university of groningen

# Computer Architecture 2023-24 (WBCS010-05)

## Lecture 4: Sequential Logic

Reza Hassanpour
r.zare.hassanpour@rug.nl

# Introduction

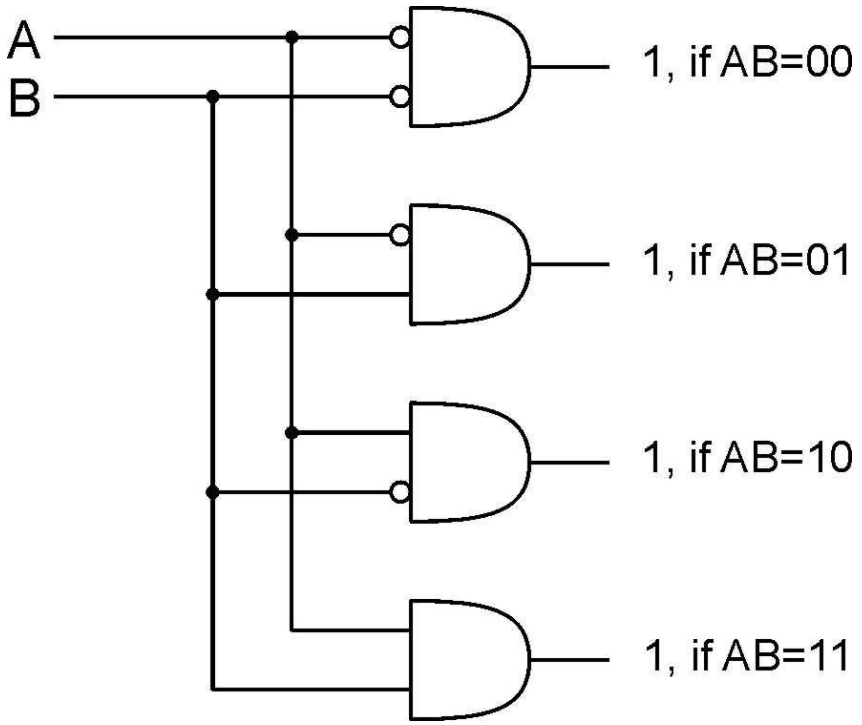
- The output from combinational logic circuits depends only on their inputs.
- Digital computers need to store data/state, and decide the output based on inputs and stored data/state.
- Sequential logic circuits the output depends upon present as well as past input.

- **Q:** How can we store the results we received in the past?

university of groningen

# Decoder (recap)

- A decoder has $n$ inputs and $2^n$ outputs
- Each output corresponds to one possible input combination
- Exactly one output will be 1, and all others will be 0



