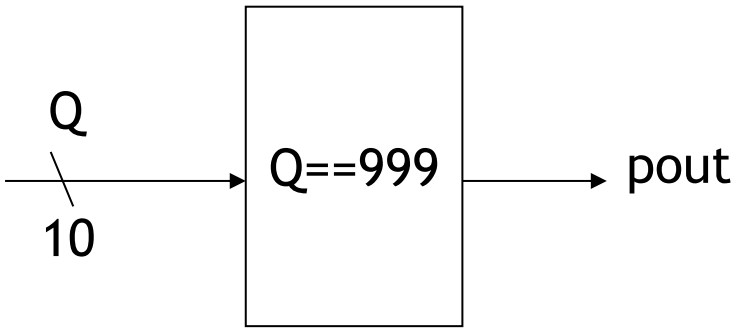


How the pulser works

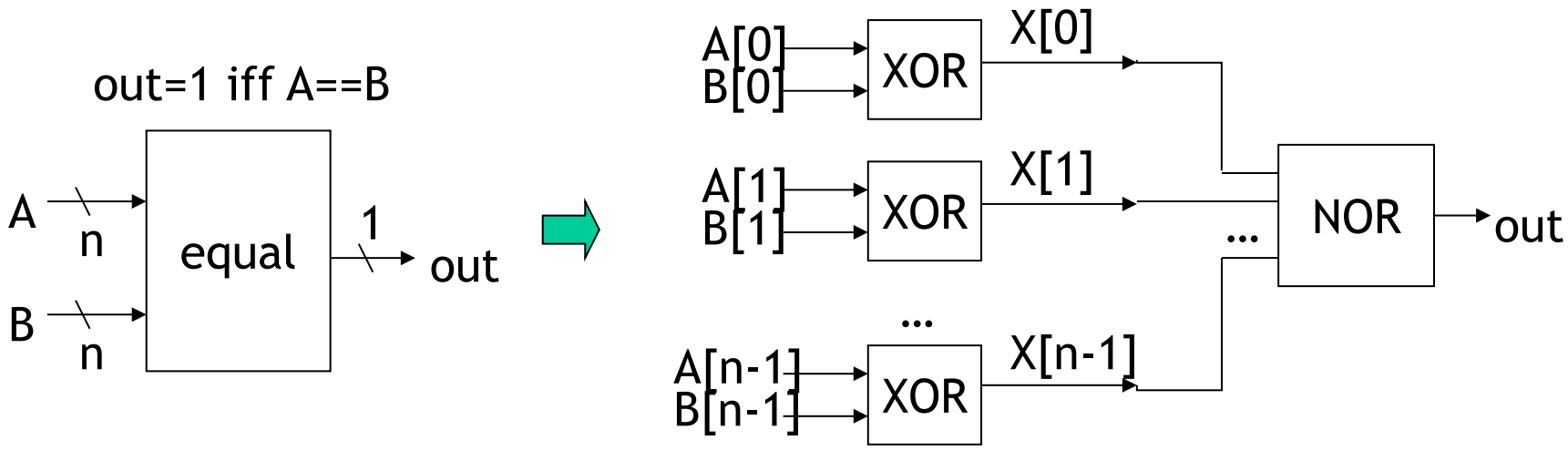


How to check equal?

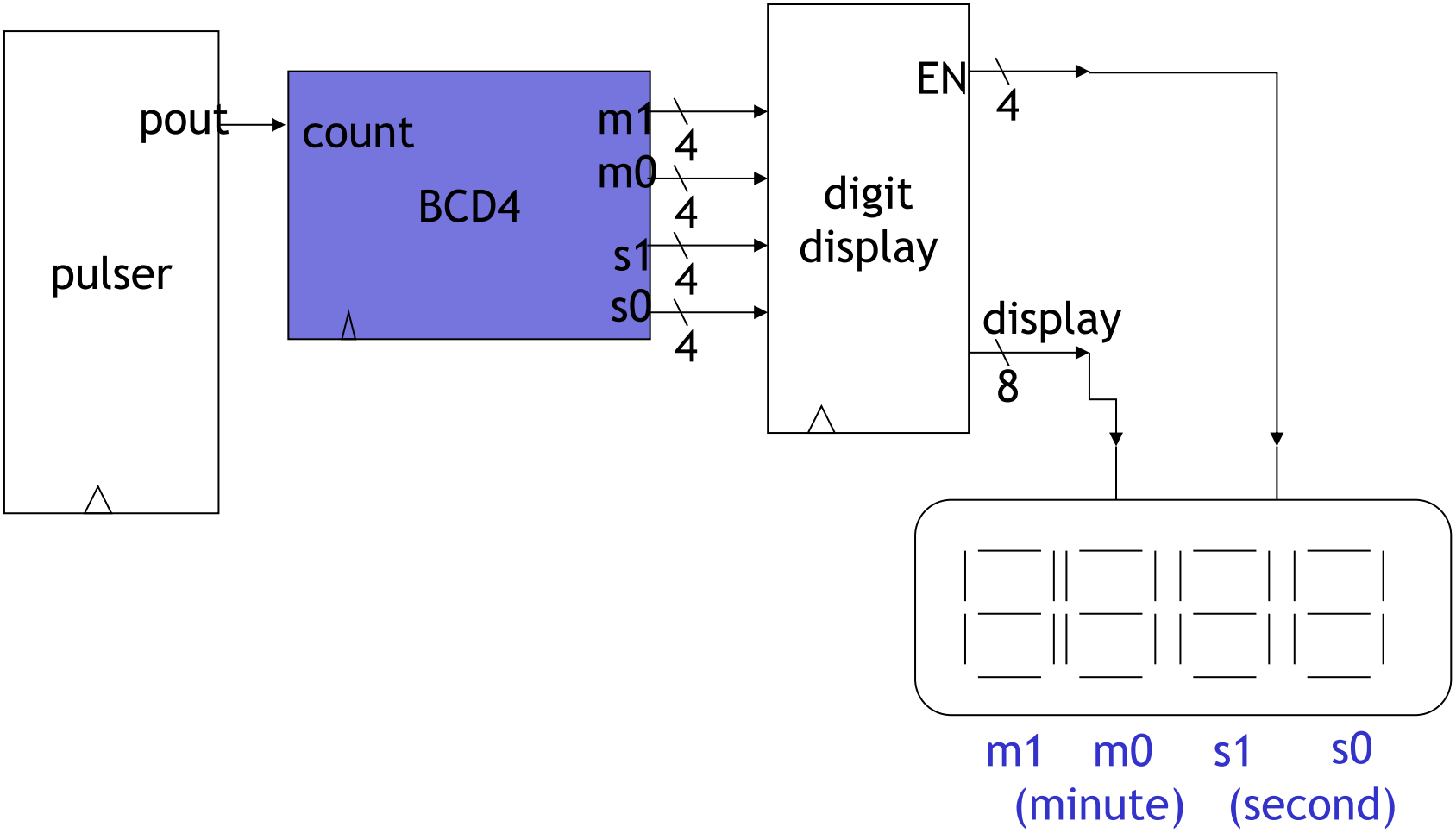
- pout=1 iff Q=999?



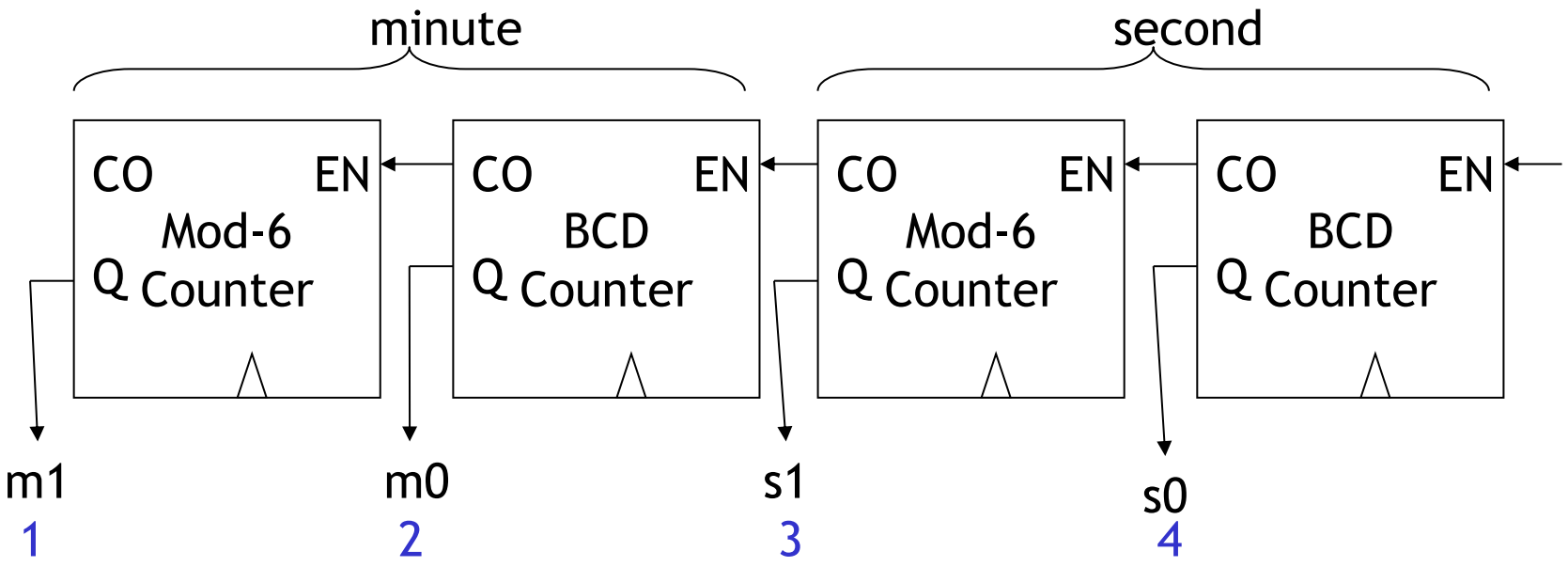
General scheme to check A==B?



