# CS 303
# Logic & Digital System Design

Ömer Ceylan

Sabancı Üniversitesi

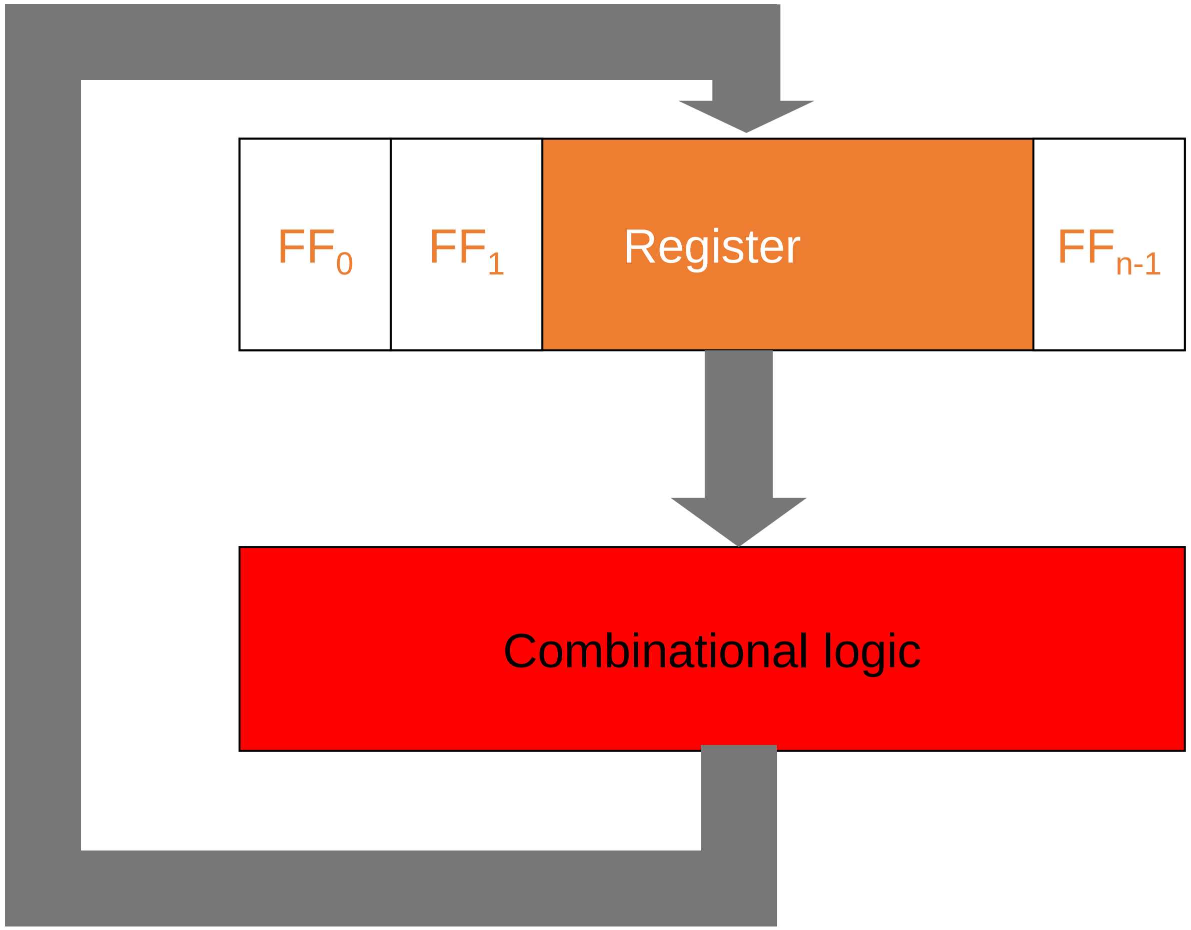
# Chapter 6
# Registers & Counters

## Registers

- Registers are clocked sequential circuits
- A register is a group of flip-flops
  - Each flip-flop capable of storing one bit of information
  - An n-bit register
    - consists of n flip-flops
    - capable of storing n bits of information
  - besides flip-flops, a register usually contains combinational logic to perform some simple tasks
  - In summary
    - flip-flops to hold information
    - combinational logic to control the state transition

## Counters

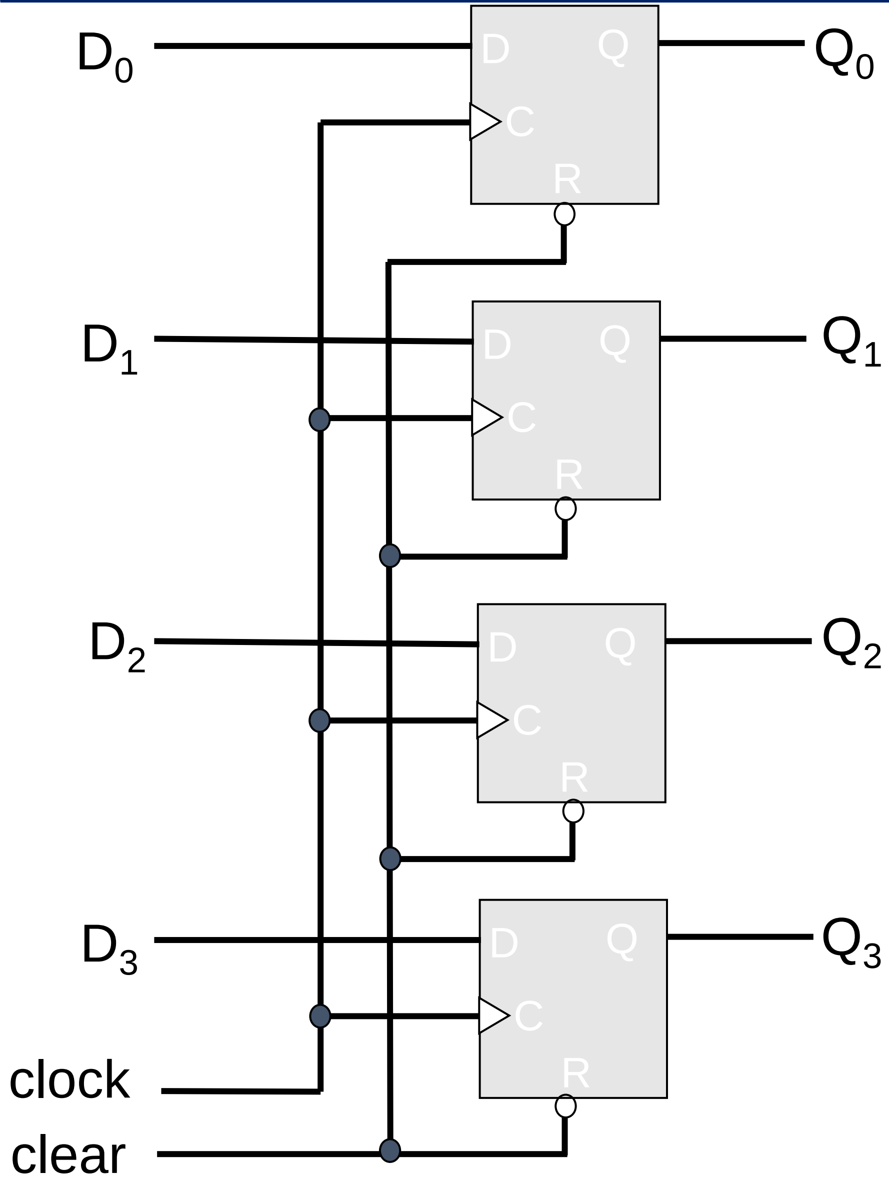
- A counter is essentially a register that goes through a <u>predetermined sequence of states</u>
- i.e., "Counting sequence"

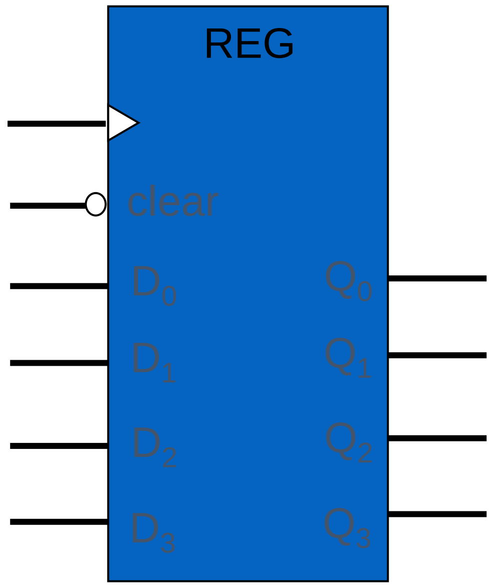


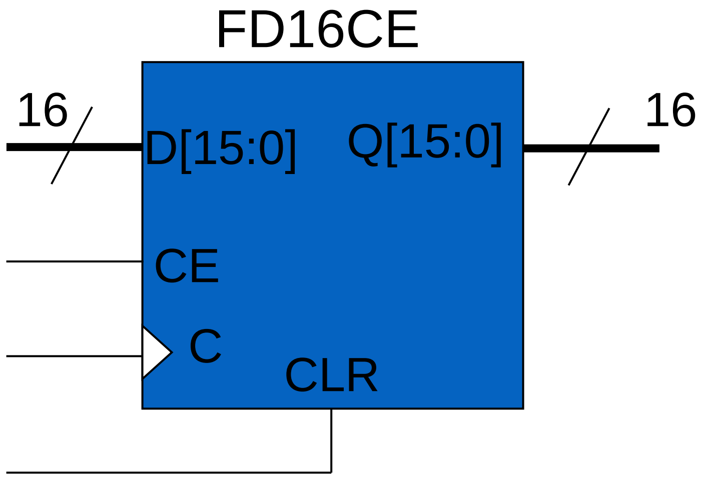
## Uses of Registers and Counters

- Registers are useful for storing and manipulating information
  - internal registers in microprocessors to manipulate data
- Counters are extensively used in control logic
  - PC (program counter) in microprocessors