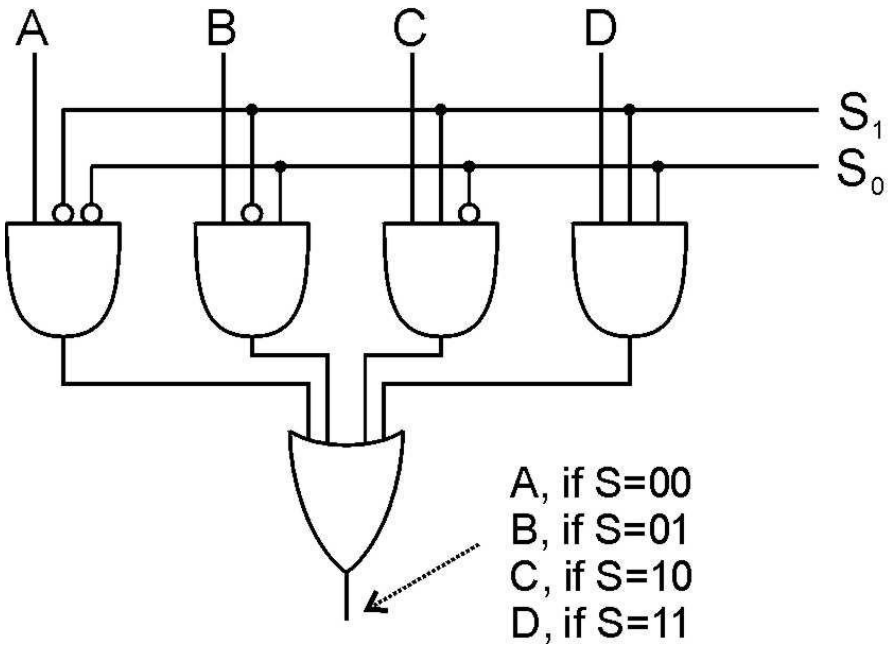

**2-bit decoder**

*Useful for converting encoded values (integers) into a set of control signals*

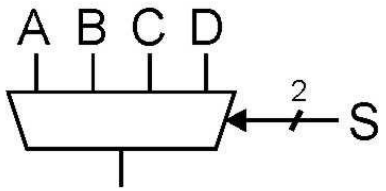
university of groningen

# Multiplexer (Mux) (recap)

- A mux has $2^n$ data inputs, $n$ select inputs, and one output
- The select bits are used to "choose" one of the data inputs to flow through to the output




**4-to-1 mux**
4 input bits,
2 select bits

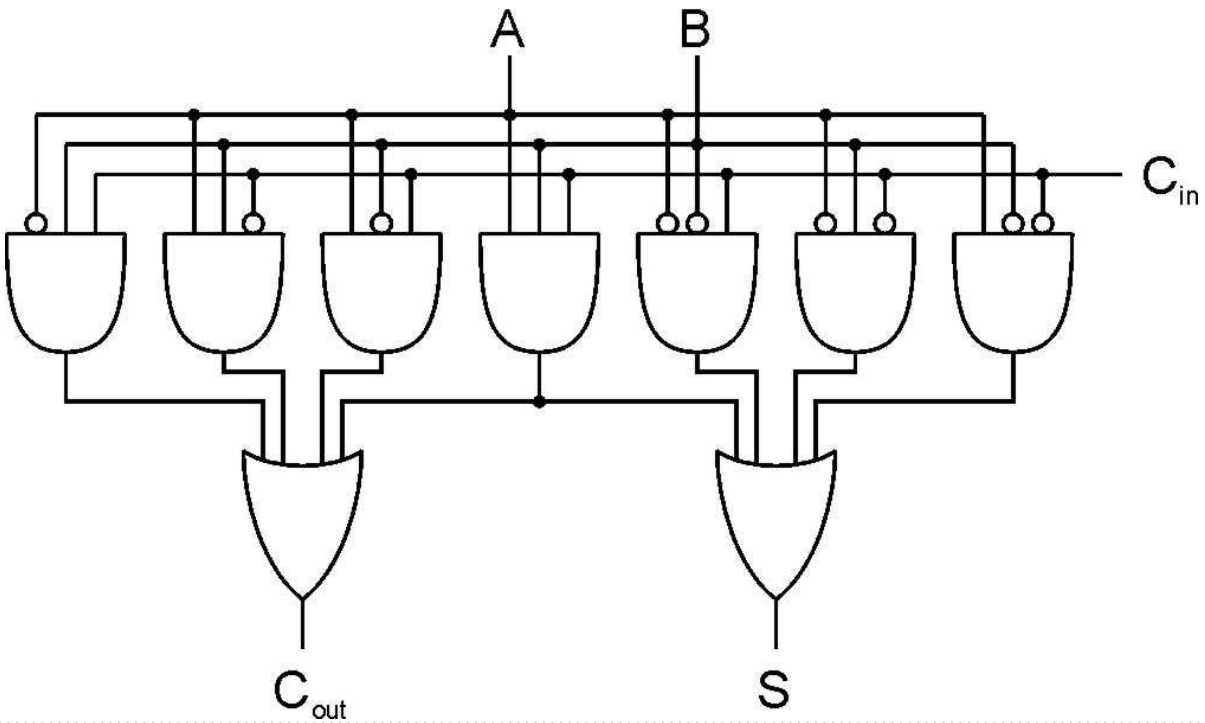



- *This is the standard symbol for a mux. S is a two-bit signal*
- *$S_1$ is the most significant bit, and $S_0$ is the least significant bit*

university of groningen

# Full Adder (1-bit binary addition) (recap)

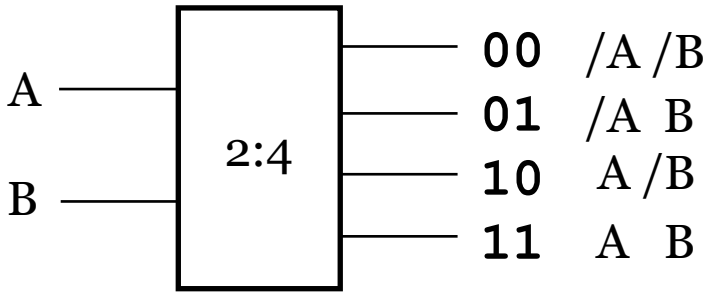
› A full adder represents one column in a binary addition operation. A and B are the bits to be added. $C_{in}$ is the carry-in from the previous column (from the right). There are two outputs: S = sum, $C_{out}$ = carry-out (to the next column).