The counter architecture for m1m0:s1s0

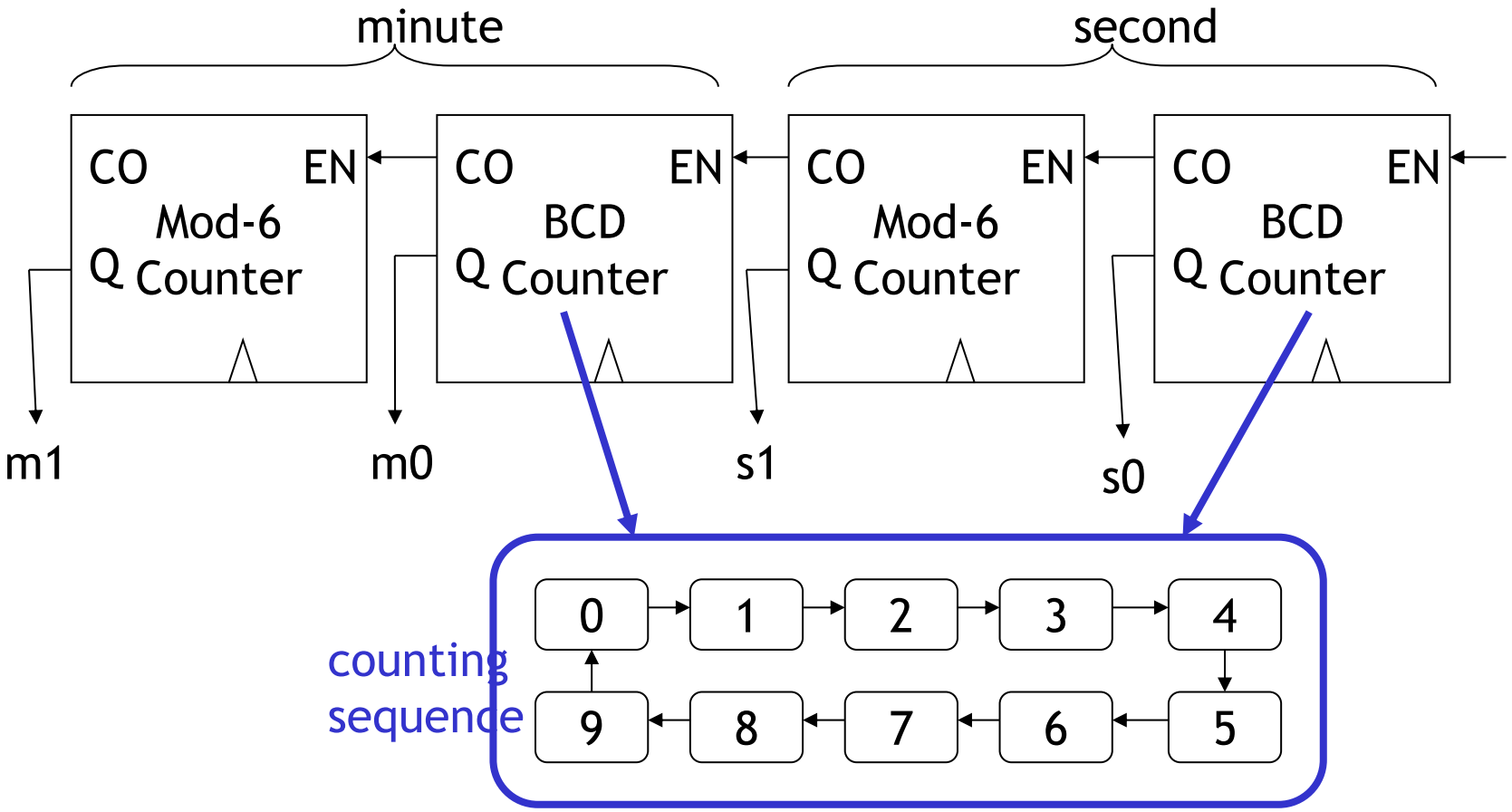


Architecture of the counter

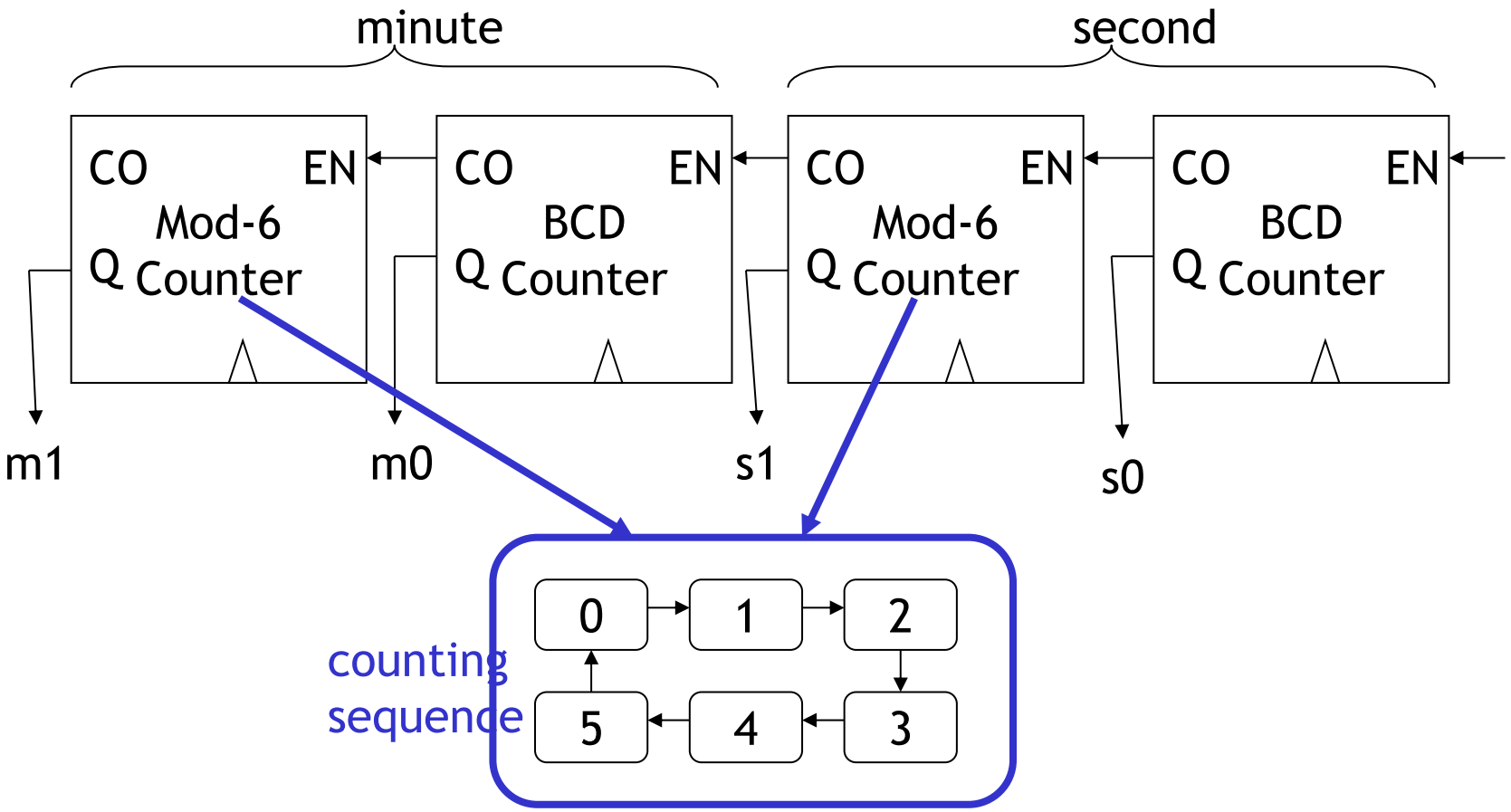


current time: 12:34

Architecture of the counter

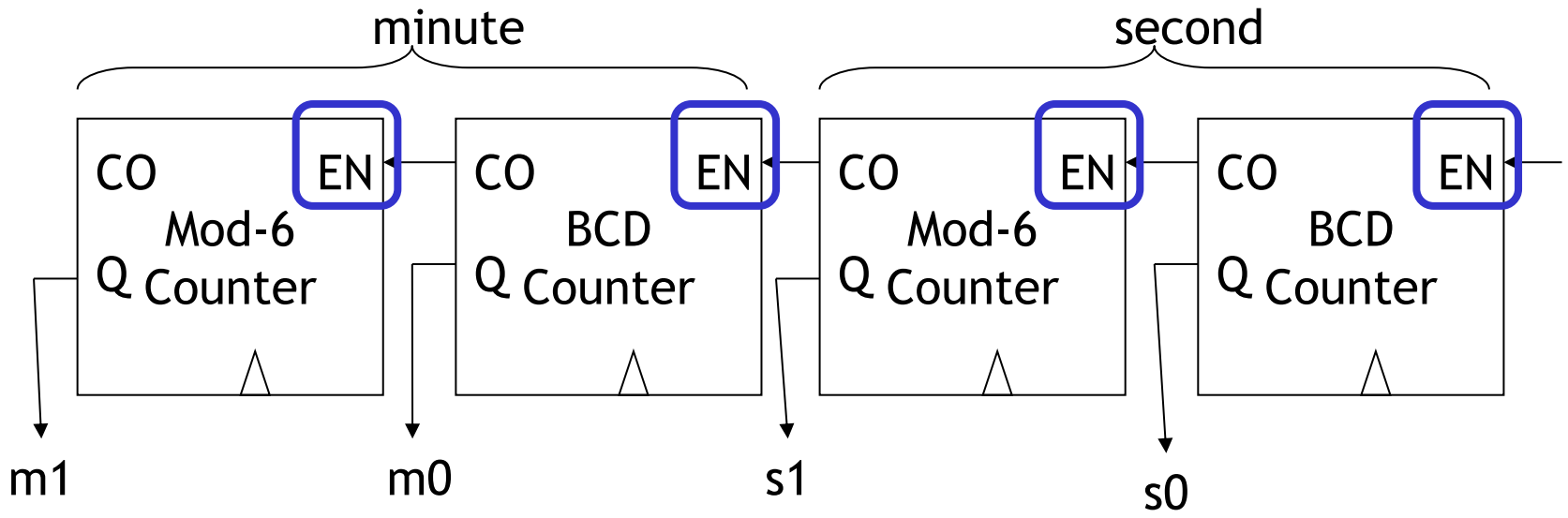


Architecture of the counter



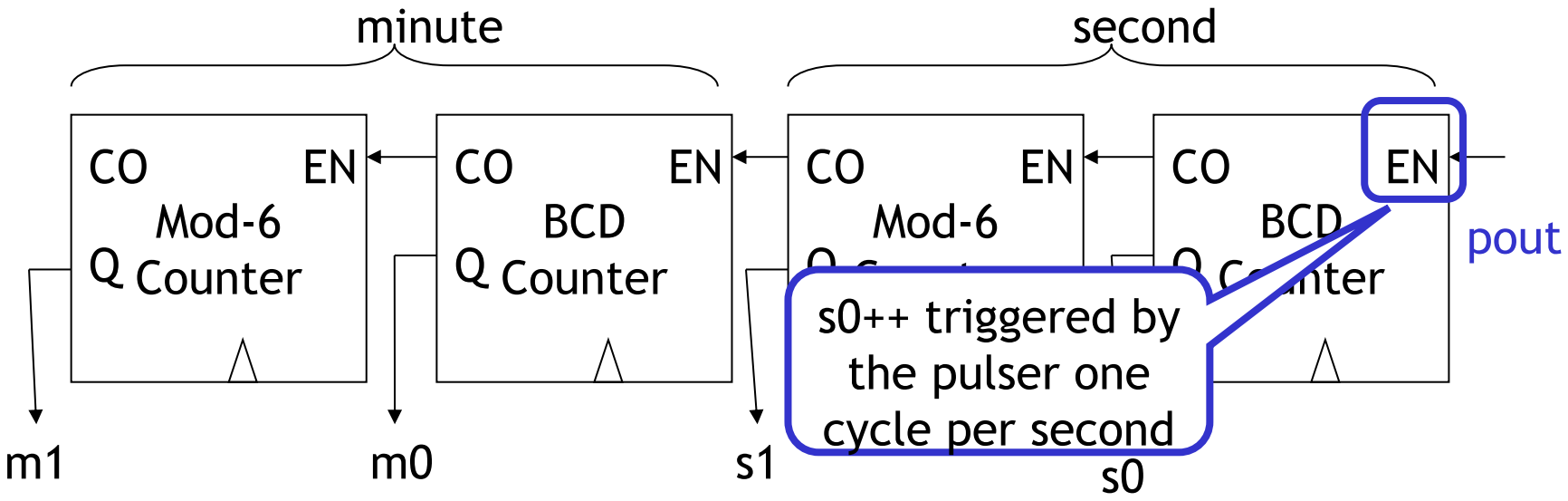
Architecture of the counter

- each counter receives an enable signal to make Q(t+1)=Q(t)+1
- the counting is triggered by previous digit or pulser

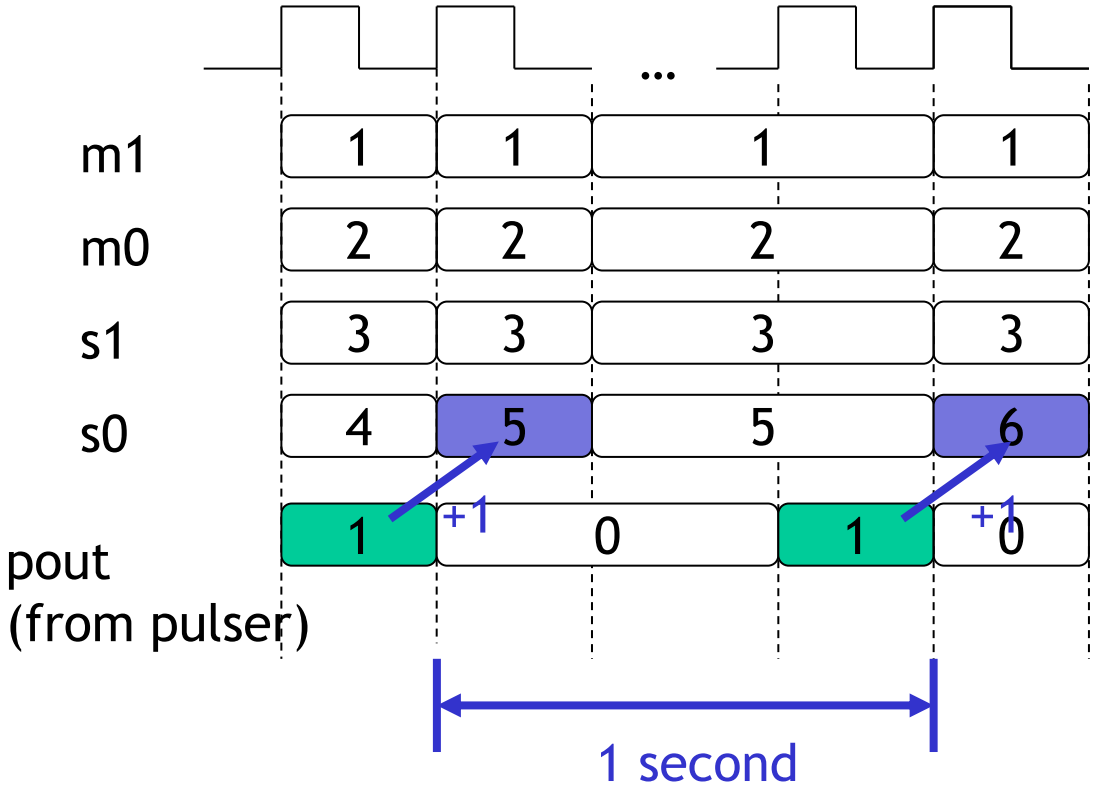


Architecture of the counter

- each counter receives an enable signal to make Q(t+1)=Q(t)+1
- the counting is triggered by previous digit or pulser