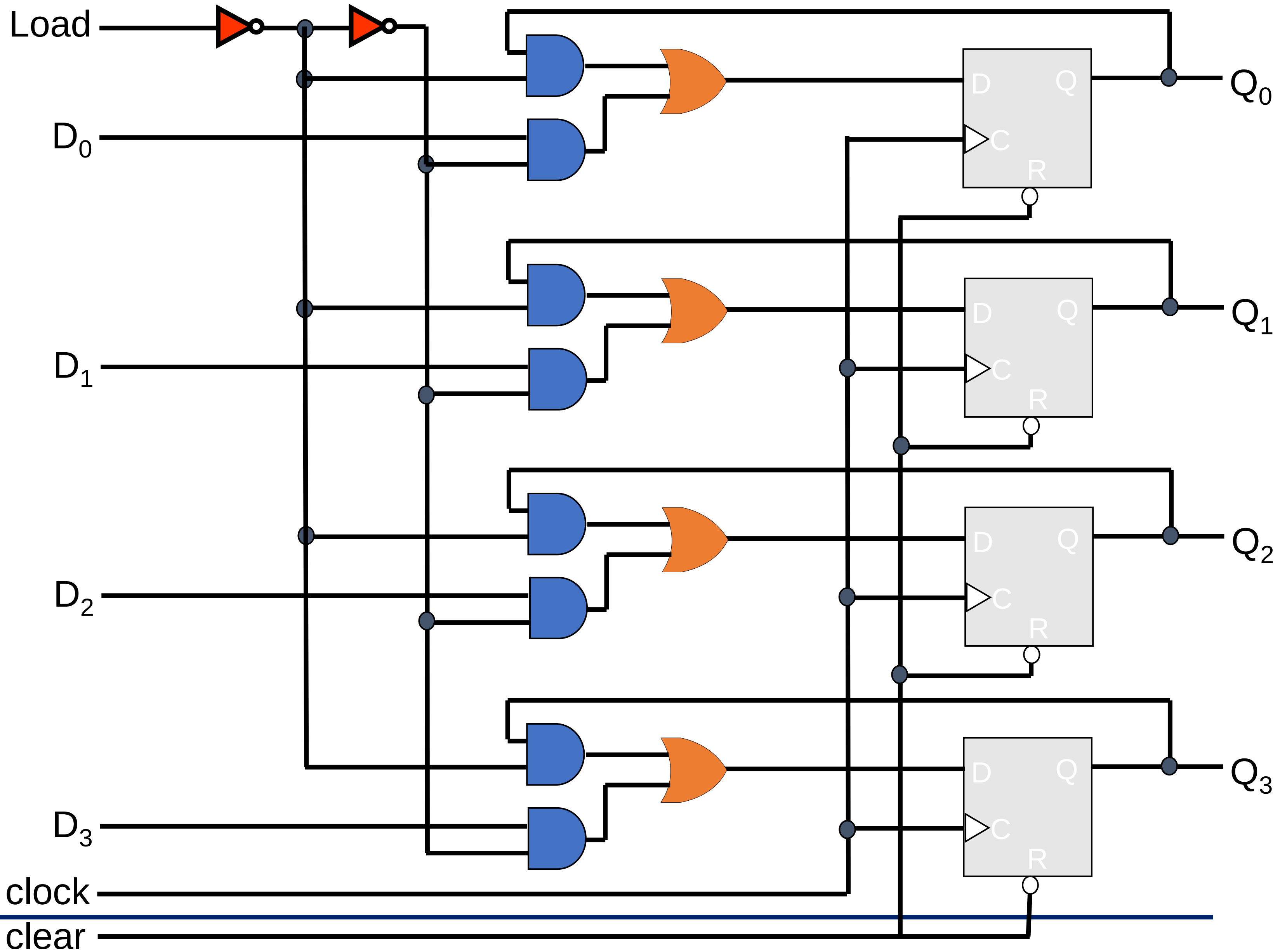
# 4-bit Register



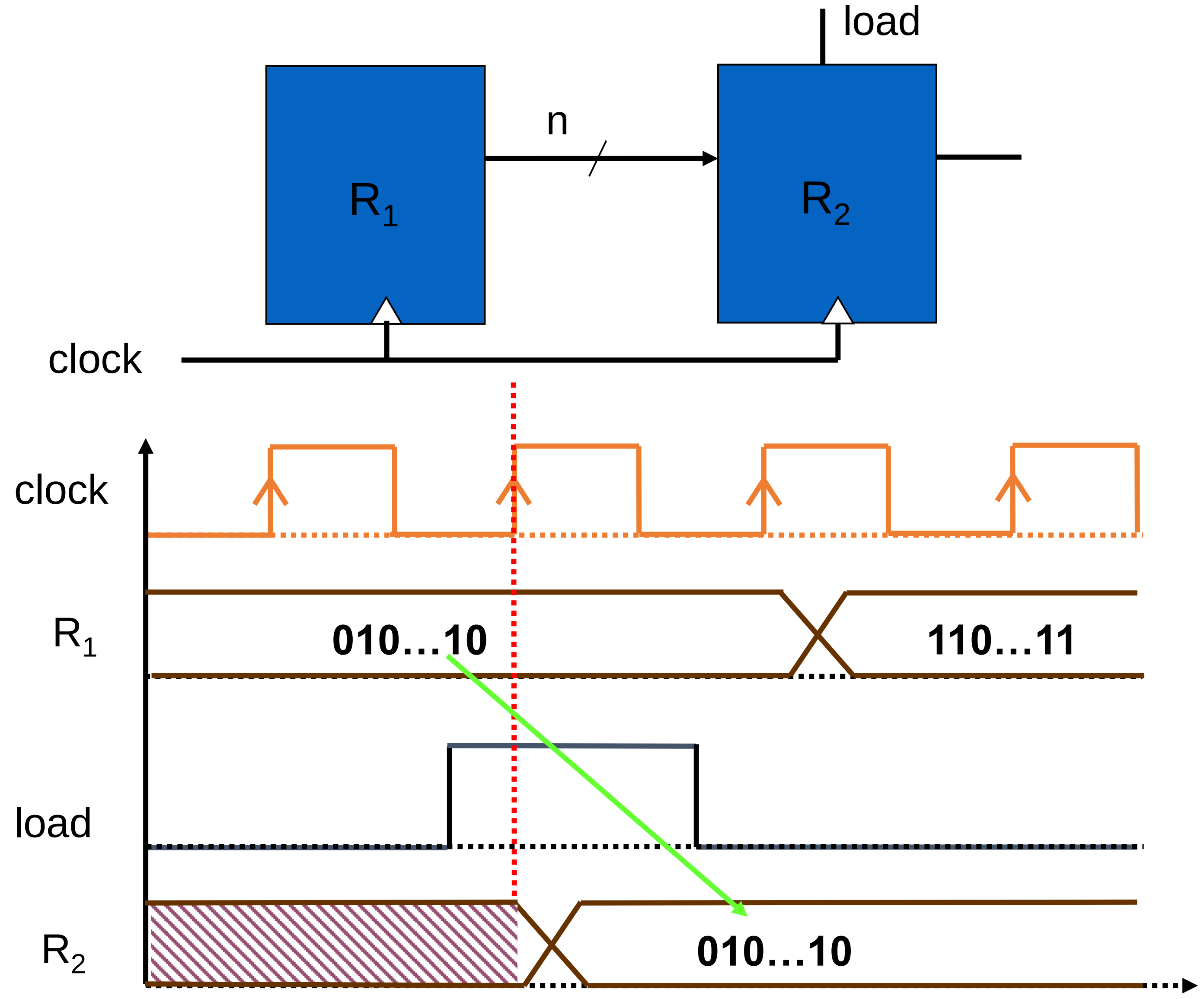




# Register with Parallel Load

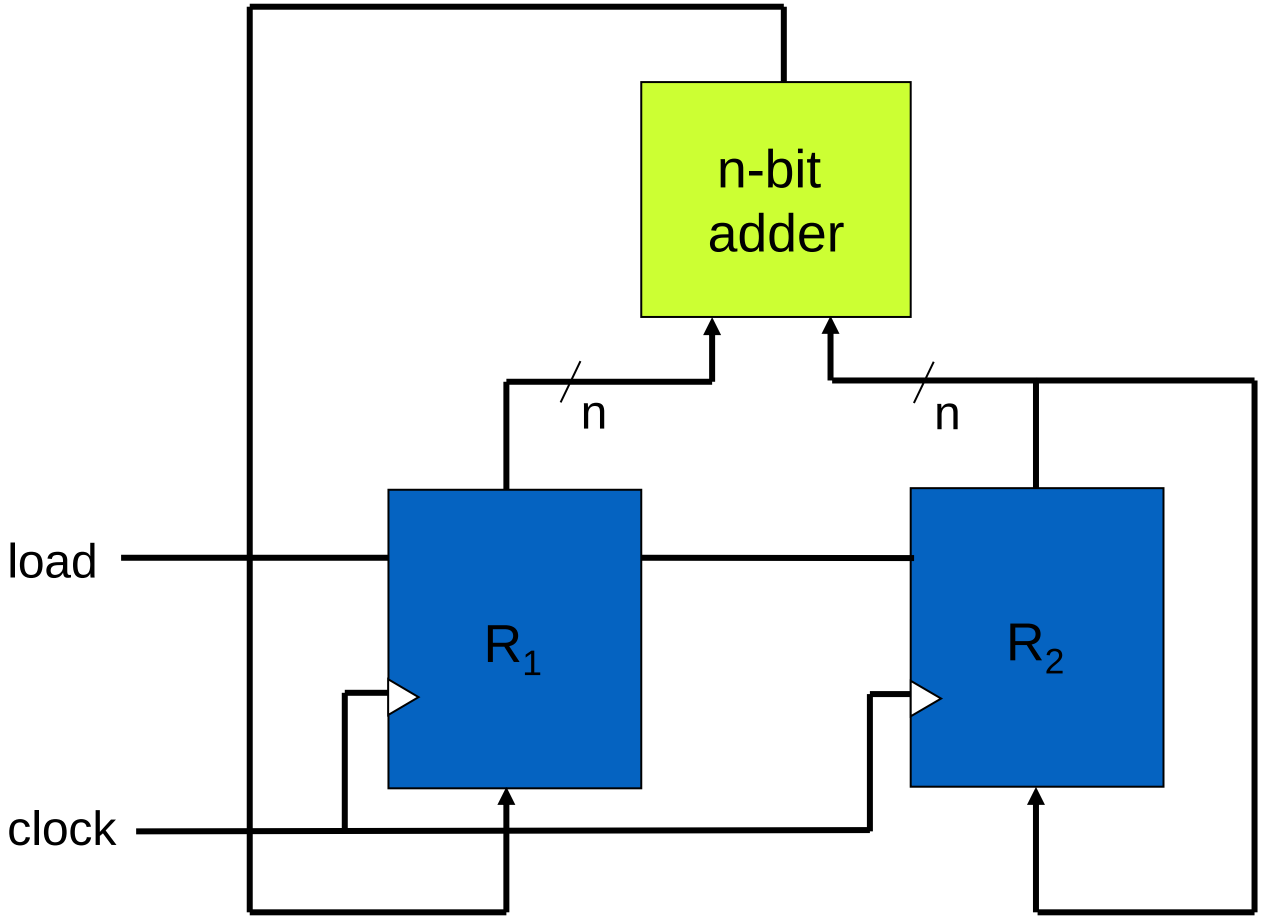


# Register Transfer 1/2



$R_2 \leftarrow R_1$

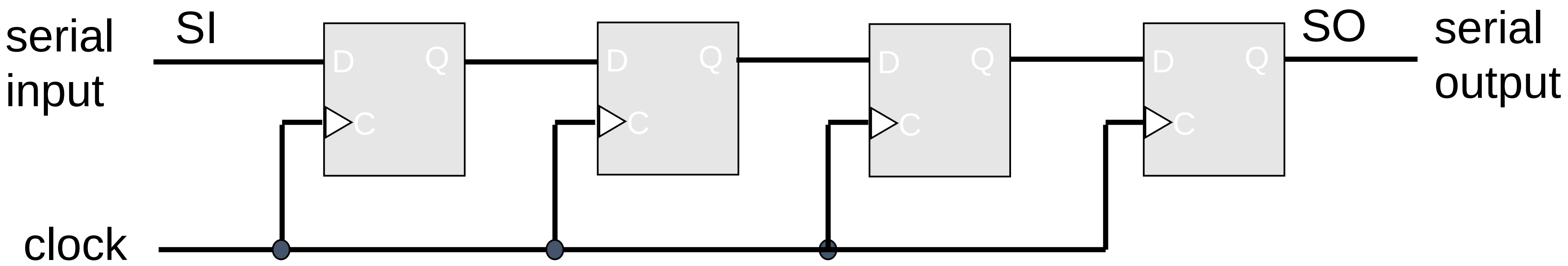
# Register Transfer 2/2



$R_1 \leftarrow R_1 + R_2$

## Shift Registers

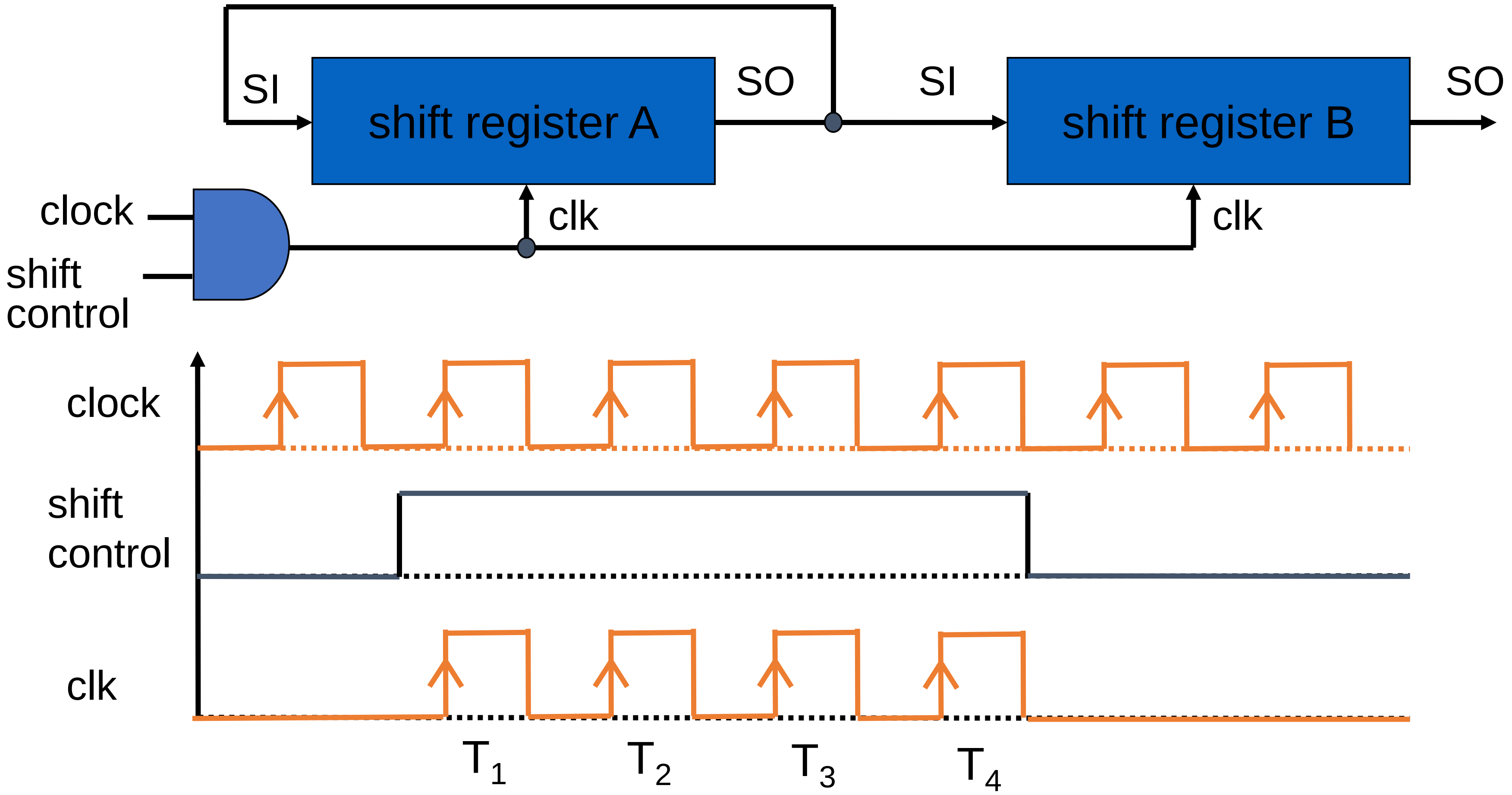
- A register capable of shifting its content in one or both directions
  - Flip-flops in cascade



- The state of an n-bit shift register can be transferred in n clock cycles

## Serial Mode

- A digital system is said to operate in serial mode when information is transferred and manipulated one bit a time.

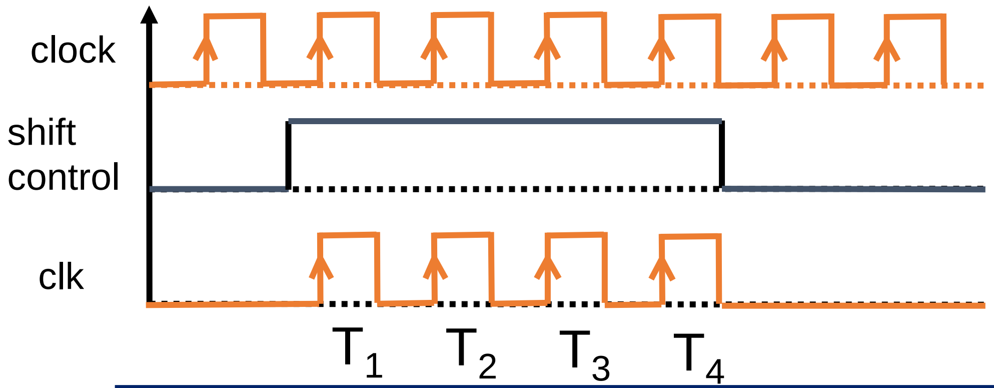


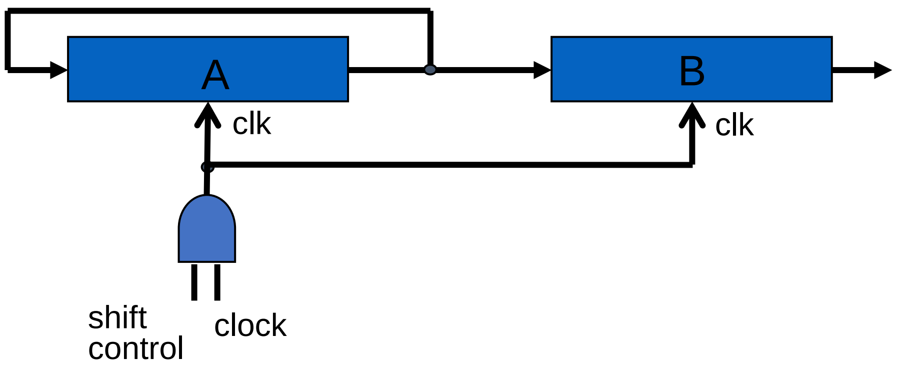
## Serial Transfer

- Suppose we have two 4-bit shift registers

B ← A

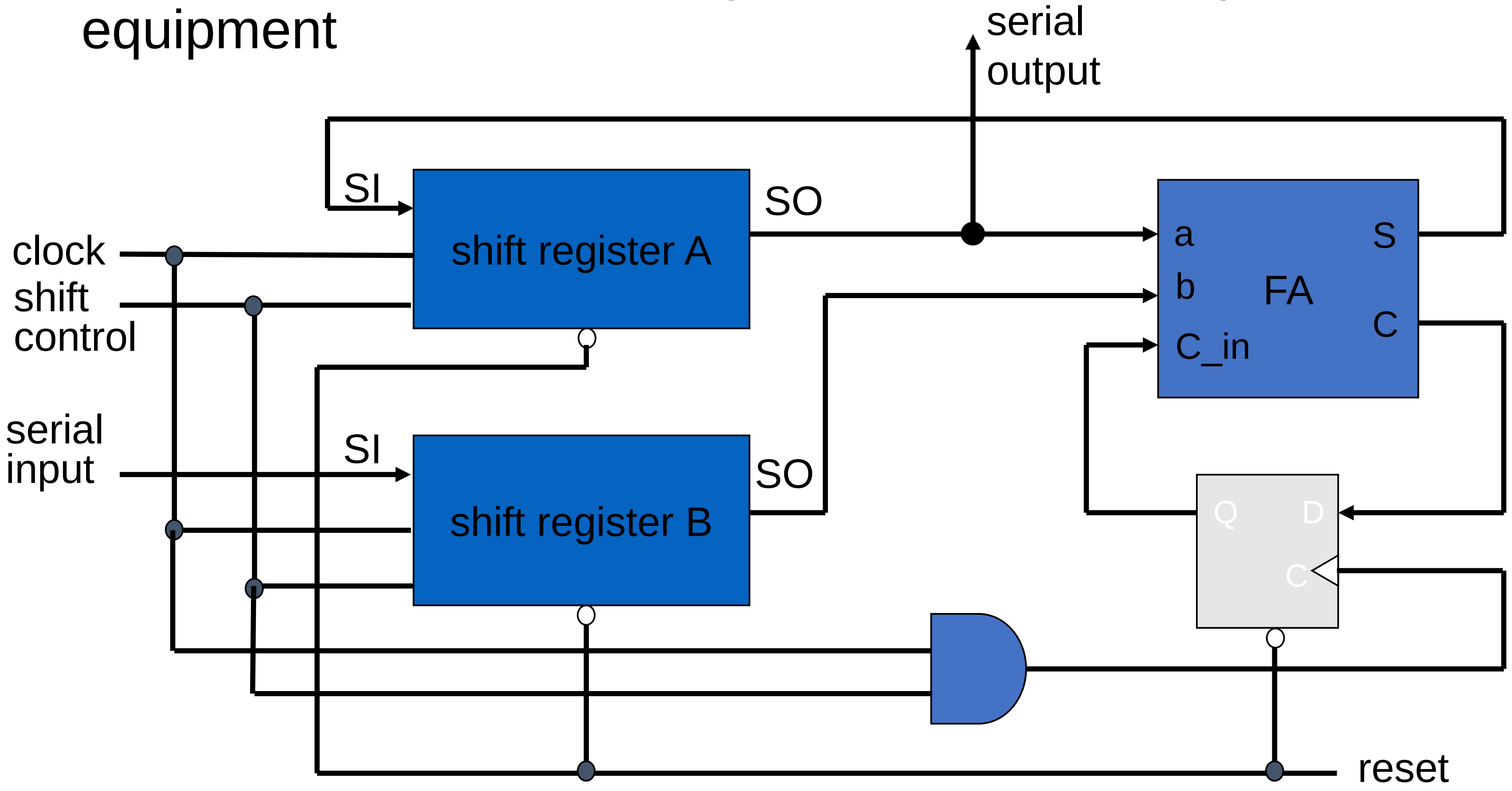
| Timing pulse | Shift register A | | | | Shift register B | | | |
|---|---|---|---|---|---|---|---|---|
| initial value | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| After $T_1$ | | | | | | | | |
| After $T_2$ | | | | | | | | |
| After $T_3$ | | | | | | | | |
| After $T_4$ | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |





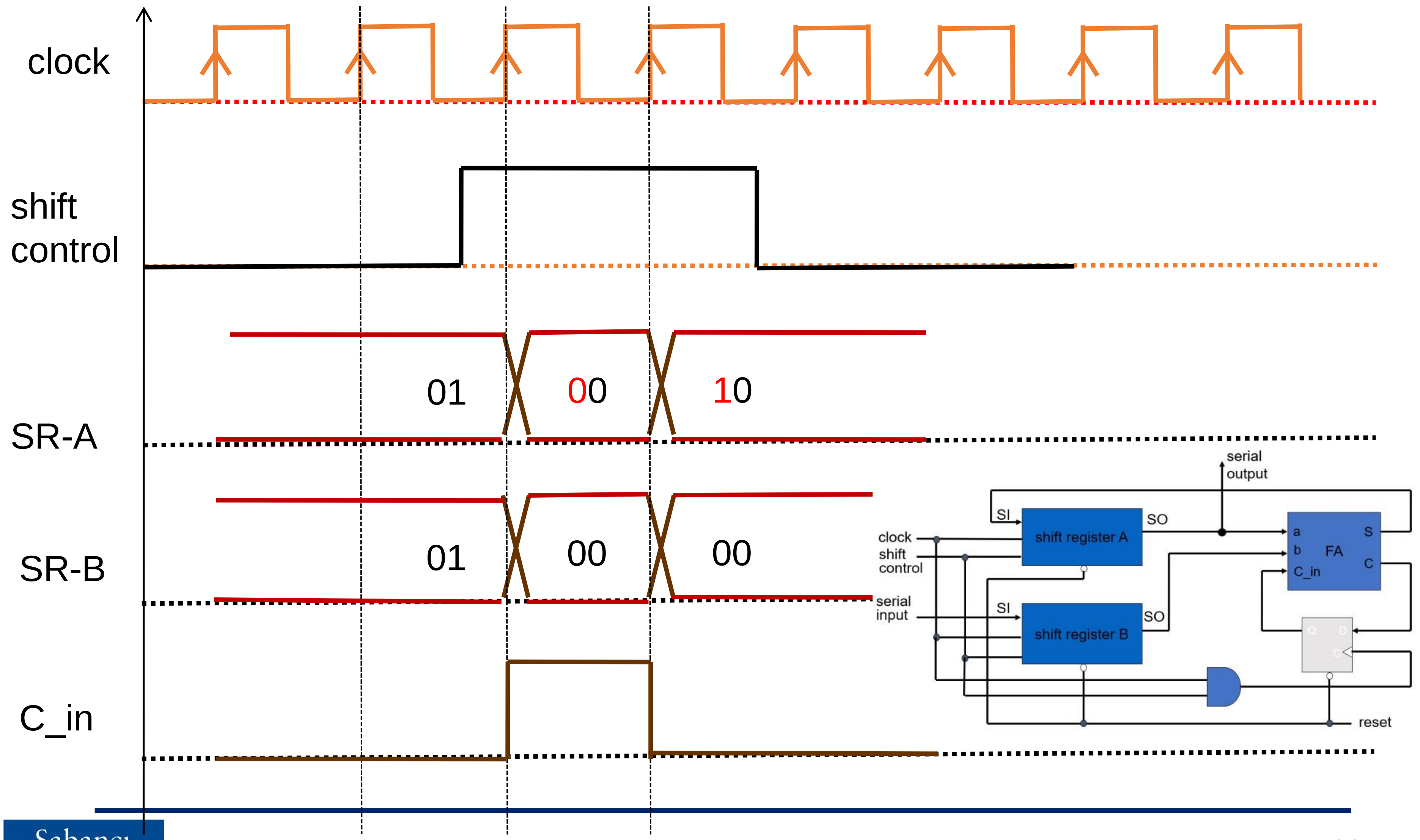
## Serial Addition

- In digital computers, operations are usually executed in parallel, since it is faster
- Serial mode is sometimes preferred since it requires less equipment