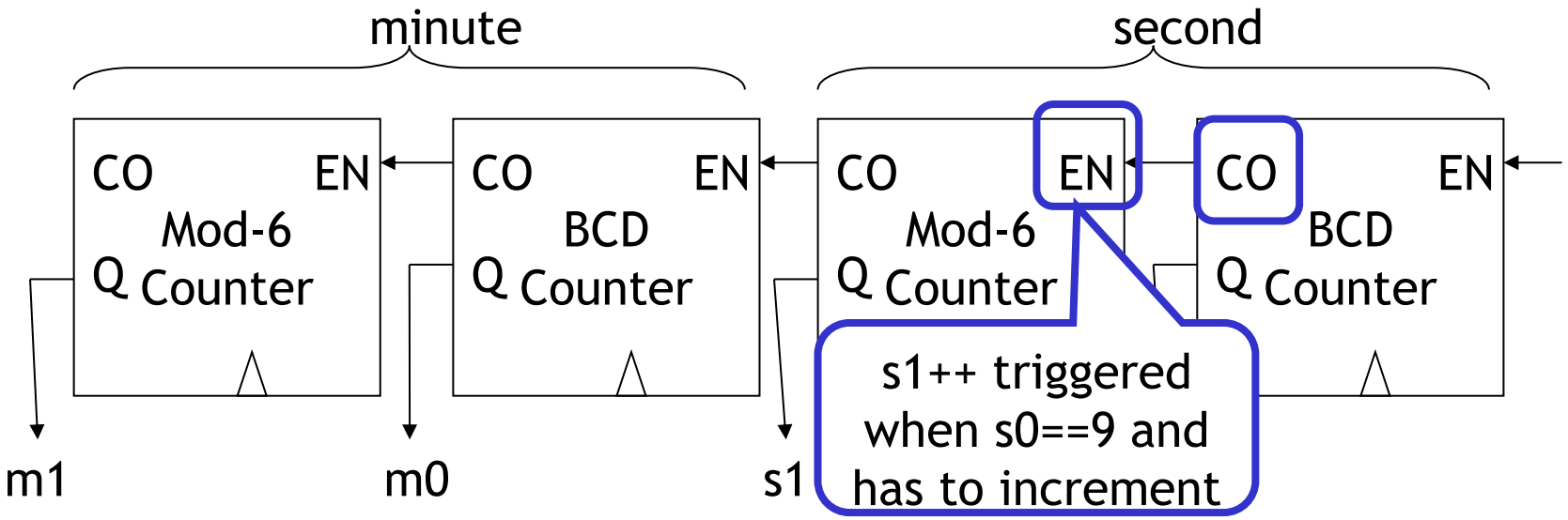


Timing diagram of carry-in

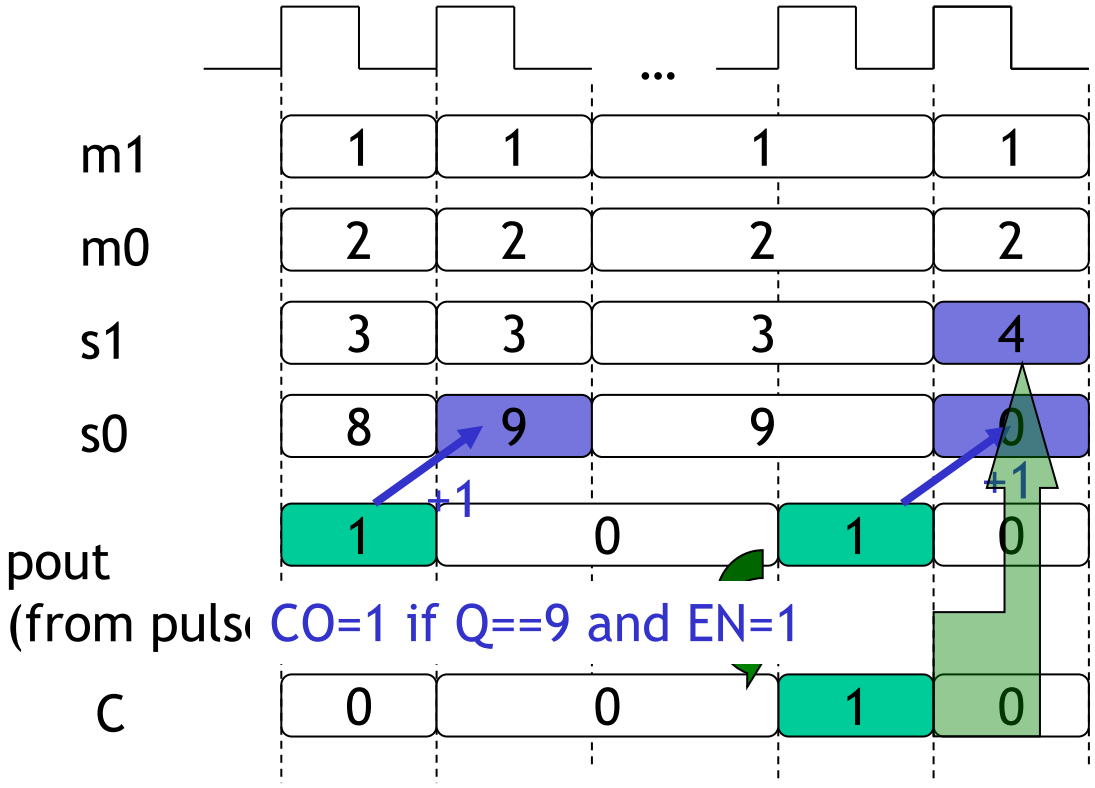


Architecture of the counter

- each counter receives an enable signal to make Q(t+1)=Q(t)+1
- the counting is triggered by previous digit or pulser

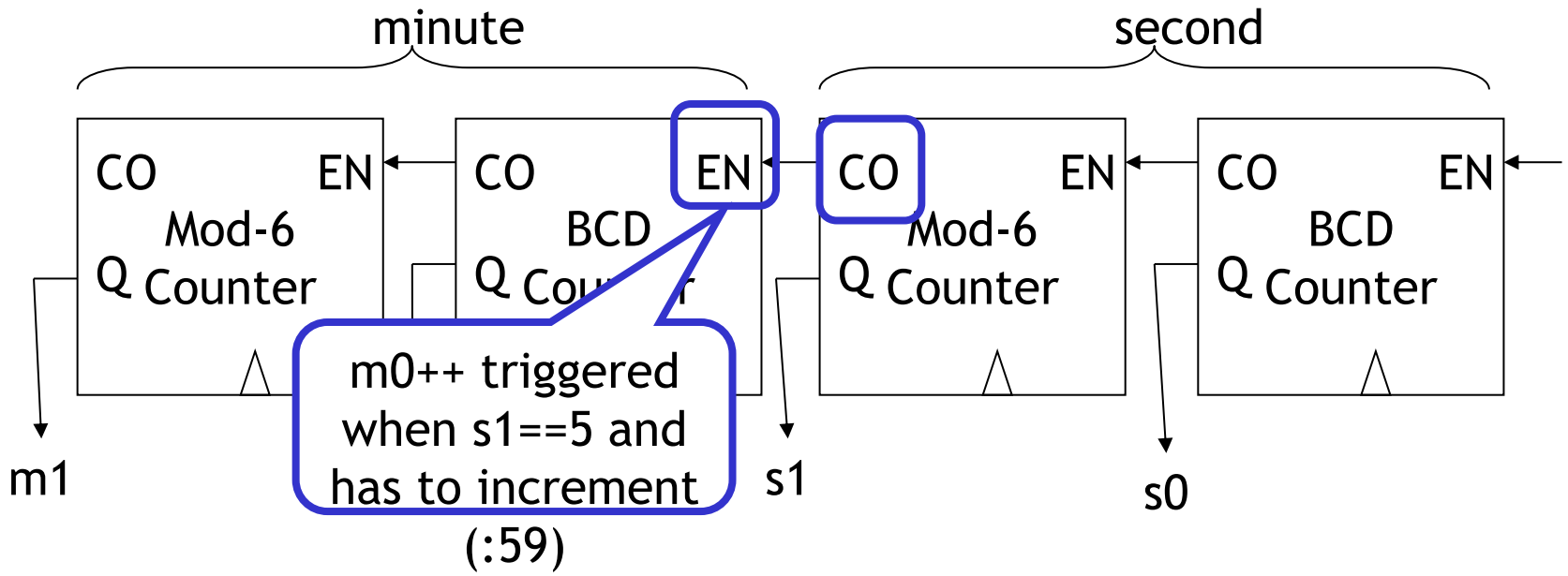


Timing diagram of carry-in

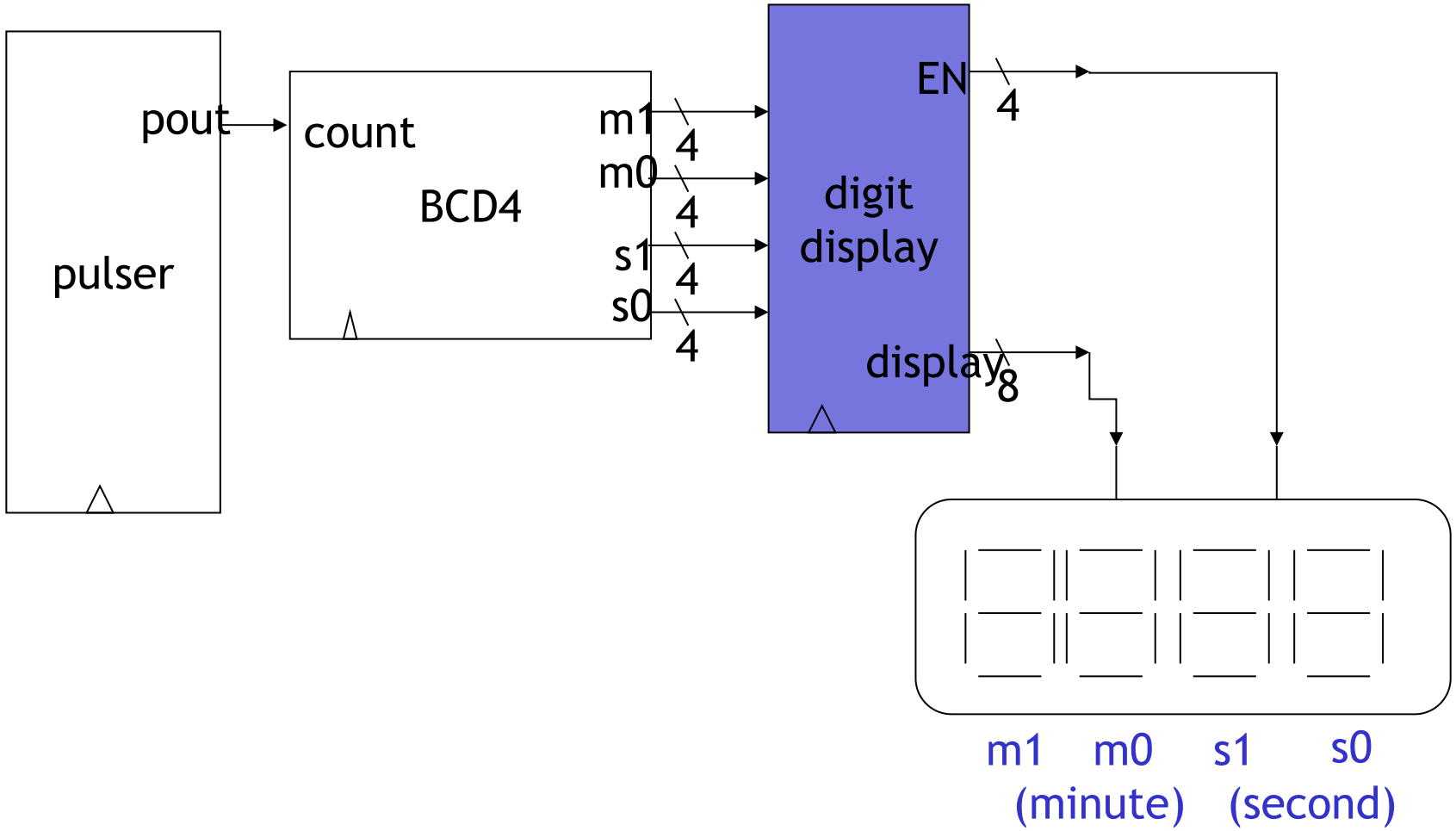


Architecture of the counter

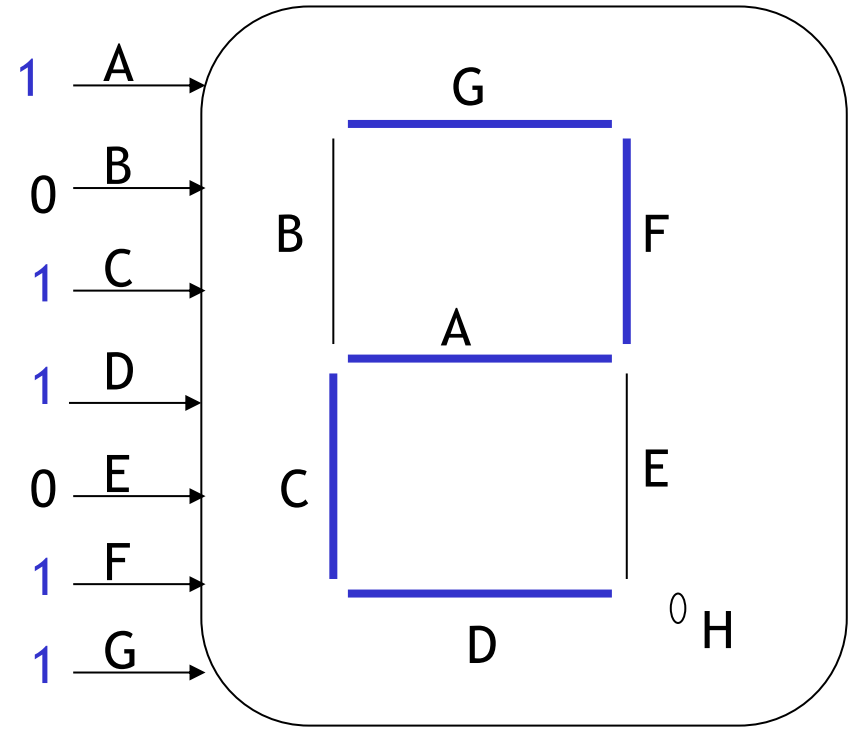
- each counter receives an enable signal to make Q(t+1)=Q(t)+1
- the counting is triggered by previous digit or pulser