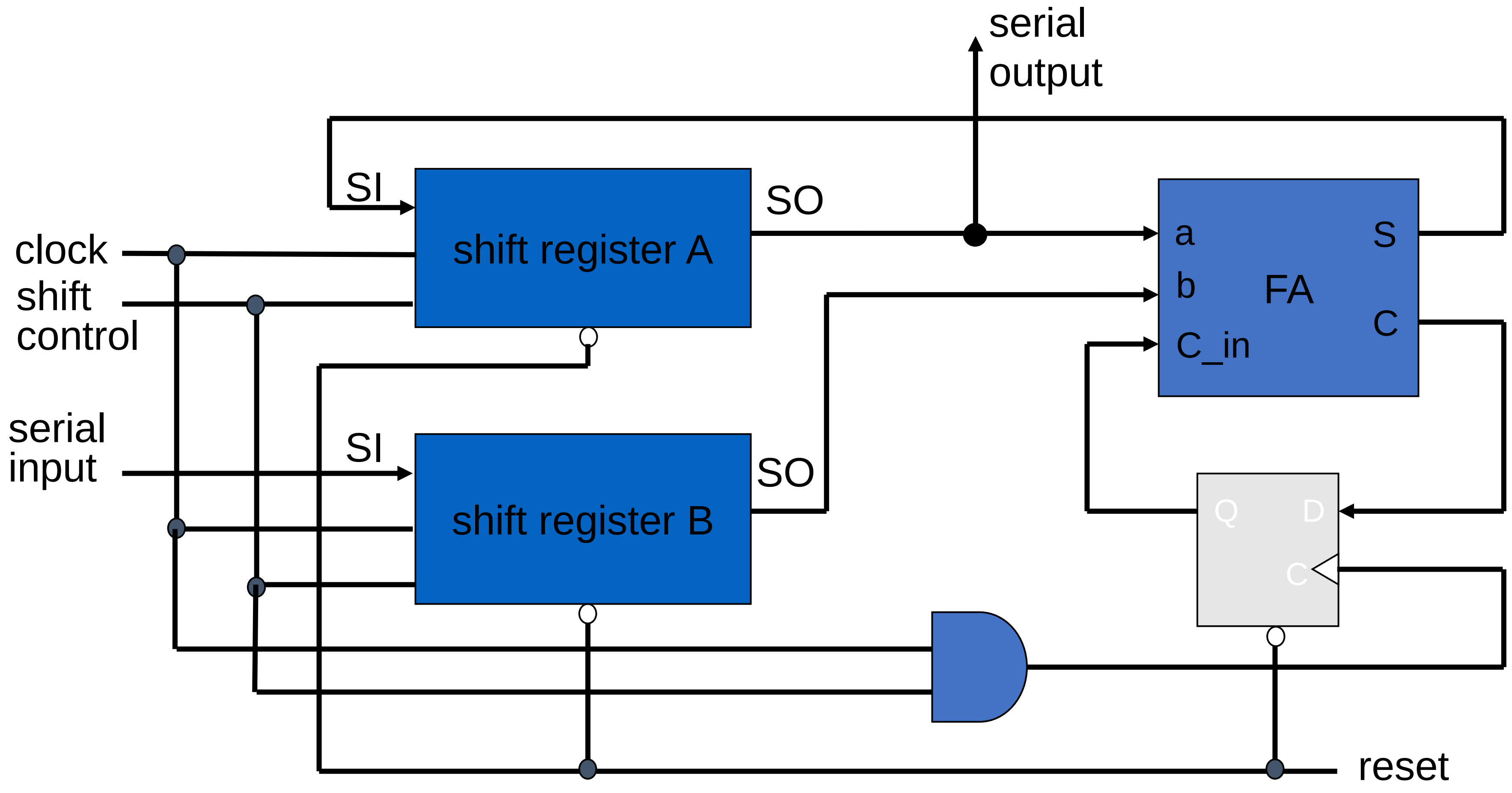


# Example: Serial Addition

- A and B are 2-bit shift registers

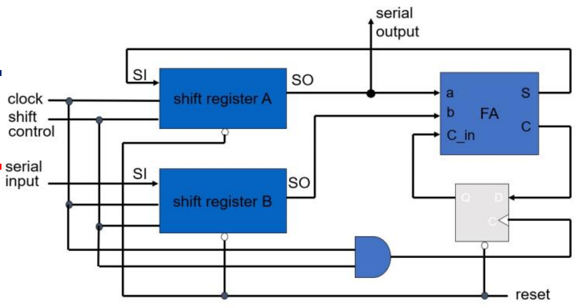


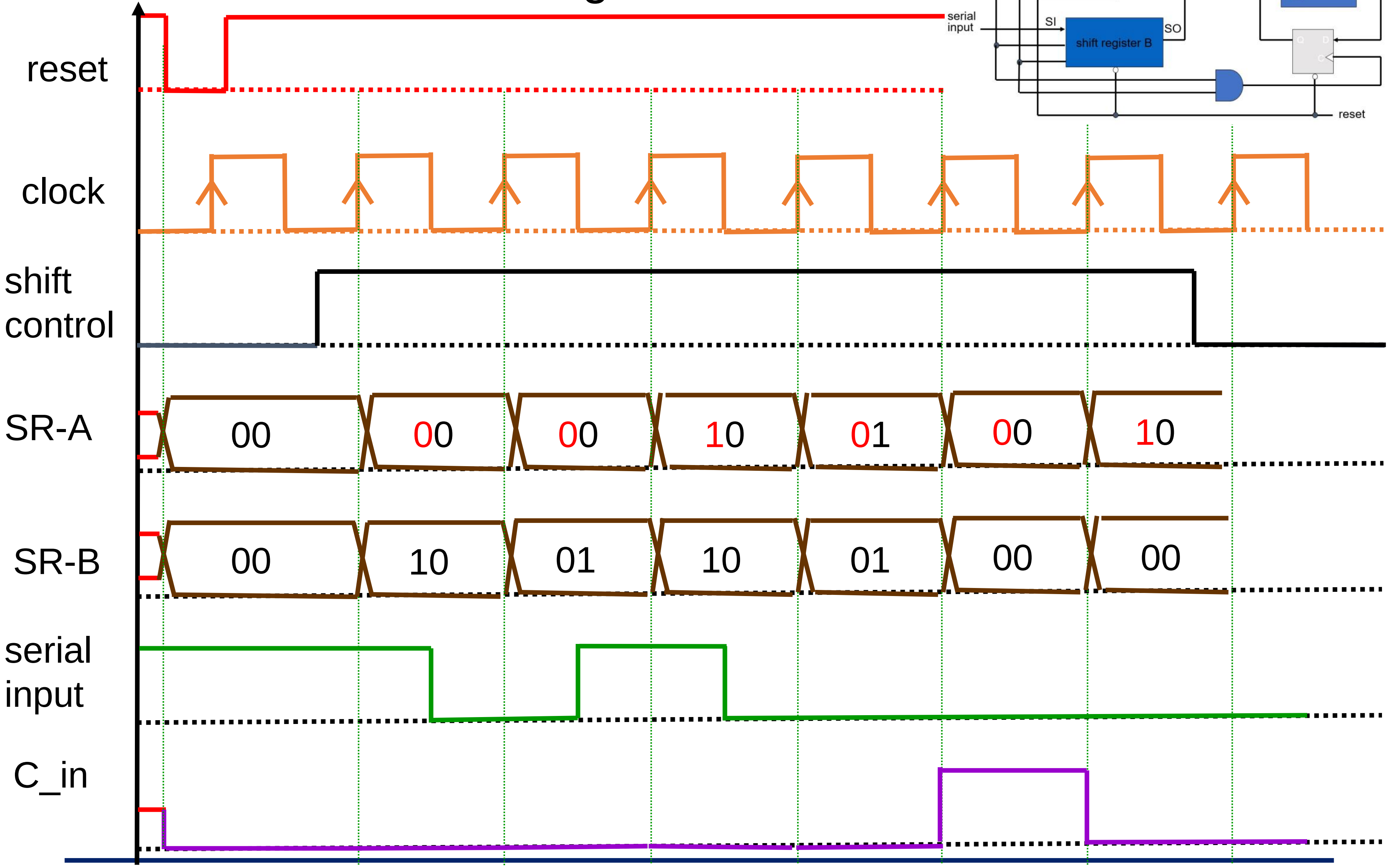
# How to Write Inputs to Registers?



# Example: Serial Addition

- A and B are 2-bit shift registers

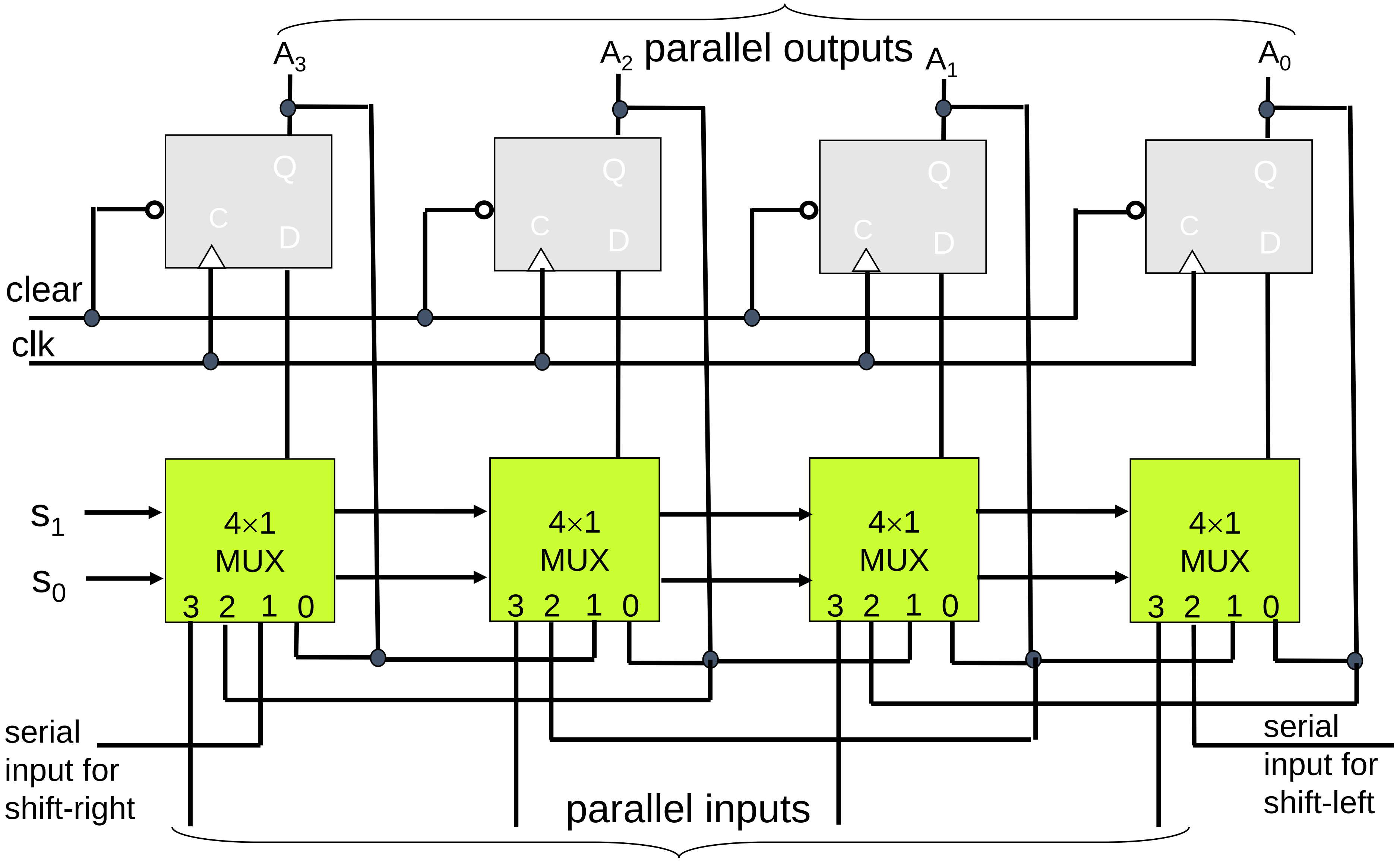




## Universal Shift Register

- Capabilities:
  1. A “clear” control to set the register to 0.
  2. A “clock” input
  3. A “shift-right” control
  4. A “shift-left” control
  5. n input lines & a “parallel-load” control
  6. n parallel output lines

# 4-Bit Universal Shift Register



## Universal Shift Register

| Mode Control | | Register operation |
|---|---|---|
| $s_1$ | $s_0$ | |
| 0 | 0 | No change |
| 0 | 1 | Shift right |
| 1 | 0 | Shift left |
| 1 | 1 | Parallel load |

# Coding Universal 32-bit Shift Register

```
module SR_32_BEH(
    output reg [31:0]   A_par,                   // Register output
    input      [31:0]   I_par,                   // Parallel input
    input               s1, s0,                  // selection inputs
                        MSB_in, LSB_in,          // Serial inputs
                        clk, clear);             // clock, reset
    always @(posedge clk, negedge clear)
        if(~clear) A_par <= 32'h00000000;
        else
            case ({s1,s0})
                2b'00: A_par <= A_par                  // no change
                2b'01: A_par <= {MSB_in, A_par[31:1]};      // shift right
                2b'10: A_par <= {A_par[30:0], LSB_in};      // shift left
                2b'11: A_par <= I_par;                 // parallel load
            endcase
endmodule
```

## Counters

- registers that go through a prescribed sequence of states upon the application of input pulses
  - input pulses are usually clock pulses
- Example: n-bit binary counter
  - count in binary from 0 to $2^n$-1
- Classification
  1. Synchronous counters
     - flip-flops receive the same common clock as *the* pulse
  2. Ripple counters
     - flip-flop output transition serves as *the* pulse to trigger other flip-flops