7-Segment Digit Display Architecture

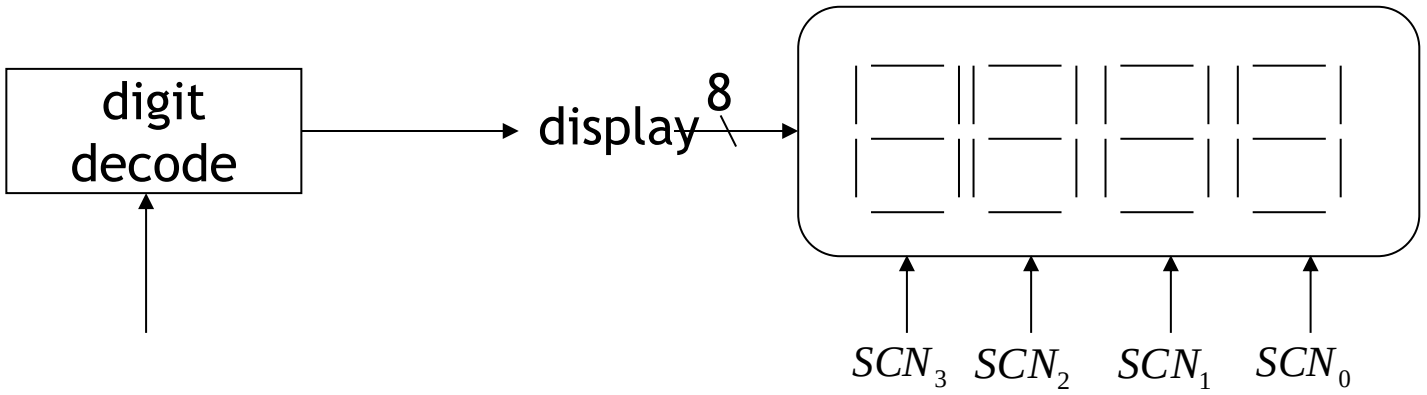


7-segment LED display



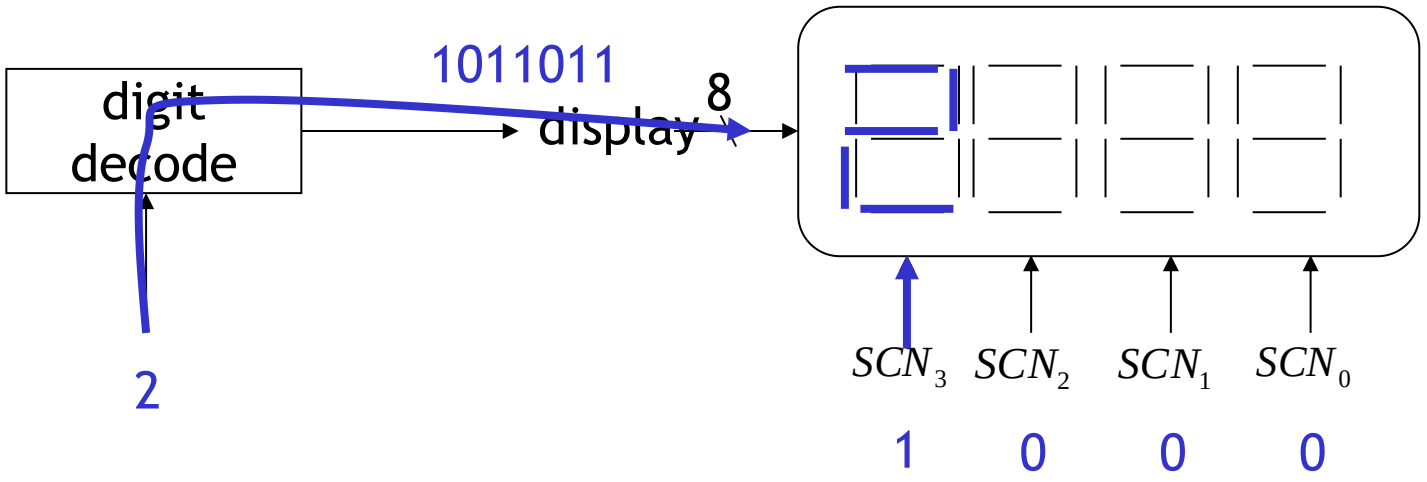
Group of 7-seg digit display

- sharing decoded display signal



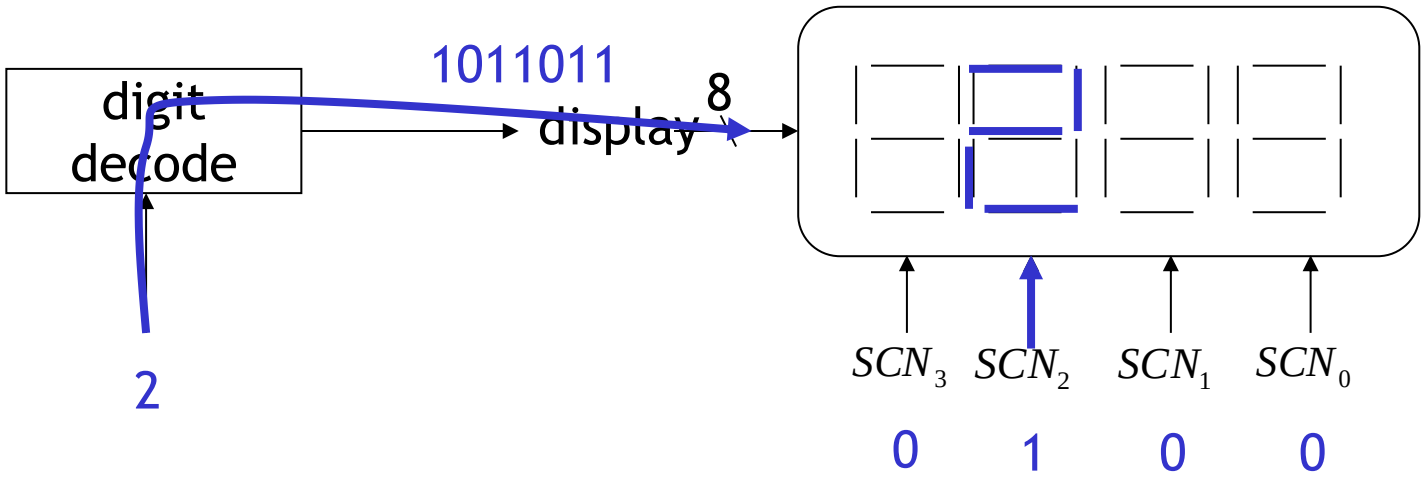
Group of 7-seg digit display

- sharing decoded display signal



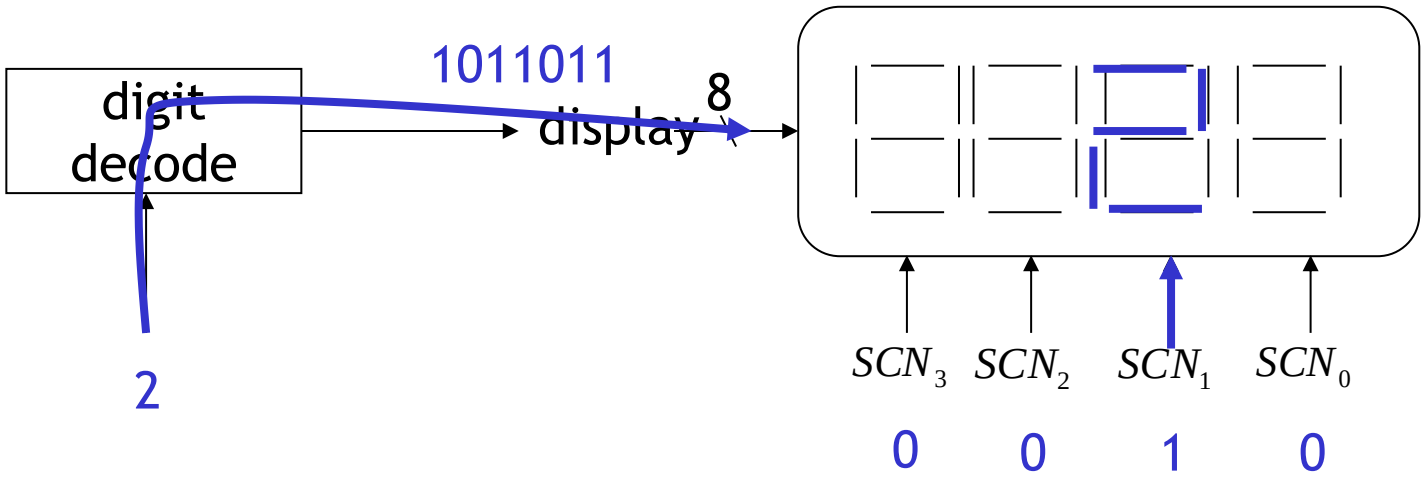
Group of 7-seg digit display

- sharing decoded display signal



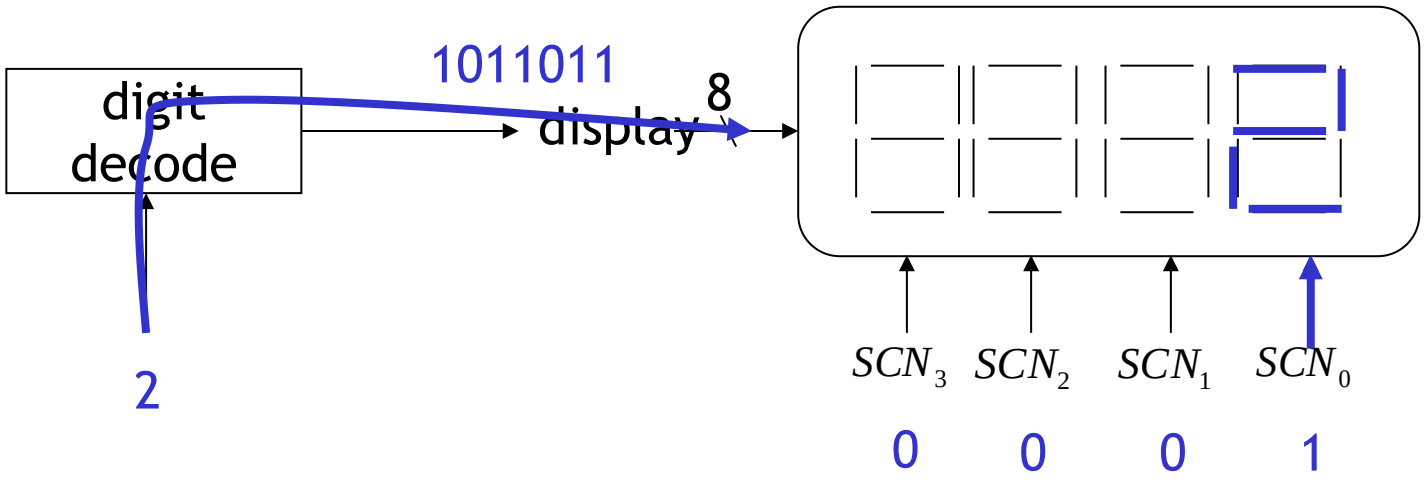
Group of 7-seg digit display