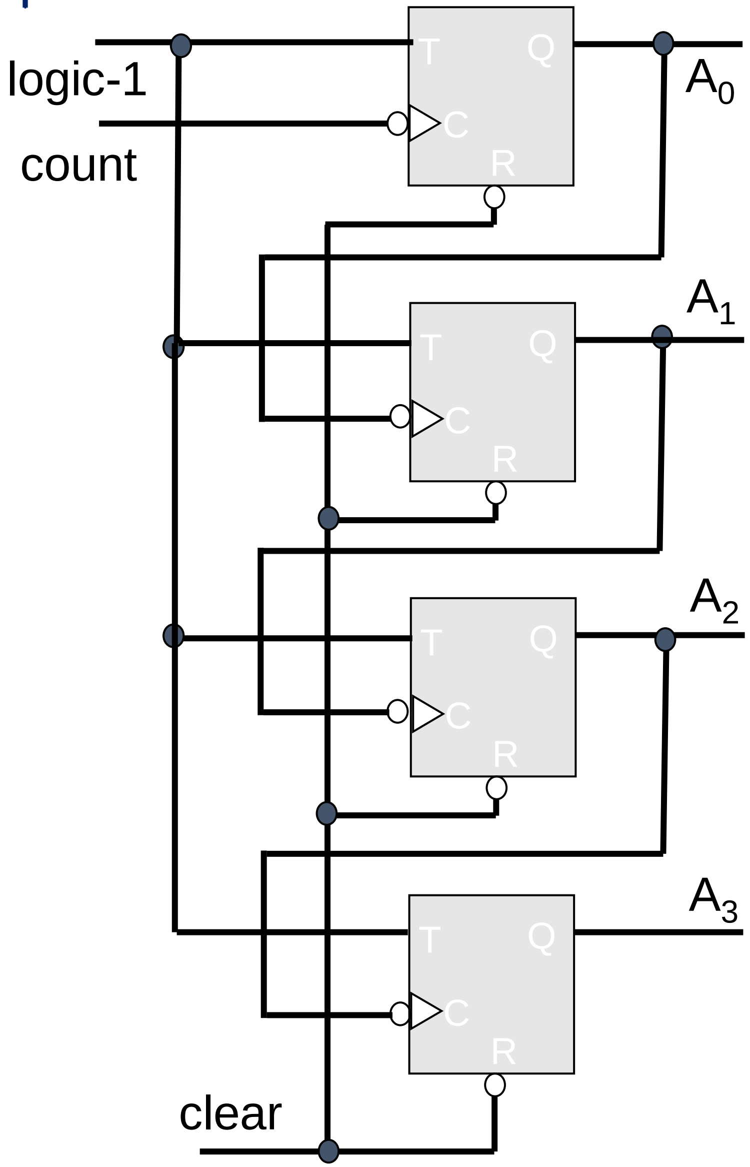
## Binary Ripple Counter

3-bit binary ripple counter

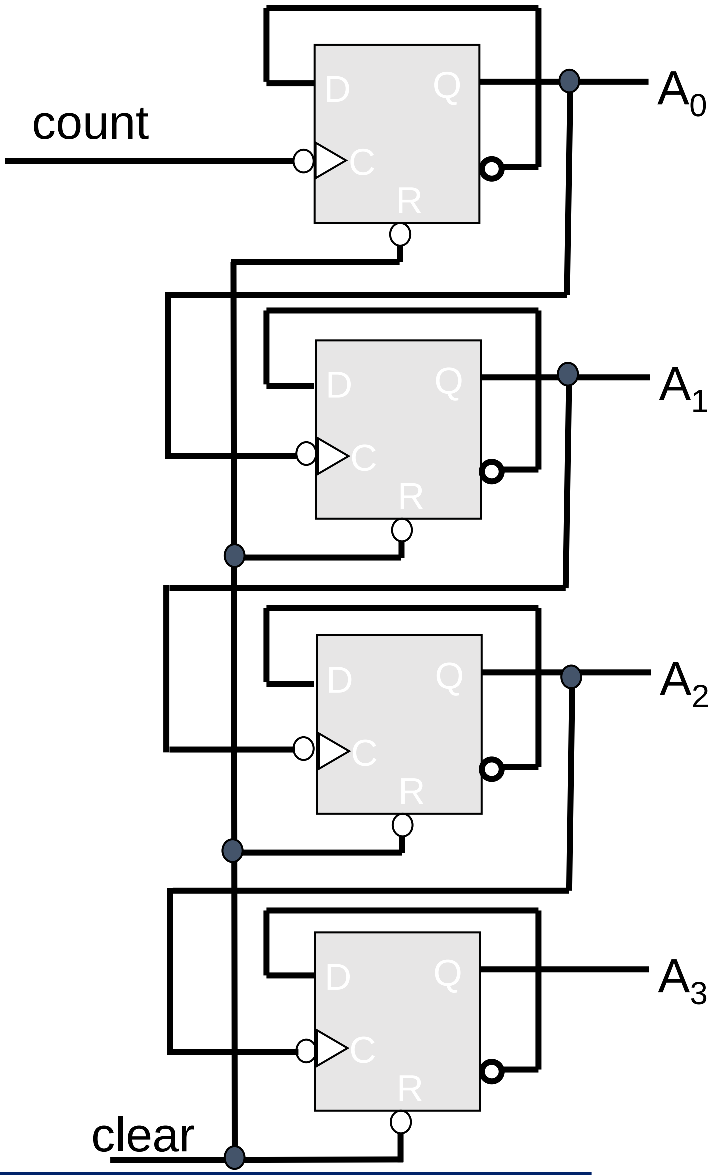
| | |
|---|---|
| 0 | 0 0 0 |
| 1 | 0 0 1 |
| 2 | 0 1 0 |
| 3 | 0 1 1 |
| 4 | 1 0 0 |
| 5 | 1 0 1 |
| 6 | 1 1 0 |
| 7 | 1 1 1 |
| 0 | 0 0 0 |

- Idea:
  - to connect the output of one flip-flop to the C input of the next high-order flip-flop
- We need “complementing” flip-flops
  - We can use T flip-flops to obtain complementing flip-flops or
  - JK flip-flops with its inputs are tied together or
  - D flip-flops with the complement output connected to the D input.

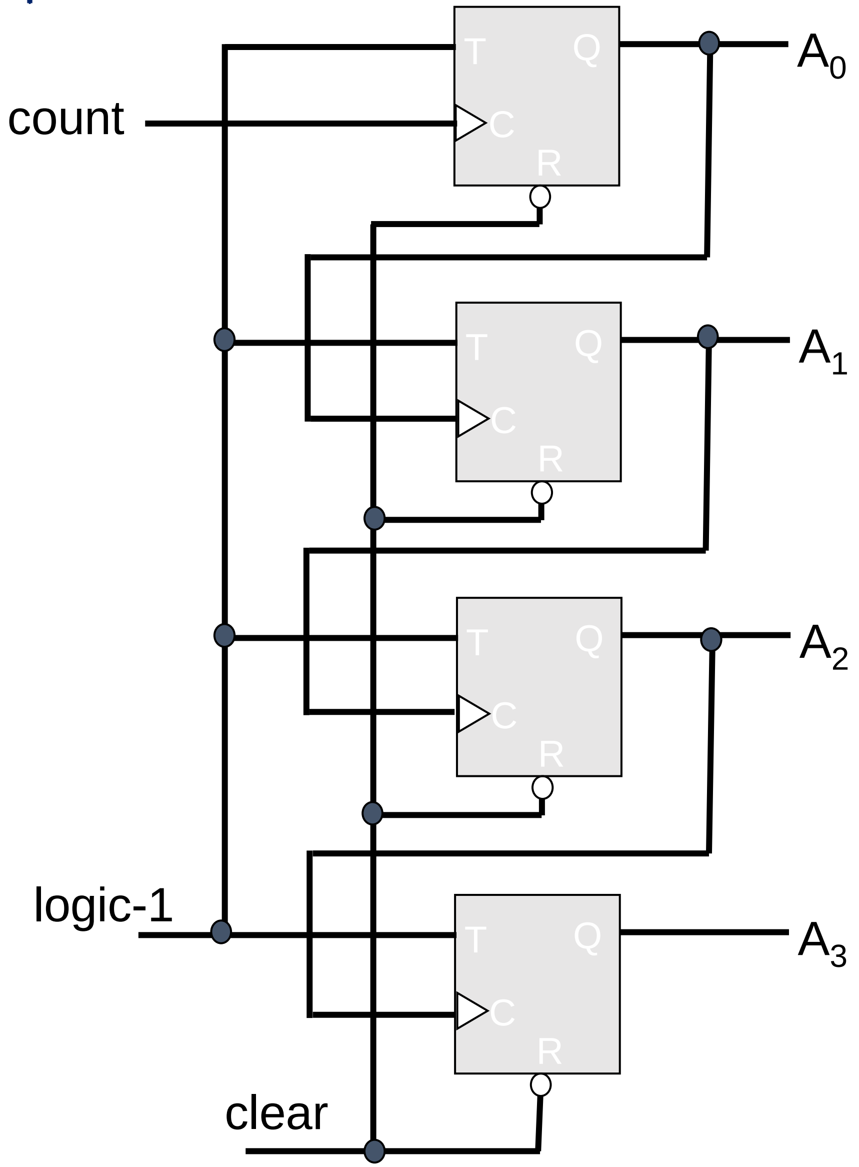
## 4-bit Binary Ripple Counter



| | |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| 10 | 1 0 1 0 |
| 11 | 1 0 1 1 |
| 12 | 1 1 0 0 |
| 13 | 1 1 0 1 |
| 14 | 1 1 1 0 |
| 15 | 1 1 1 1 |
| 0 | 0 0 0 0 |



# 4-bit Binary Ripple Counter



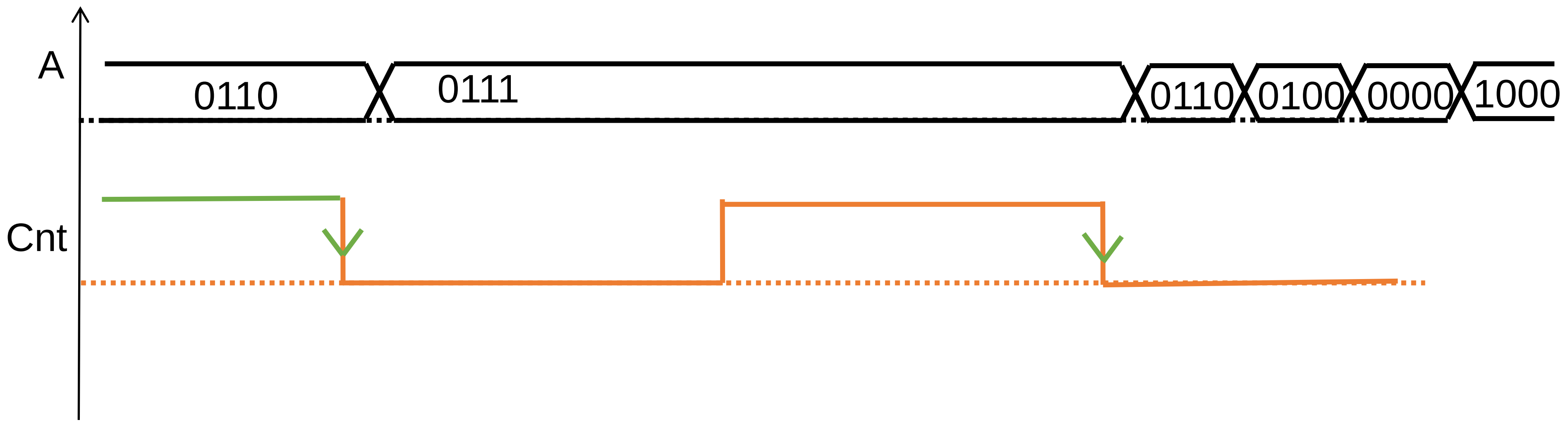
- Suppose the current state is **`1100`**
- What is the next state?

# Verilog of Binary Ripple Counter

```
`timescale 1ns / 1ps
module TFF(Q, T, clk, reset);
   input T,reset,clk;
   output reg Q;
   always @(negedge reset, negedge
   clk)
      if(reset) Q <= 1'b0;
      else  Q <= #1 T^Q;
endmodule
module RippleCounter(
   output  [3:0] A,
   input Count, reset);
   TFF FF0(A[0], 1'b1, Count, reset);
   TFF FF1(A[1], 1'b1, A[0], reset);
   TFF FF2(A[2], 1'b1, A[1], reset);
   TFF FF3(A[3], 1'b1, A[2], reset);
endmodule
```

```
module TestRippleCounter;
   reg Cnt;
   reg Rst;
   wire [3:0] A;
    // Instantiate ripple counter
    RippleCounter Counter(A, Cnt, Rst);
  always
     #5  Cnt = ~Cnt;
   initial
      begin
         Cnt = 1'b0;
         Rst = 1'b0;
         #4 Rst = 1'b1;
      end
   initial #170 $finish;
endmodule
```

## Simulation

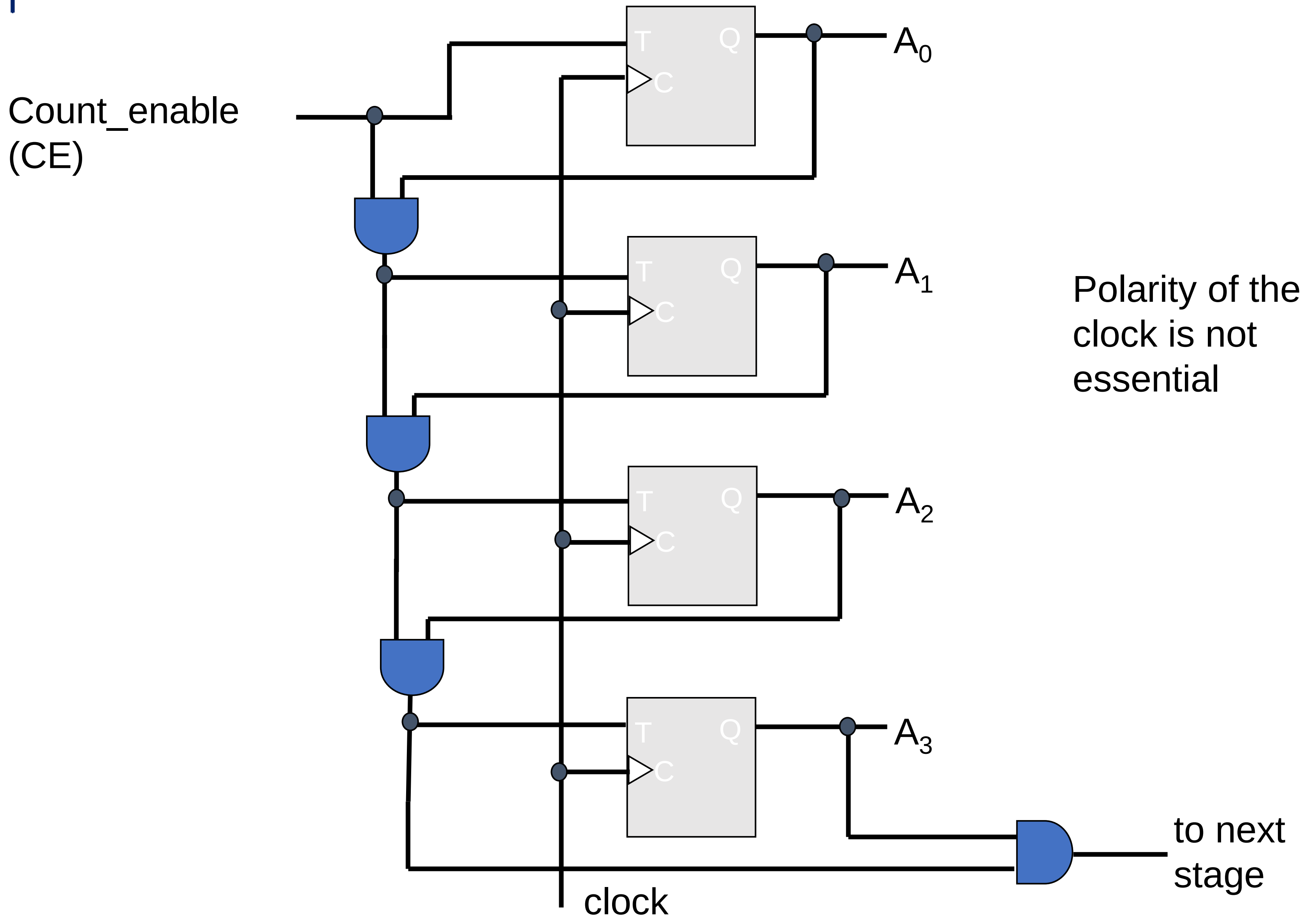


## Synchronous Counters

- There is a common clock
  - that triggers all flip-flops simultaneously
  - If T = 0 or J = K = 0 the flip-flop does not change state.
  - If T = 1 or J = K = 1 the flip-flop does change state.
- Design procedure is so simple
  - no need for going through sequential logic design process
  - $A_0$ is always complemented
  - $A_1$ is complemented when $A_0 = 1$
  - $A_2$ is complemented when $A_0 = 1$ and $A_1 = 1$
  - so on