- sharing decoded display signal

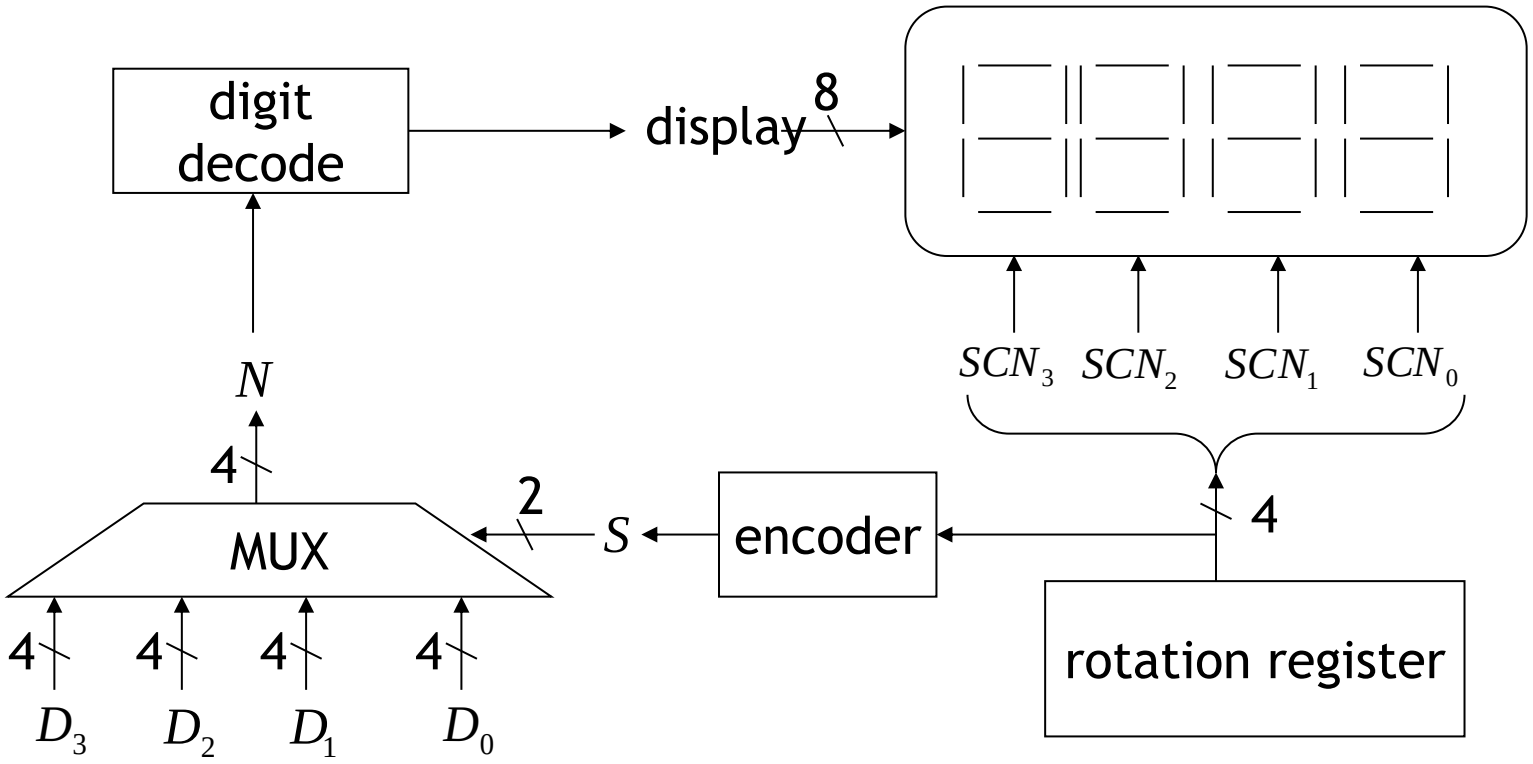


Group of 7-seg digit display

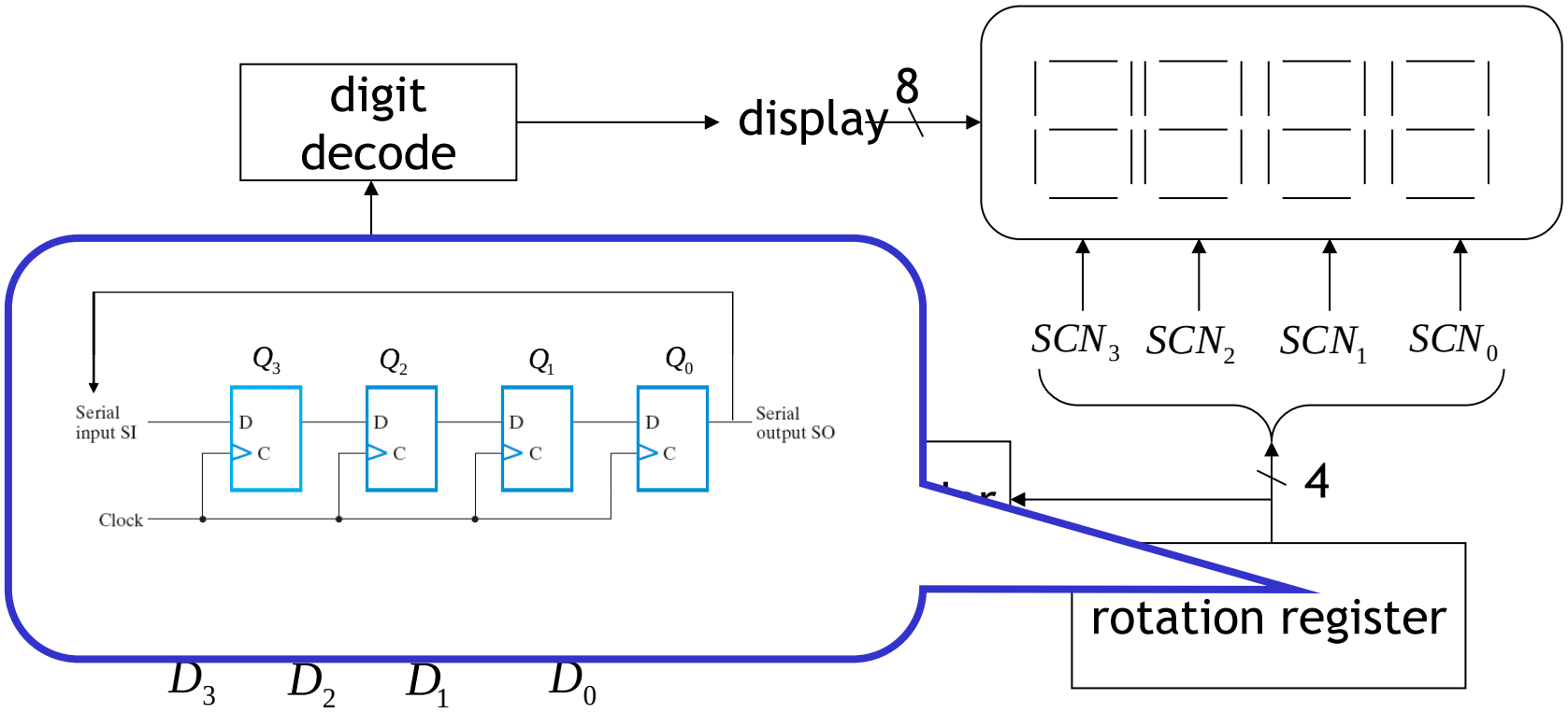
- sharing decoded display signal



General Design Diagram

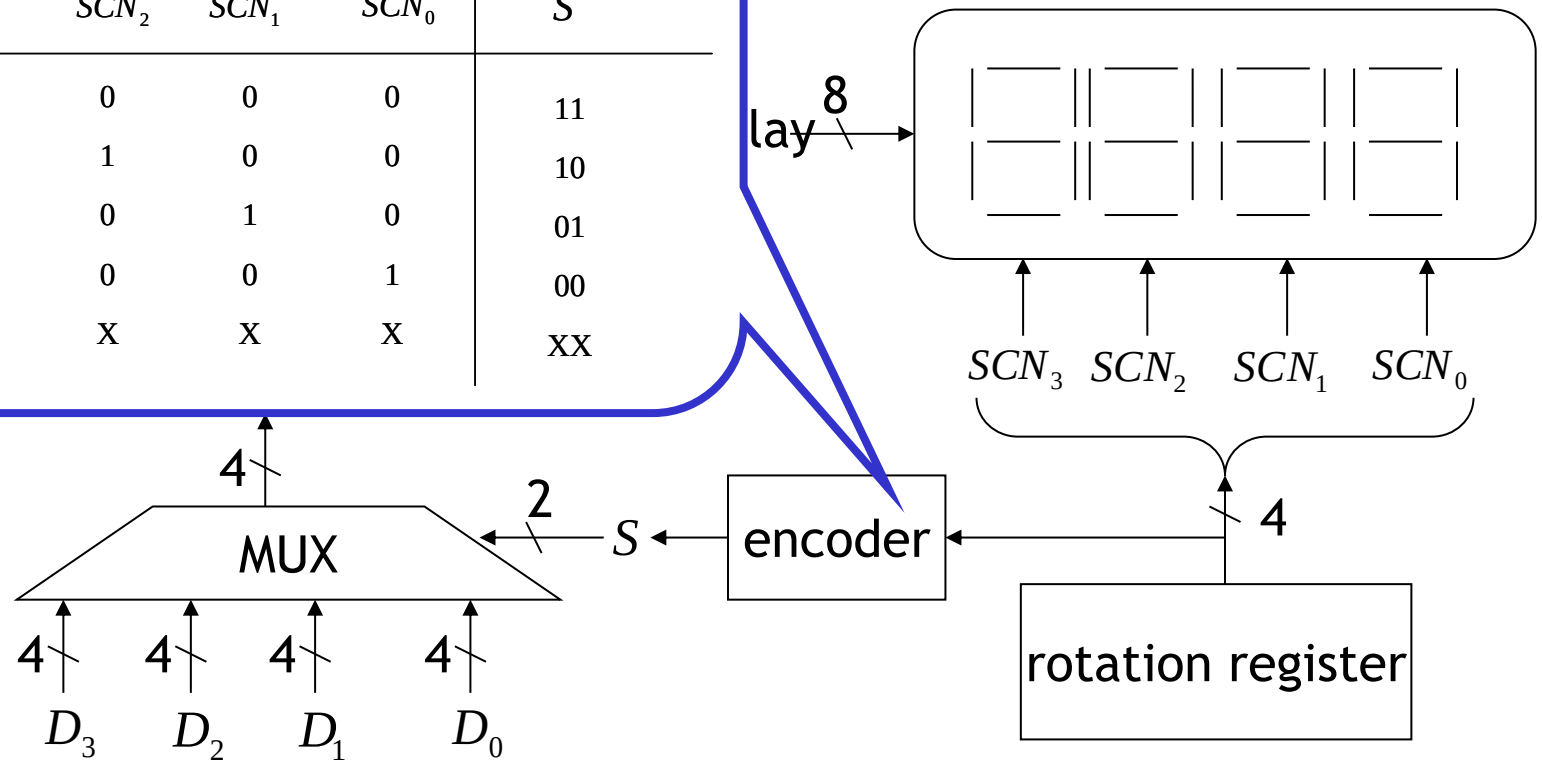


Rotation register to send SCN

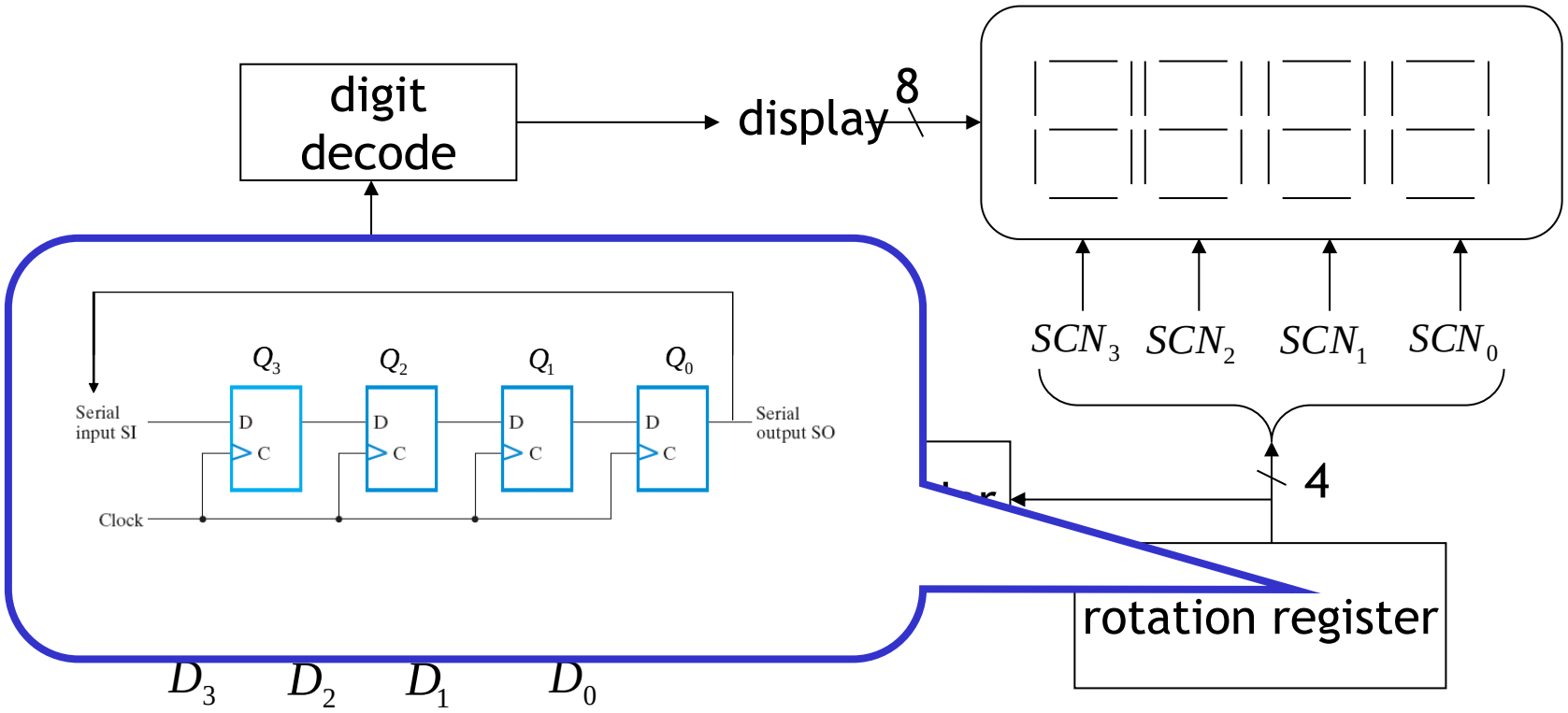


Cooperation of rotation register and MUX control

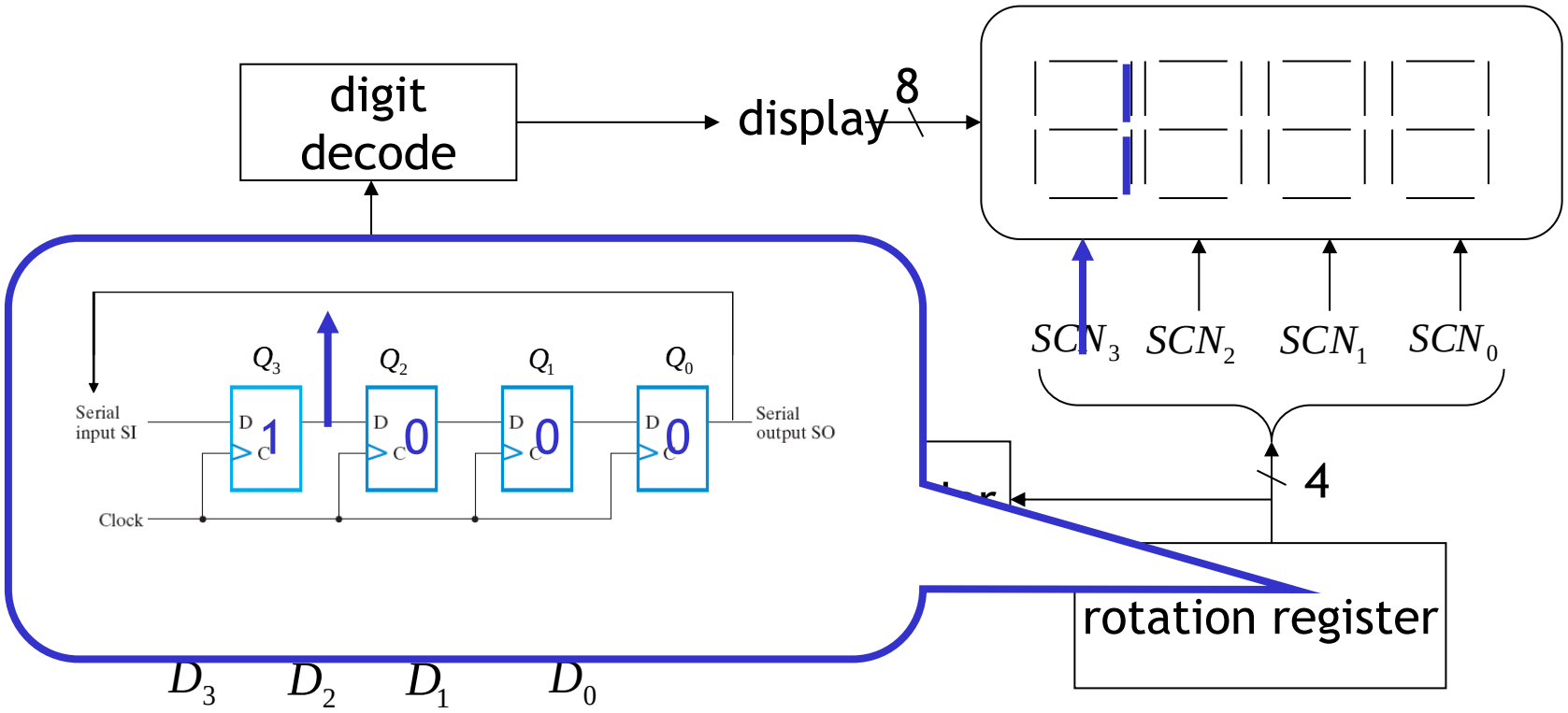
SCN_3	SCN_2	SCN_1	SCN_0	S
1	0	0	0	11
0	1	0	0	10
0	0	1	0	01
0	0	0	1	00
X	X	X	X	XX