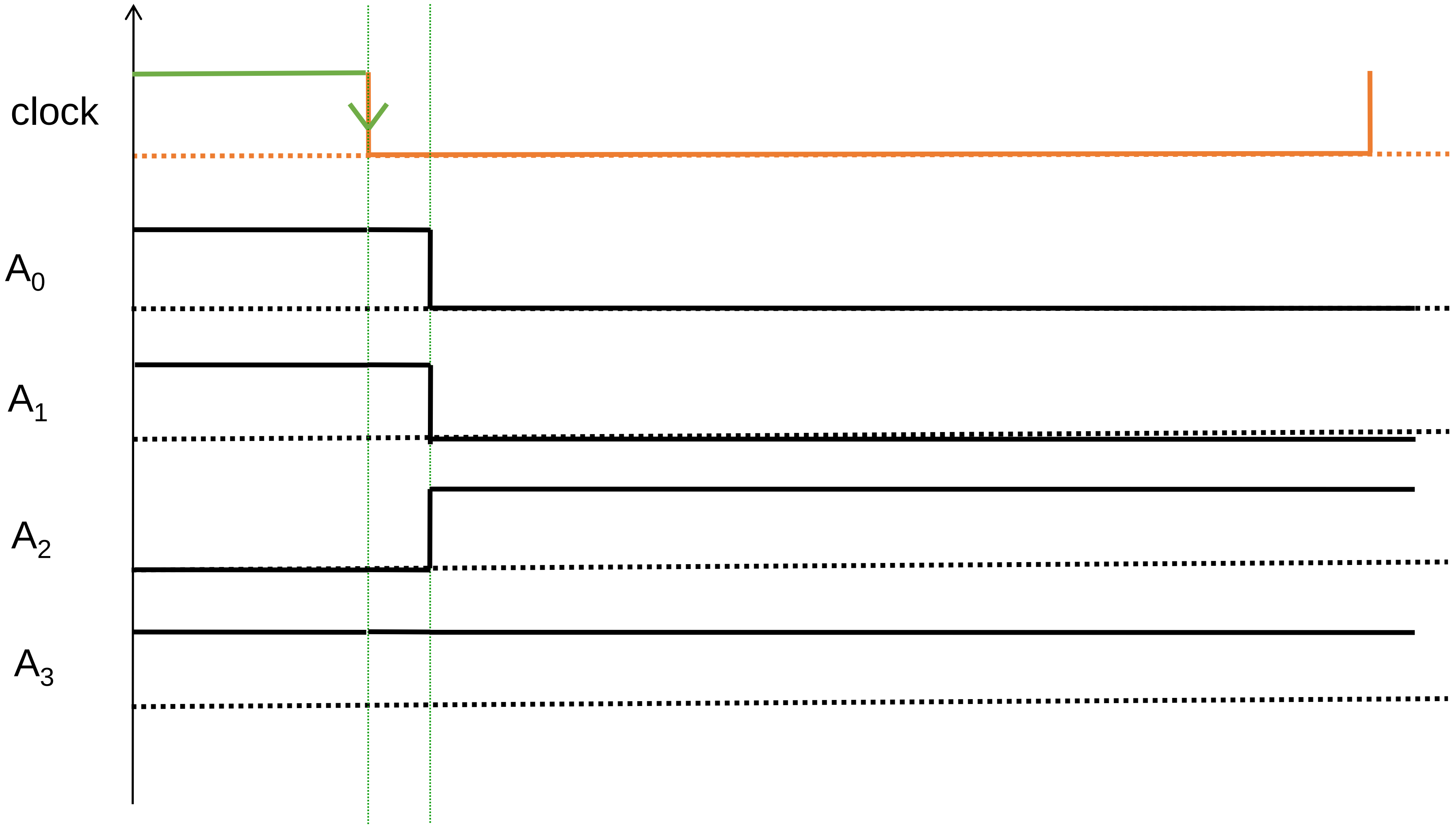
| | |
|---|---|
| 0 | 0 0 0 |
| 1 | 0 0 1 |
| 2 | 0 1 0 |
| 3 | 0 1 1 |
| 4 | 1 0 0 |
| 5 | 1 0 1 |
| 6 | 1 1 0 |
| 7 | 1 1 1 |
| 0 | 0 0 0 |