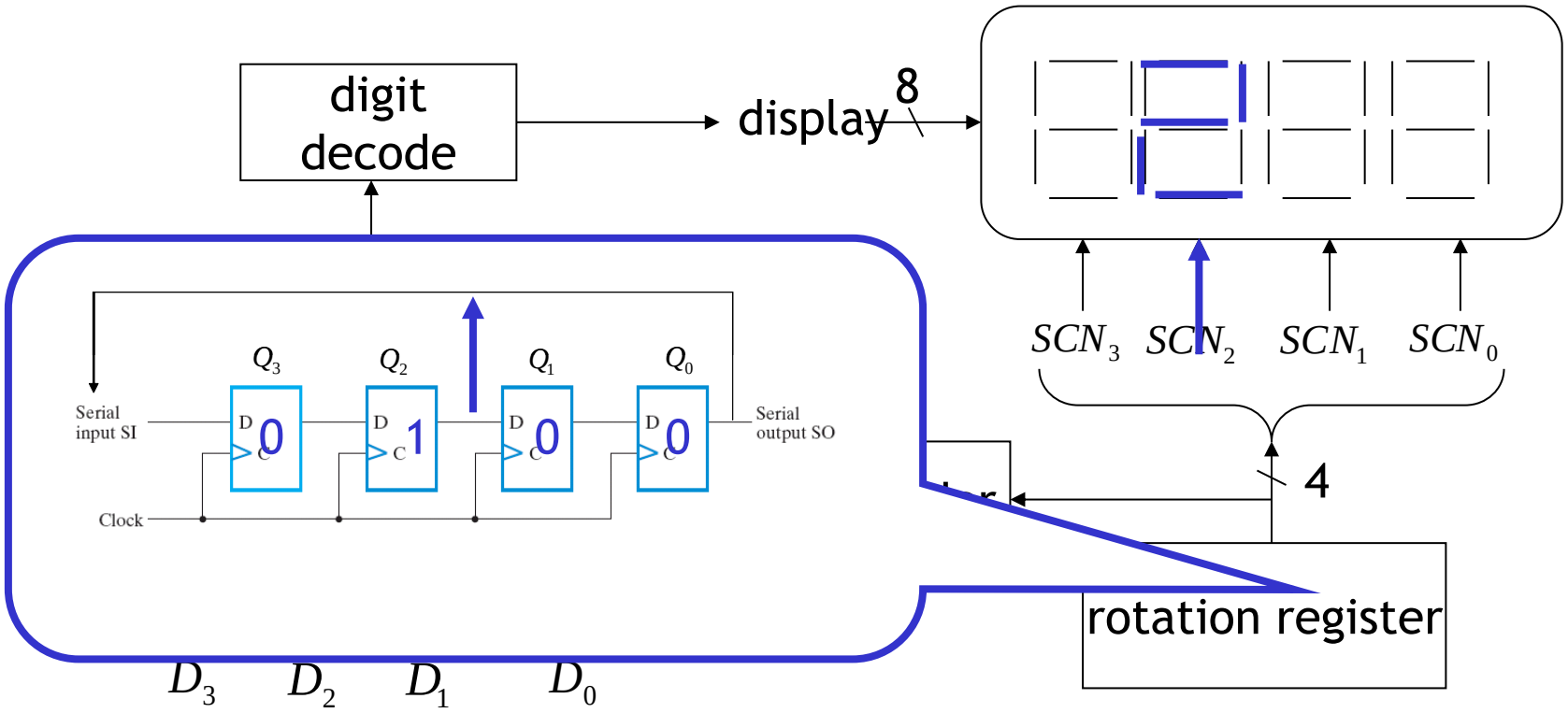
Rotation register to send SCN



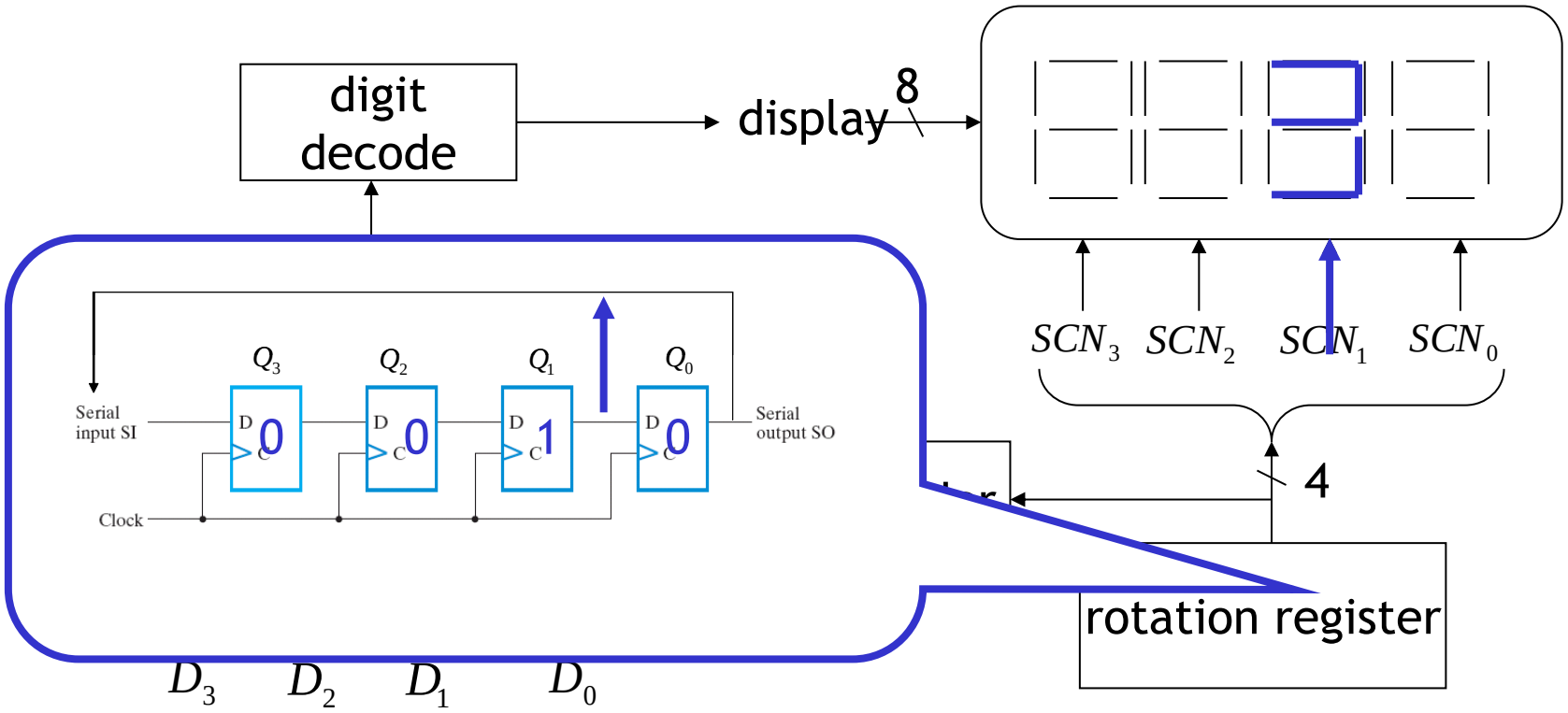
Rotation register to send SCN



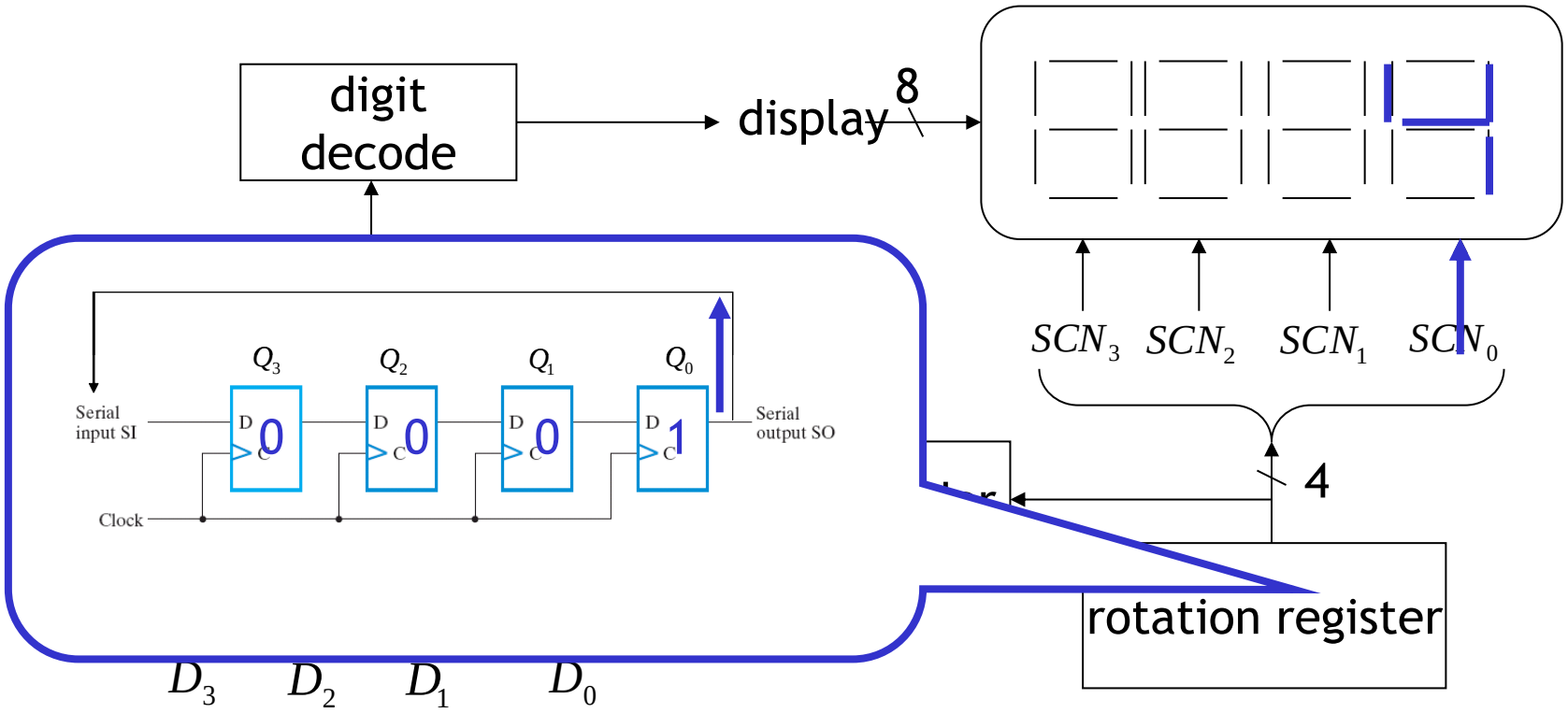
Rotation register to send SCN



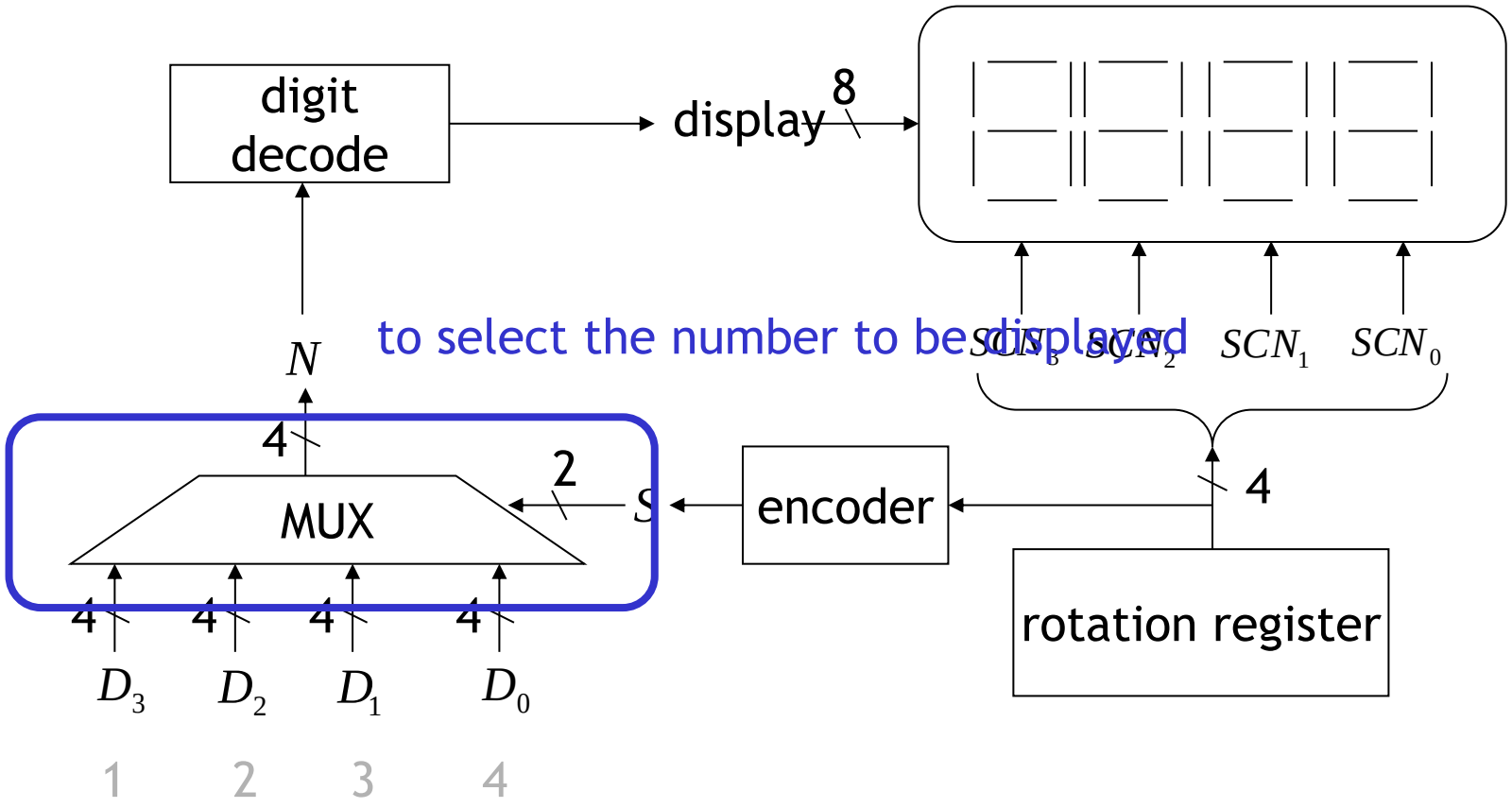
Rotation register to send SCN



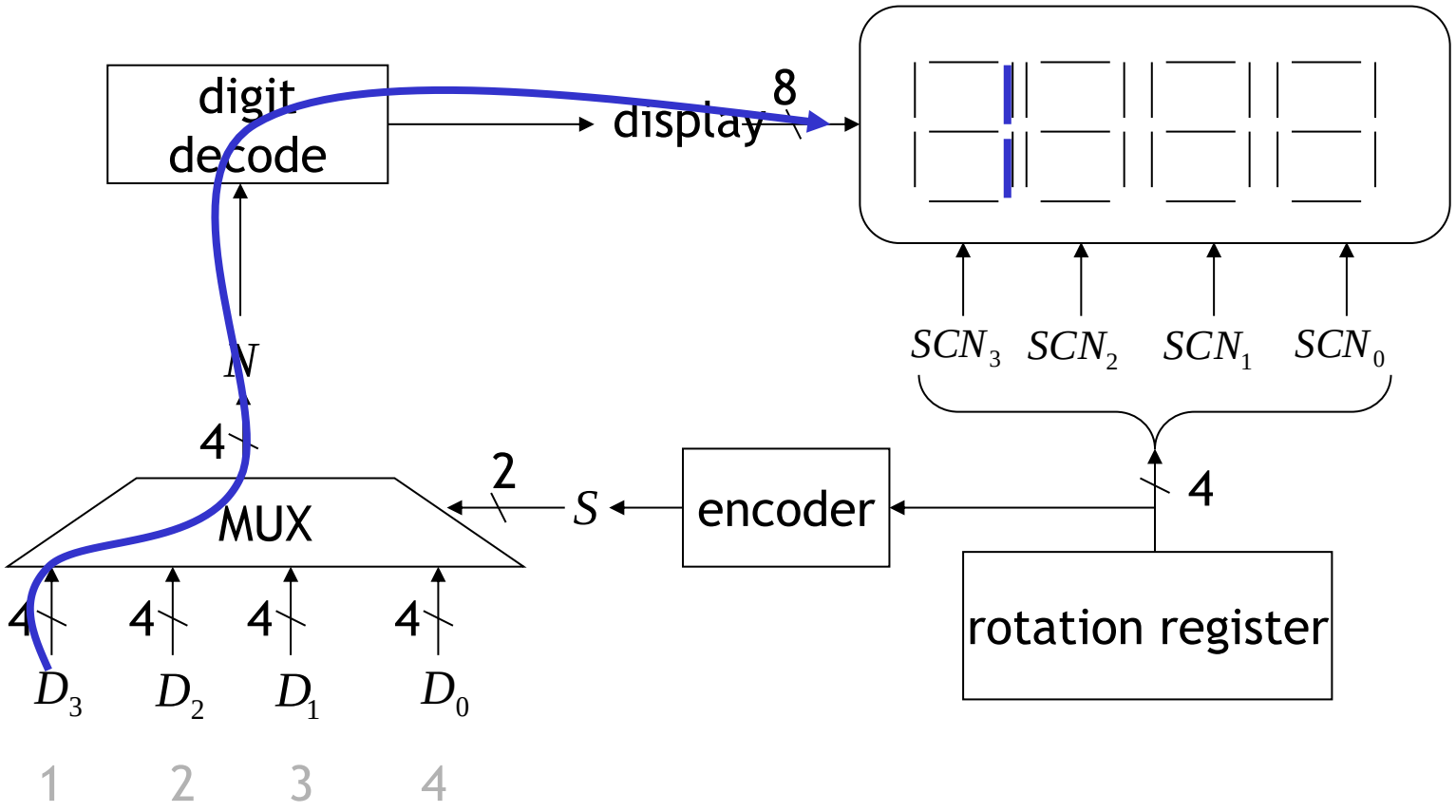
Rotation register to send SCN



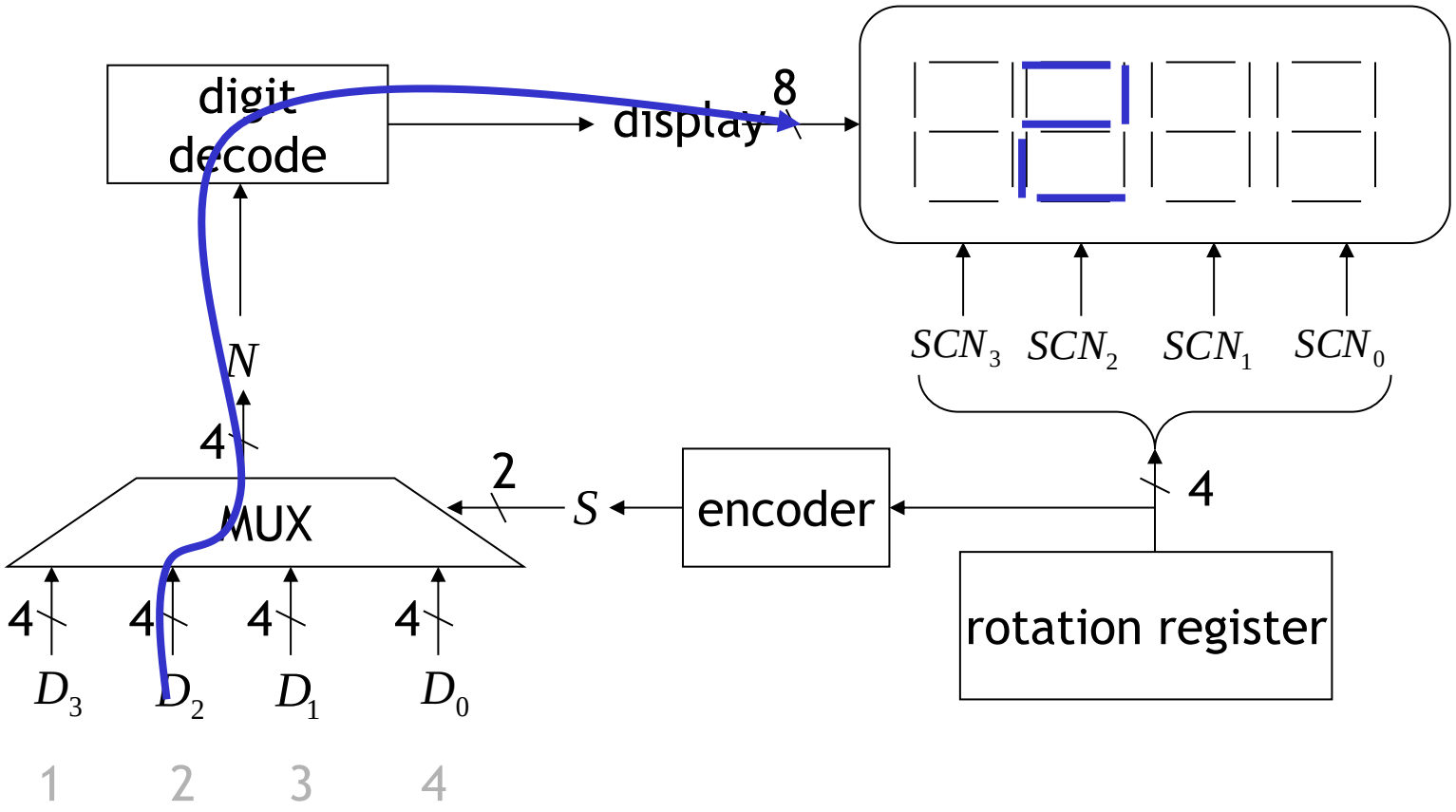
Select display number by MUX



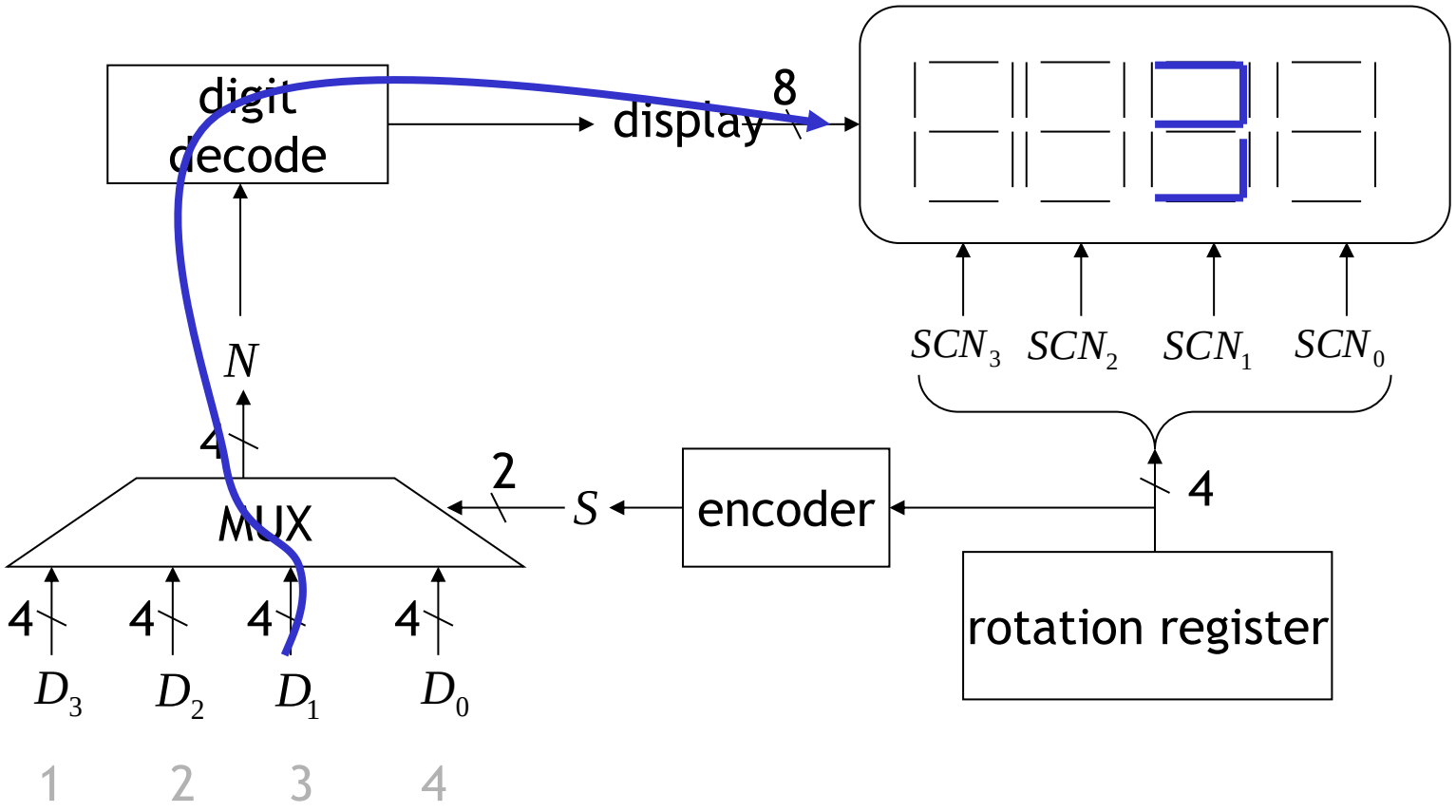
Select display number by MUX



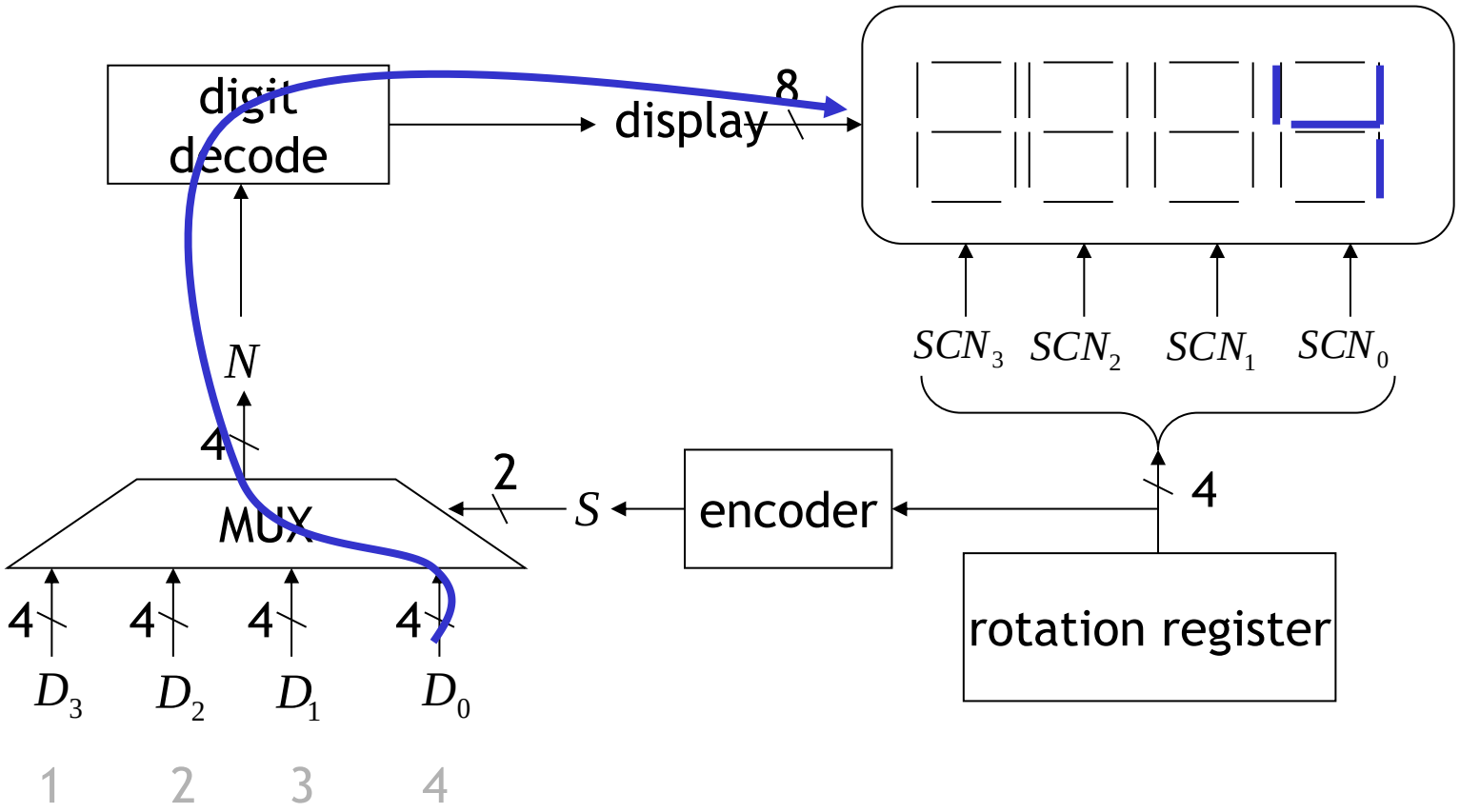
Select display number by MUX



Select display number by MUX

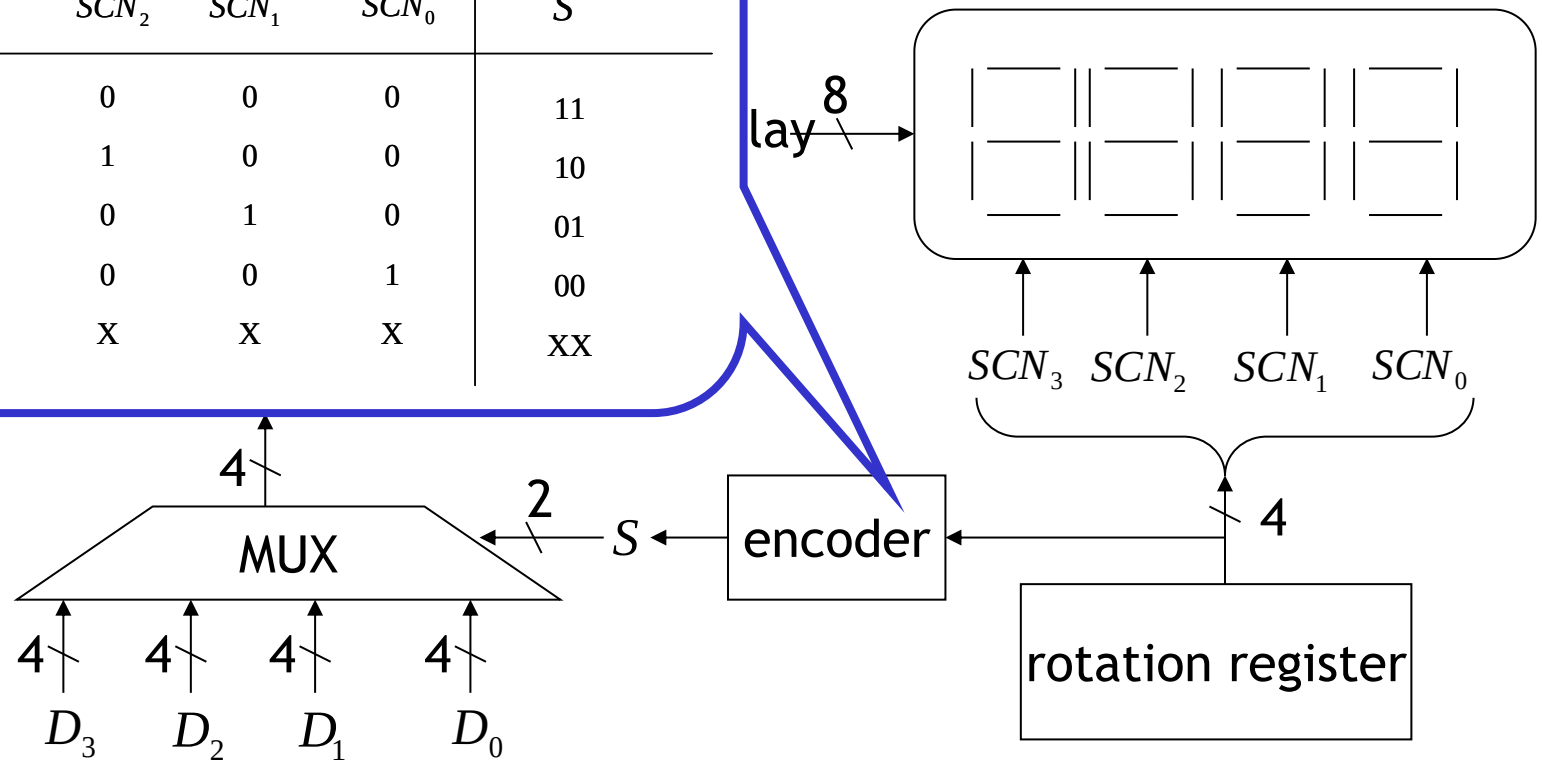


Select display number by MUX