# 4-bit Binary Synchronous Counter

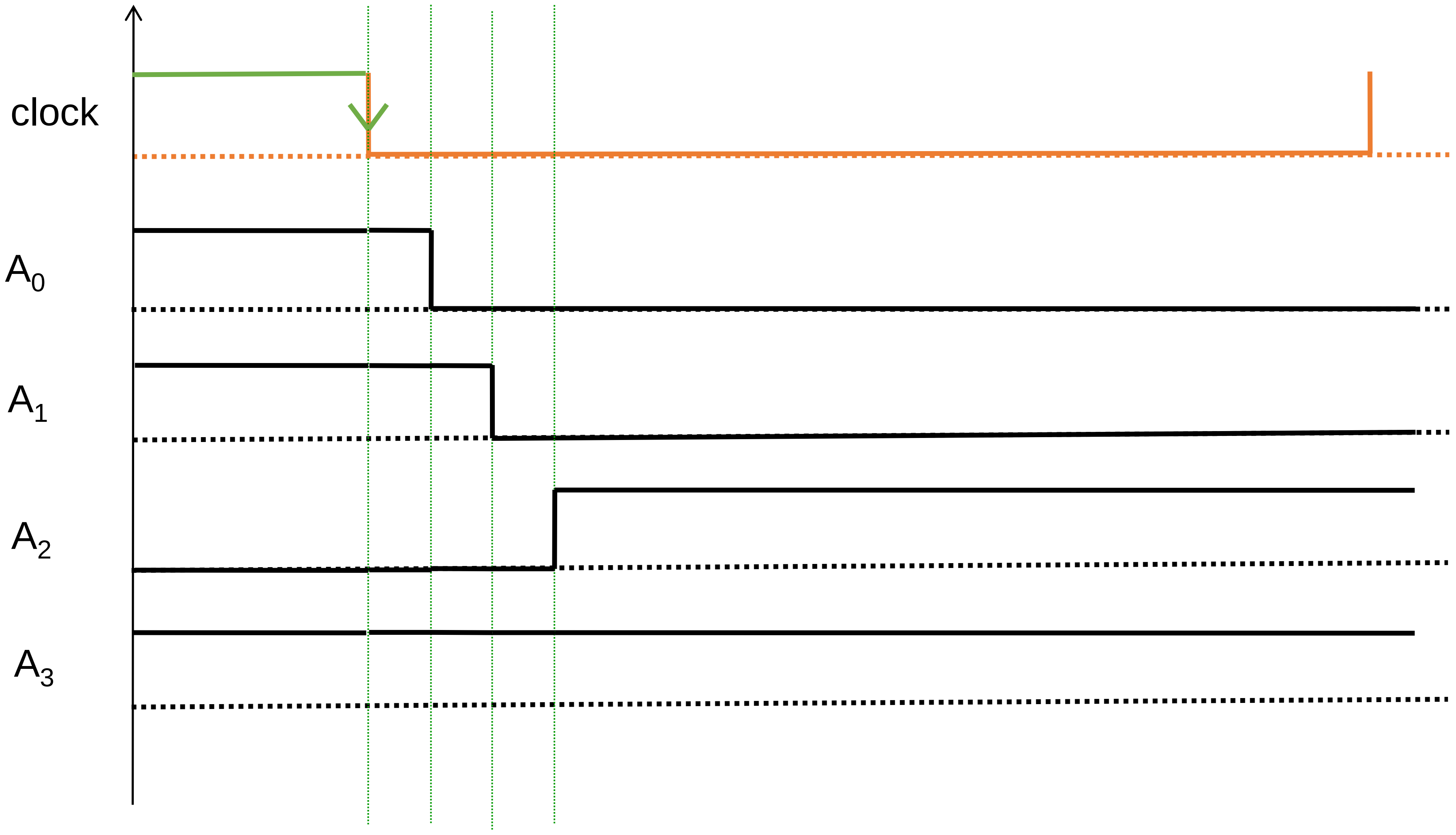


Polarity of the clock is not essential

## Timing of Synchronous Counters



## Timing of Ripple Counters

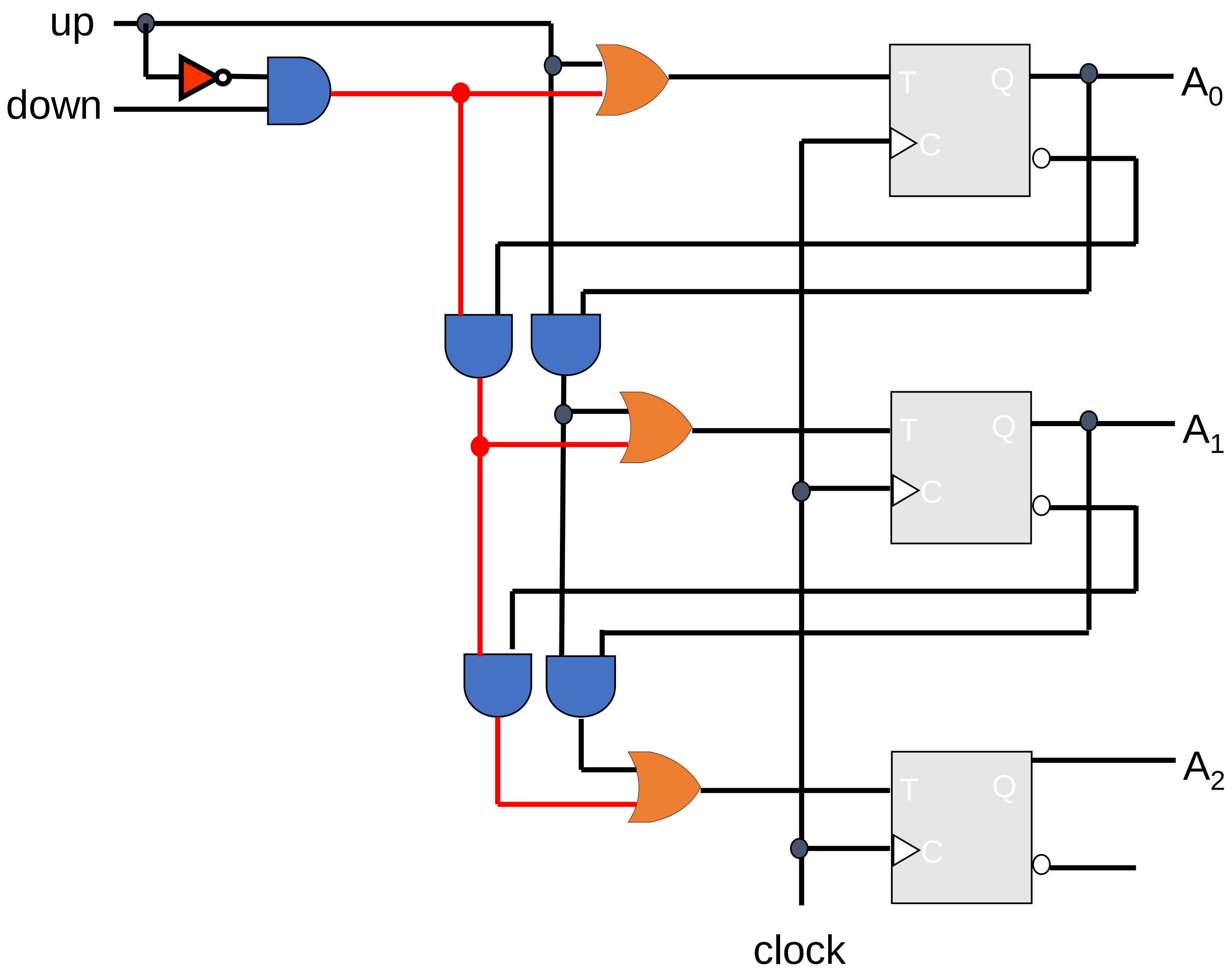


## Up-Down Binary Counter

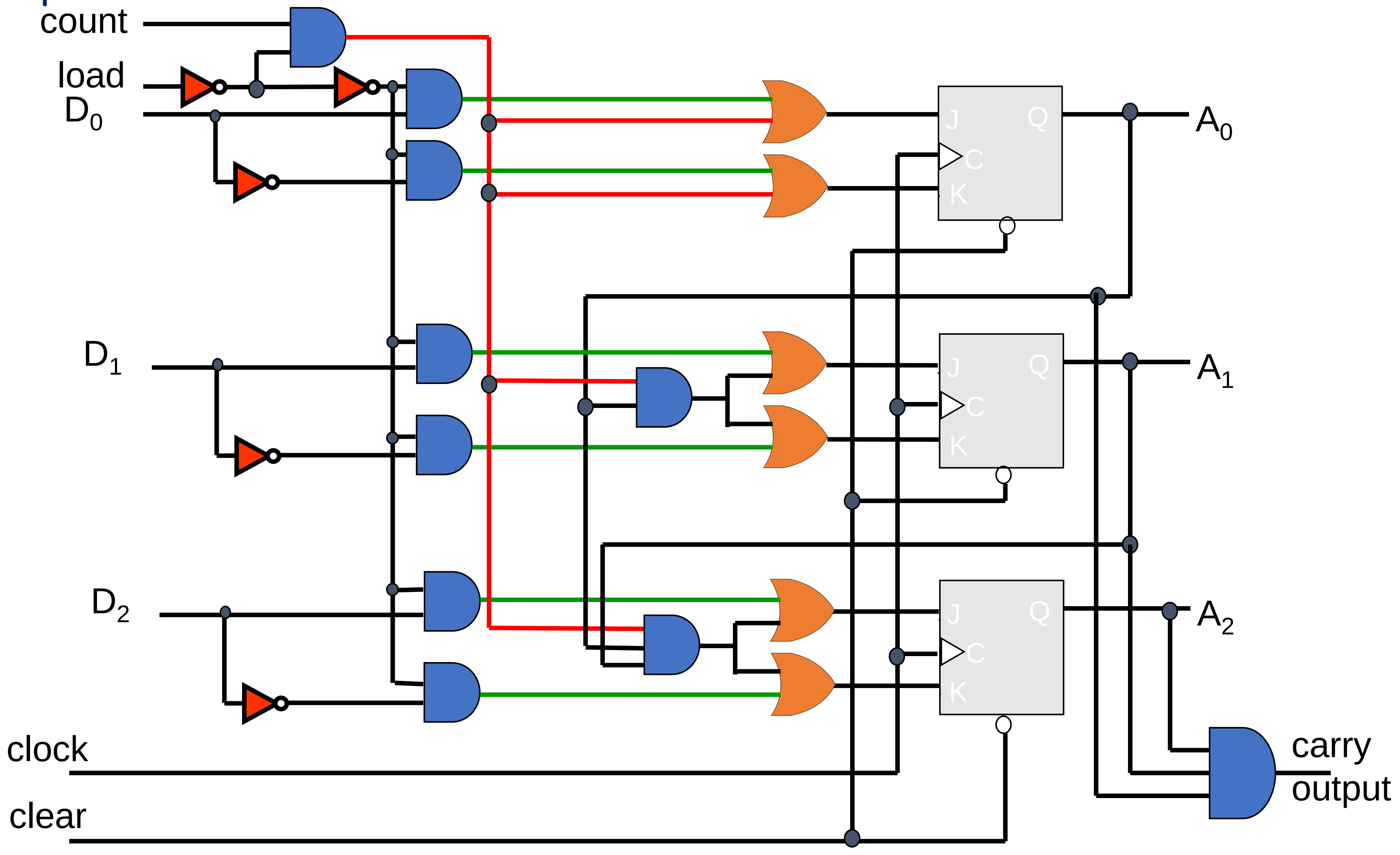
- When counting downward
  - the least significant bit is always complemented (with each clock pulse)
  - A bit in any other position is complemented if all lower significant bits are equal to 0.
  - For example: `0 1 0 0`
    - Next state:
  - For example: `1 1 0 0`
    - Next state:

| | |
|---|---|
| 0 | 0 0 0 |
| 7 | 1 1 1 |
| 6 | 1 1 0 |
| 5 | 1 0 1 |
| 4 | 1 0 0 |
| 3 | 0 1 1 |
| 2 | 0 1 0 |
| 1 | 0 0 1 |
| 0 | 0 0 0 |

# Up-Down Binary Counter



# Binary Counter with Parallel Load



## Binary Counter with Parallel Load

Function Table

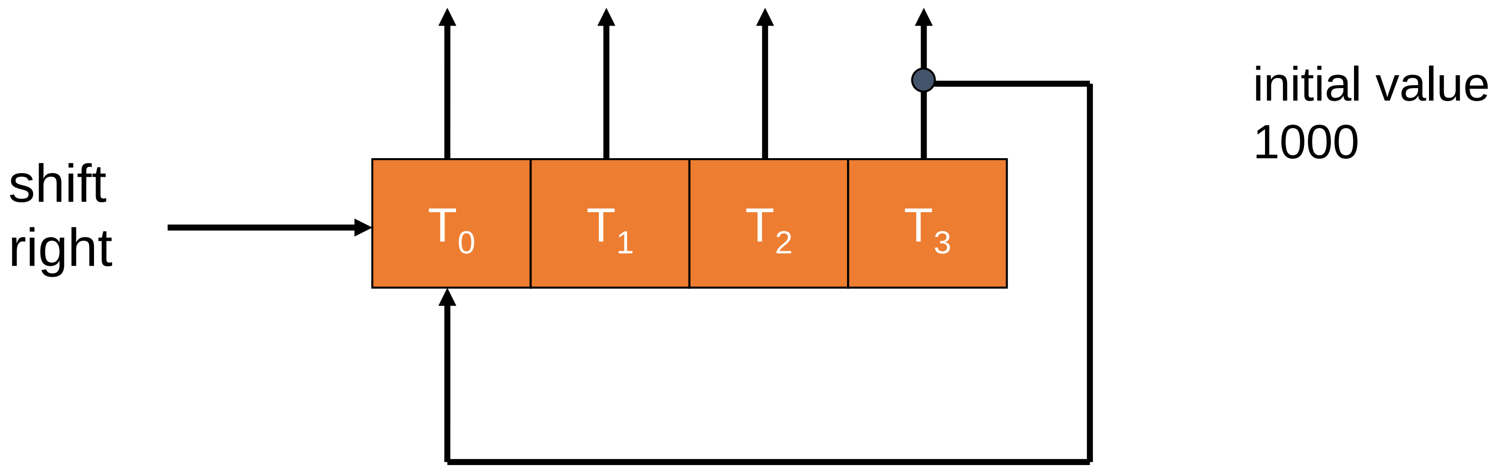
| clear | clock | load | Count | Function |
|---|---|---|---|---|
| 0 | X | X | X | clear to 0 |
| 1 | ↑ | 1 | X | load inputs |
| 1 | ↑ | 0 | 1 | count up |
| 1 | ↑ | 0 | 0 | no change |

## Verilog of Binary Counter

```
module BinaryCounter_8_BEH(
    output reg [7:0]     A_cnt,                   // Counter output
    output                          C_out,                   // If a cycle is
  completed
    input       [7:0]     Data_in,                 // Parallel input
    input                 Count,                   // Active high to count
                                Load,              // Active high to load
                          clk, clear);             // clock, reset
    assign C_out = Count & (~Load) & (A_cnt == 8'hFF);
    always @(posedge clk, negedge clear)
        if(~clear) A_cnt <= 8'h00;
        else if(Load) A_cnt <= Data_in;
        else if(Count) A_cnt <= A_cnt + 1'b1;
        else A_cnt <= A_cnt;
endmodule
```

## Other Counters

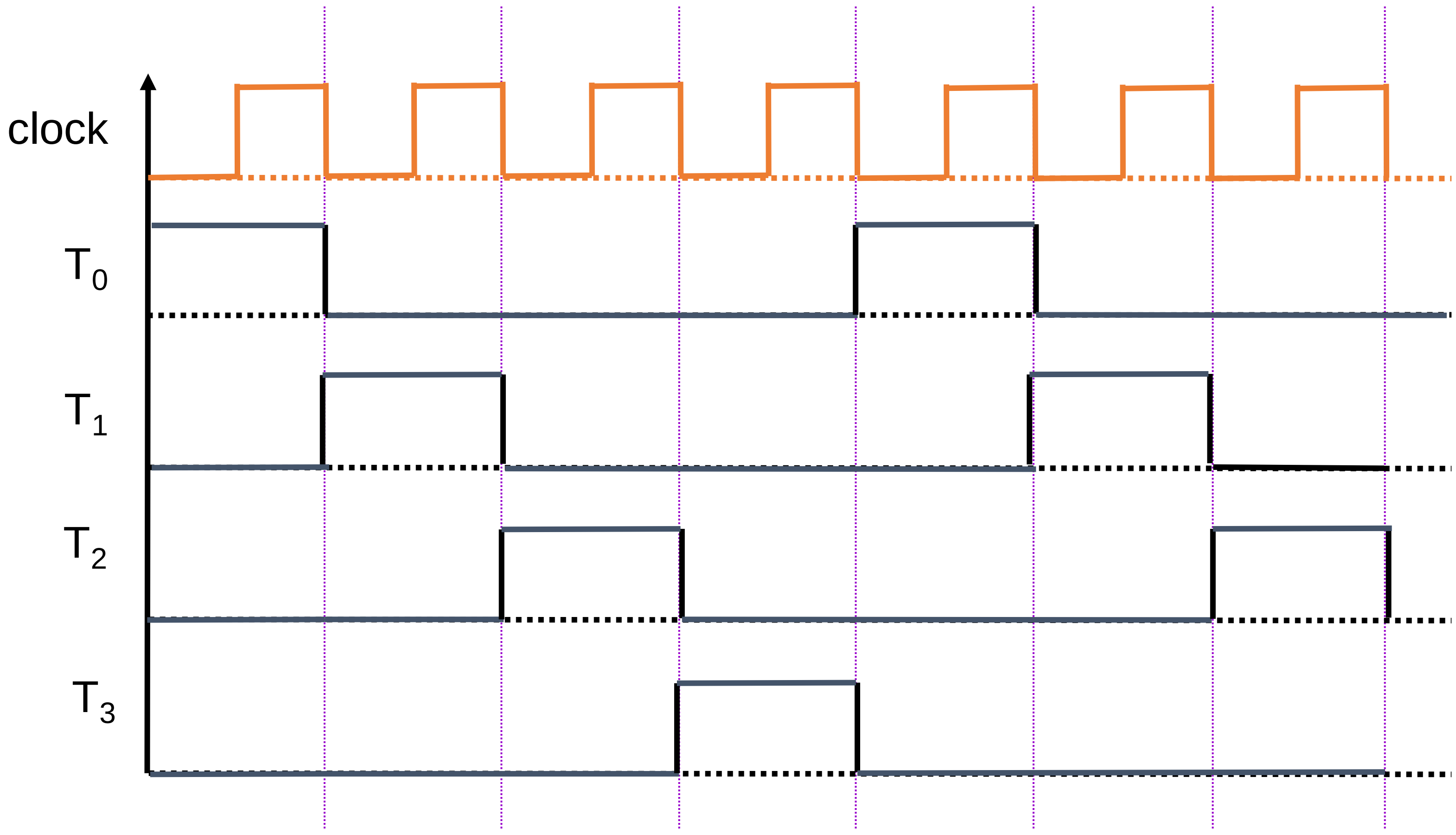
- Ring Counter
  - A ring counter is a circular shift register with only one flip-flop being set at any particular time, all others are cleared.



- Usage
  - Timing signals control the sequence of operations in a digital system

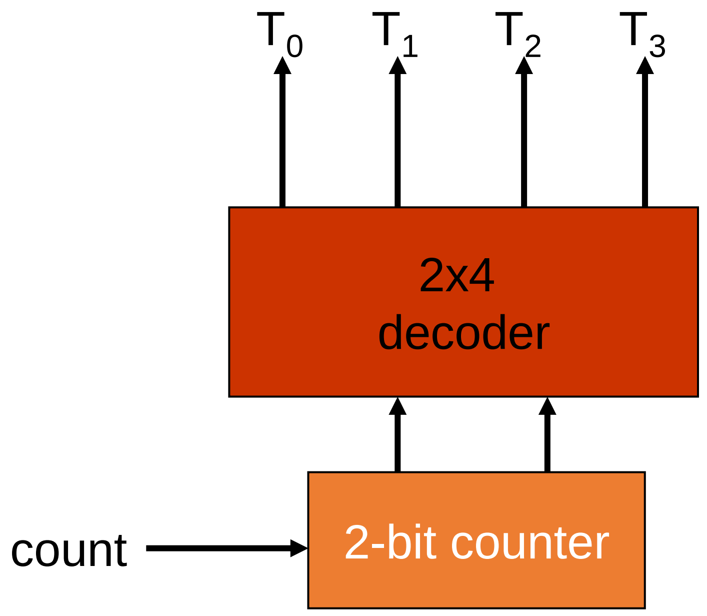
# Ring Counter

- Sequence of timing signals



# Ring Counter

- To generate $2^n$ timing signals,
  - we need a shift register with $2^n$ flip-flops
- or, we can construct the ring counter with a binary counter and a decoder

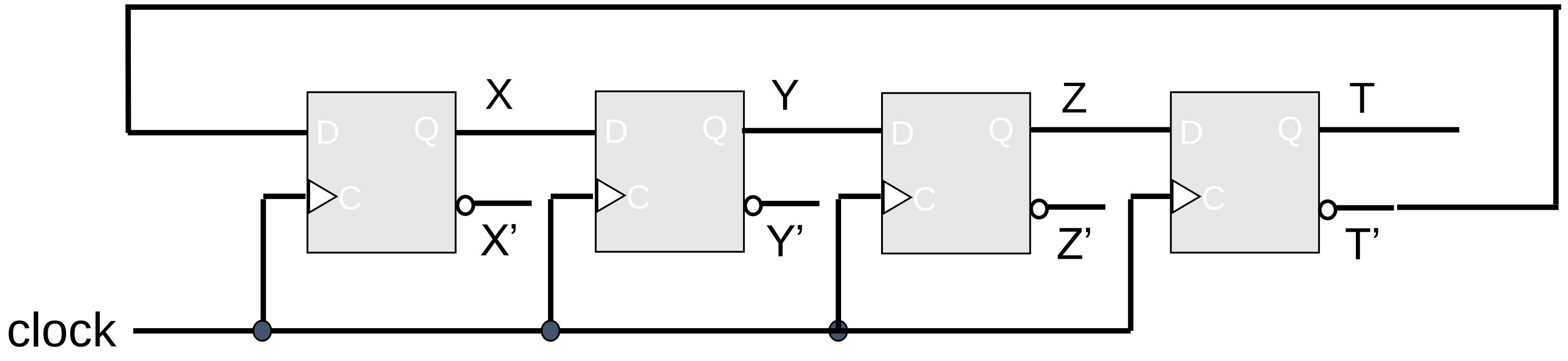


Cost:
- 2 flip-flops
- 2-to-4 line decoder

Cost in general case:
- n flip-flops
- n-to-$2^n$ line decoder
  - $2^n$ n-input AND gates
  - n NOT gates

## Johnson Counter

- A k-bit ring counter can generate k distinguishable states
- The number of states can be doubled if the shift register is connected as a switch-tail ring counter



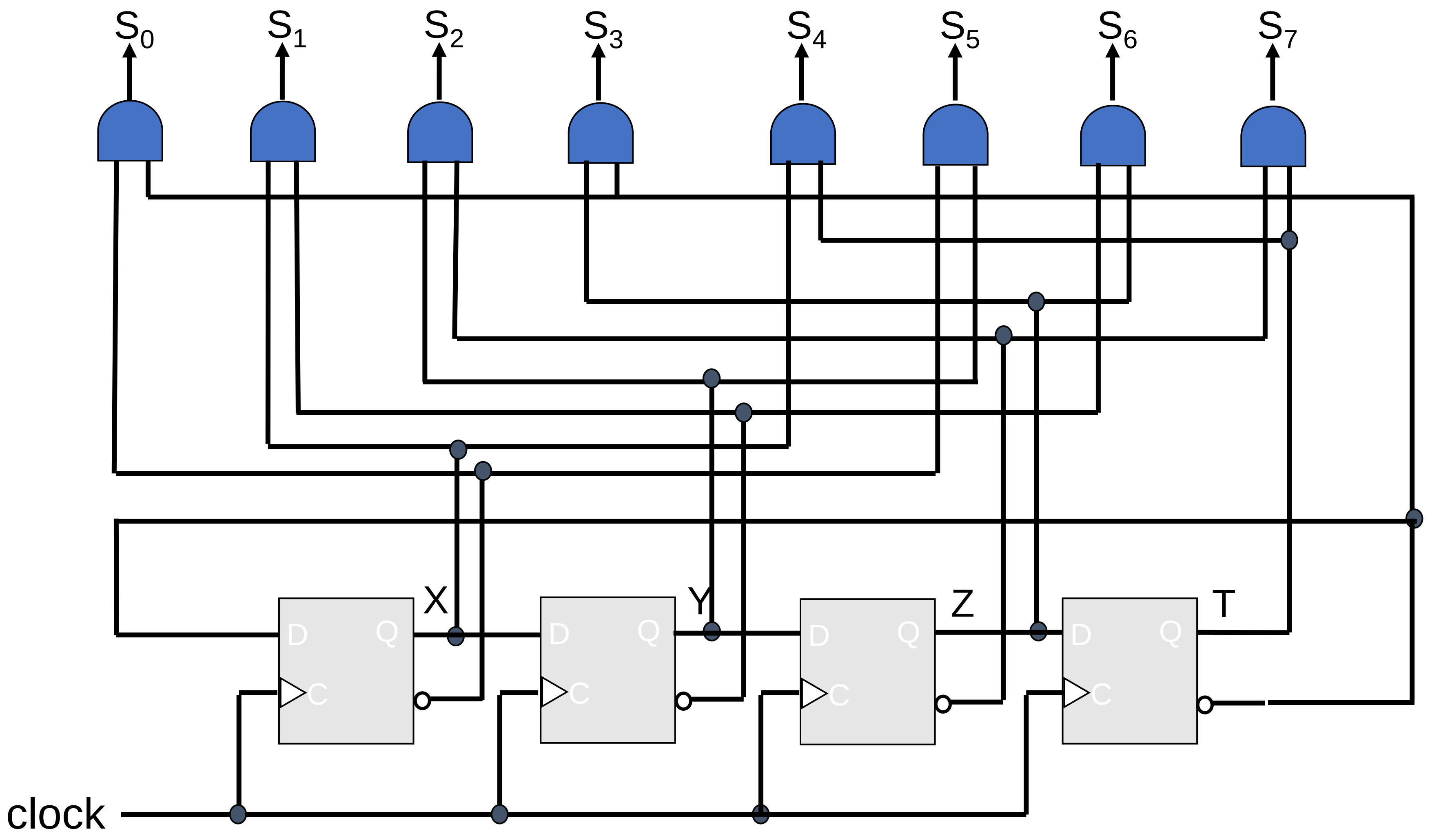
## Johnson Counter

- Count sequence and required decoding

| sequence number | Flip-flop outputs | | | | Output |
|---|---|---|---|---|---|
| | X | Y | Z | T | |
| 1 | 0 | 0 | 0 | 0 | $S_0$ = X'T' |
| 2 | | | | | $S_1$ = XY' |
| 3 | | | | | $S_2$ = YZ' |
| 4 | | | | | $S_3$ = ZT' |
| 5 | | | | | $S_4$ = XT |
| 6 | | | | | $S_5$ = X'Y |
| 7 | | | | | $S_6$ = Y'Z |
| 8 | | | | | $S_7$ = Z'T |

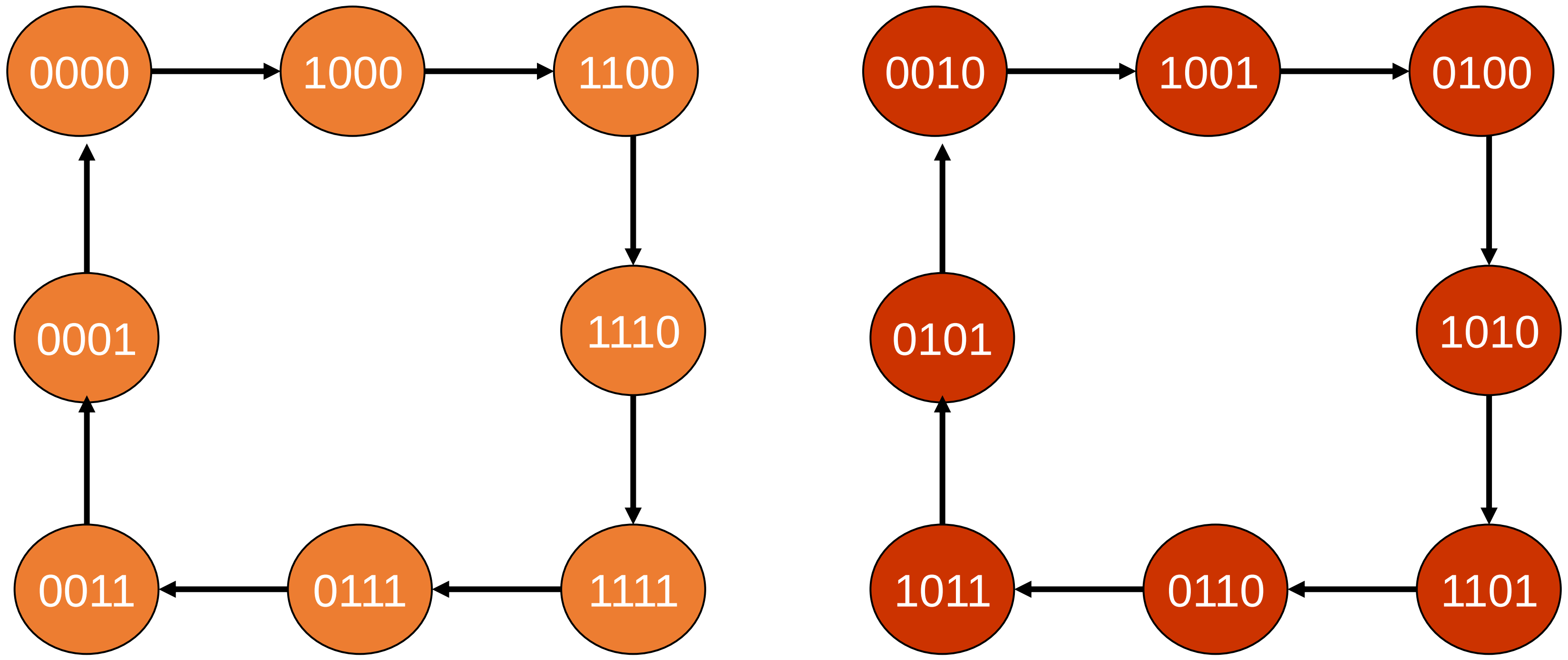
# Johnson Counter

- Decoding circuit



# Unused States in Counters

- 4-bit Johnson counter



## Correction

$$\begin{tikzcd}
0000 \arrow[r] & 1000 \arrow[r] & 1100 \arrow[d] & 0010 \arrow[r] & 1001 \arrow[r] \arrow[llll, bend right] & 0100 \arrow[d] \\
0001 \arrow[u] & & 1110 \arrow[d] & 0101 \arrow[u] & & 1010 \arrow[d] \\
0011 \arrow[u] & 0111 \arrow[l] & 1111 \arrow[l] & 1011 \arrow[u] & 0110 \arrow[l] & 1101 \arrow[l]
\end{tikzcd}$$

## Johnson Counter

| Present State | | | | Next State | | | |
|---|---|---|---|---|---|---|---|
| X | Y | Z | T | X | Y | Z | T |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

# K-Maps

| XY \ ZT | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | | 1 |
| 01 | 1 | | | 1 |
| 11 | 1 | | | 1 |
| 10 | 1 | | | 1 |

X(t+1) = T'

| XY \ ZT | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 0 | 1 | 1 |

Y(t+1) = XY + XZ + XT'

| XY \ ZT | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | | | | |

Z(t+1) = Y

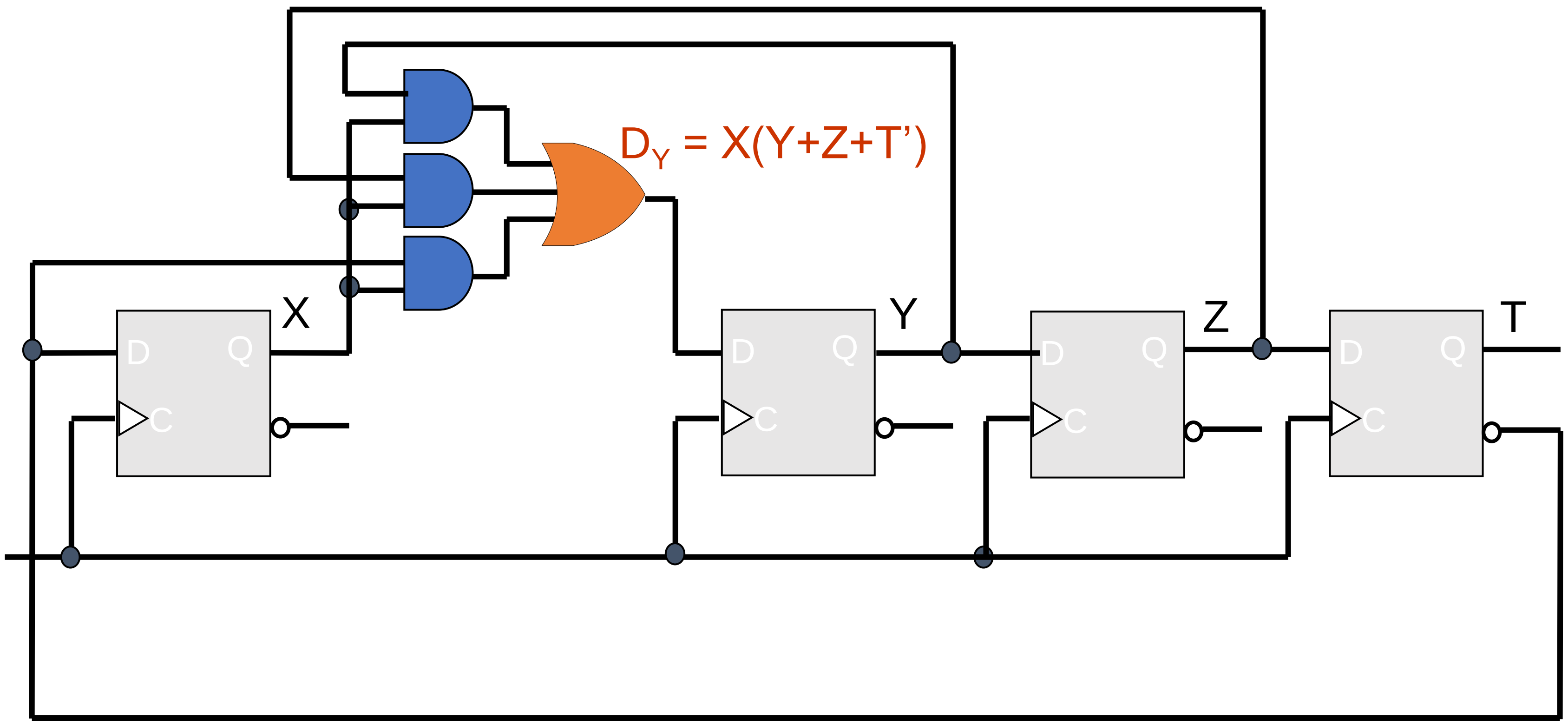
| XY \ ZT | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | 1 |
| 01 | | | 1 | 1 |
| 11 | | | 1 | 1 |
| 10 | | | 1 | 1 |

T(t+1) = Z

## Unused States in Counters

- Remedy $X(t+1) = T'$ $Y(t+1) = XY + XZ + XT'$

  $Z(t+1) = Y$ $T(t+1) = Z$



$D_Y = X(Y+Z+T')$
X
Y
Z
T
D
Q
C
clock