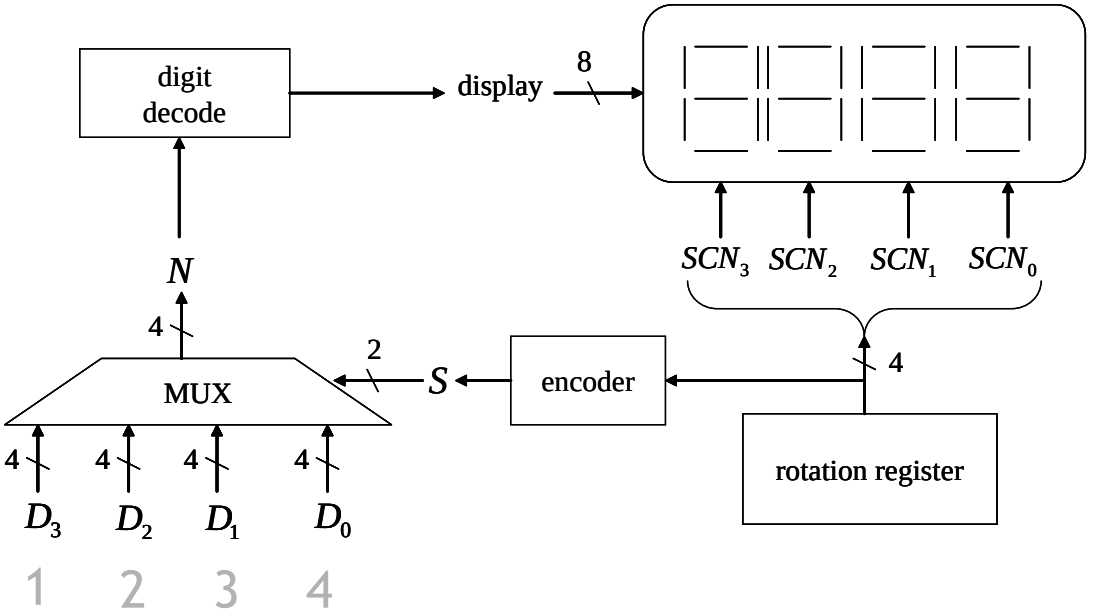


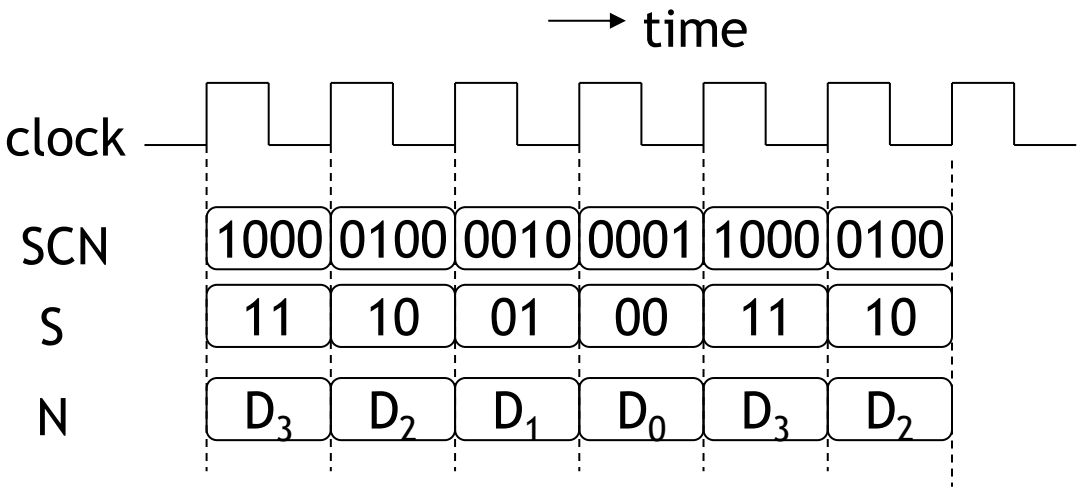
Cooperation of rotation register and MUX control

SCN_3	SCN_2	SCN_1	SCN_0	S
1	0	0	0	11
0	1	0	0	10
0	0	1	0	01
0	0	0	1	00
X	X	X	X	XX

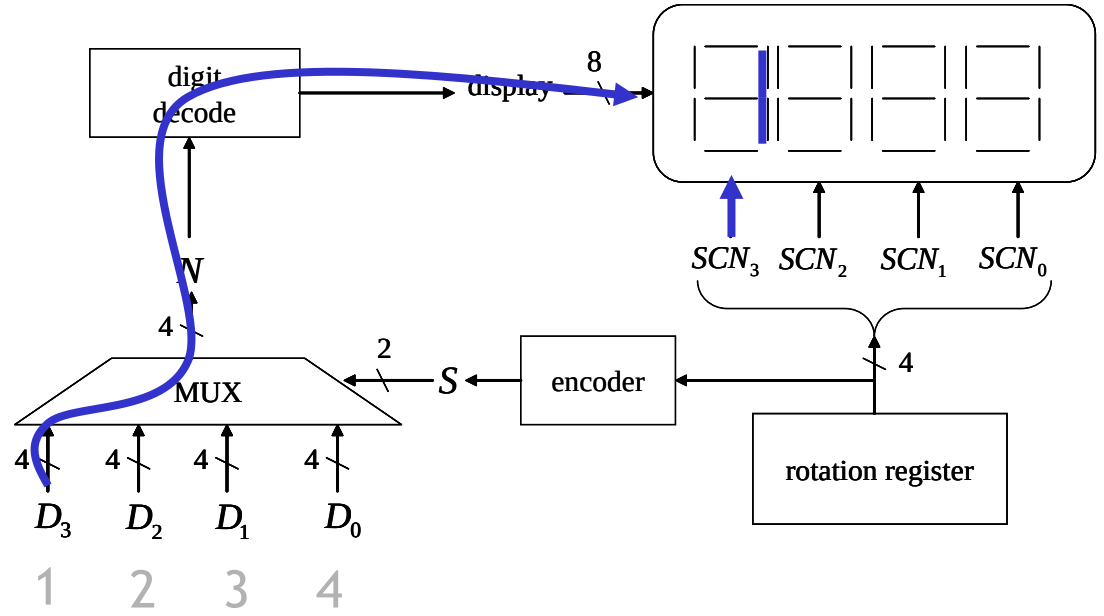


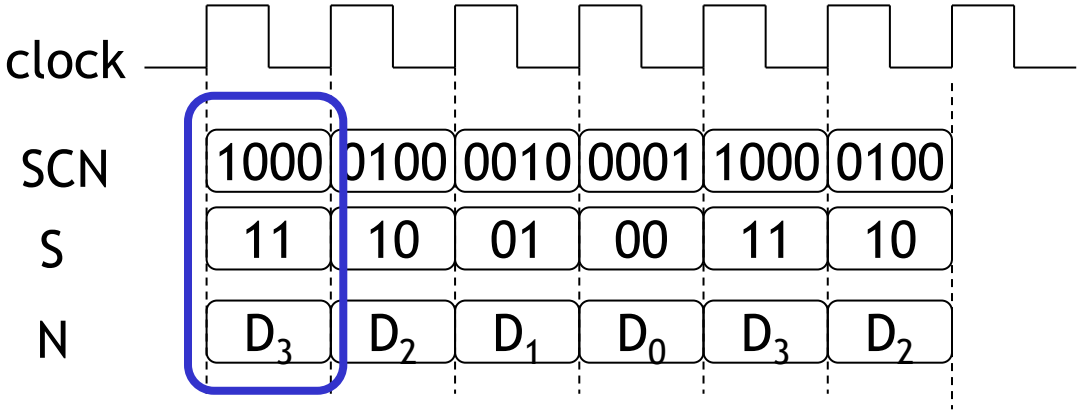
Timing Diagram





Timing Diagram





Timing Diagram