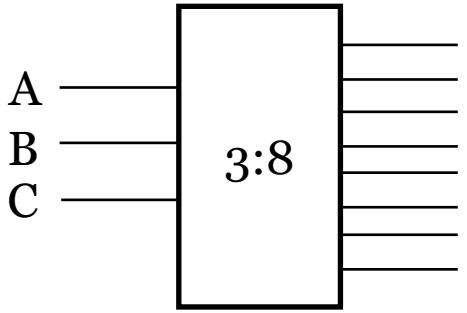


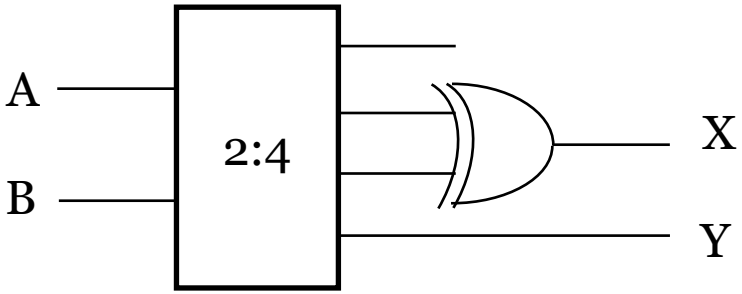

| $C_{in}$ | A | B | S | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

university of groningen

# Decoder (more)



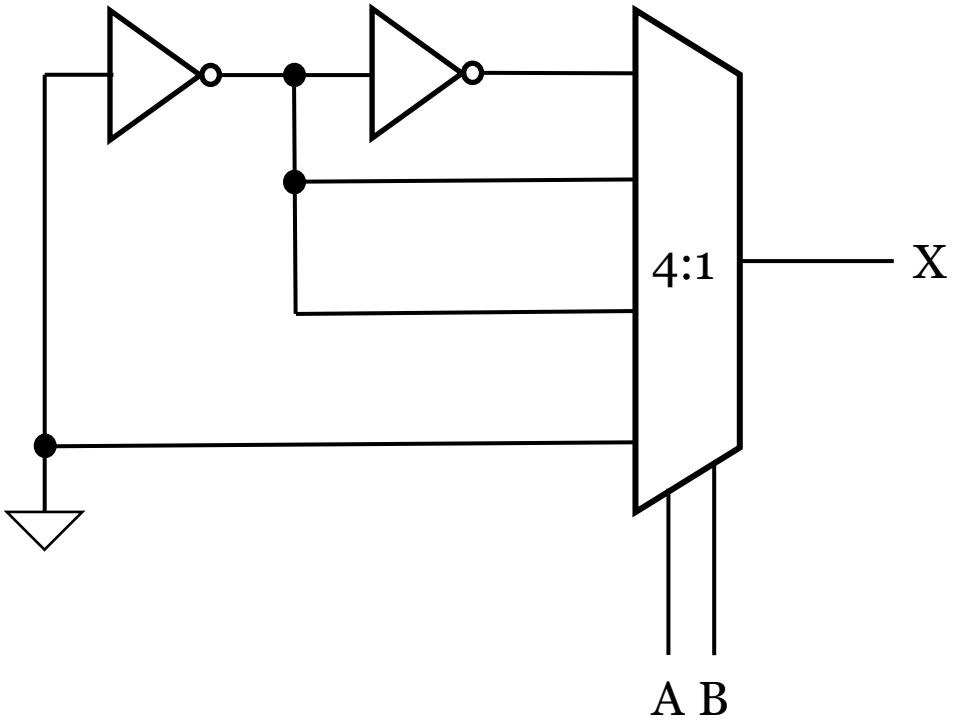




| A | B | X | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

- What are X and Y?
- What if we name X = S (sum) and Y = $C_{out}$?
- This is a combinational circuit called *half-adder*
- Can we also make a full-adder with a decoder?

# Multiplexer (Mux) (more)



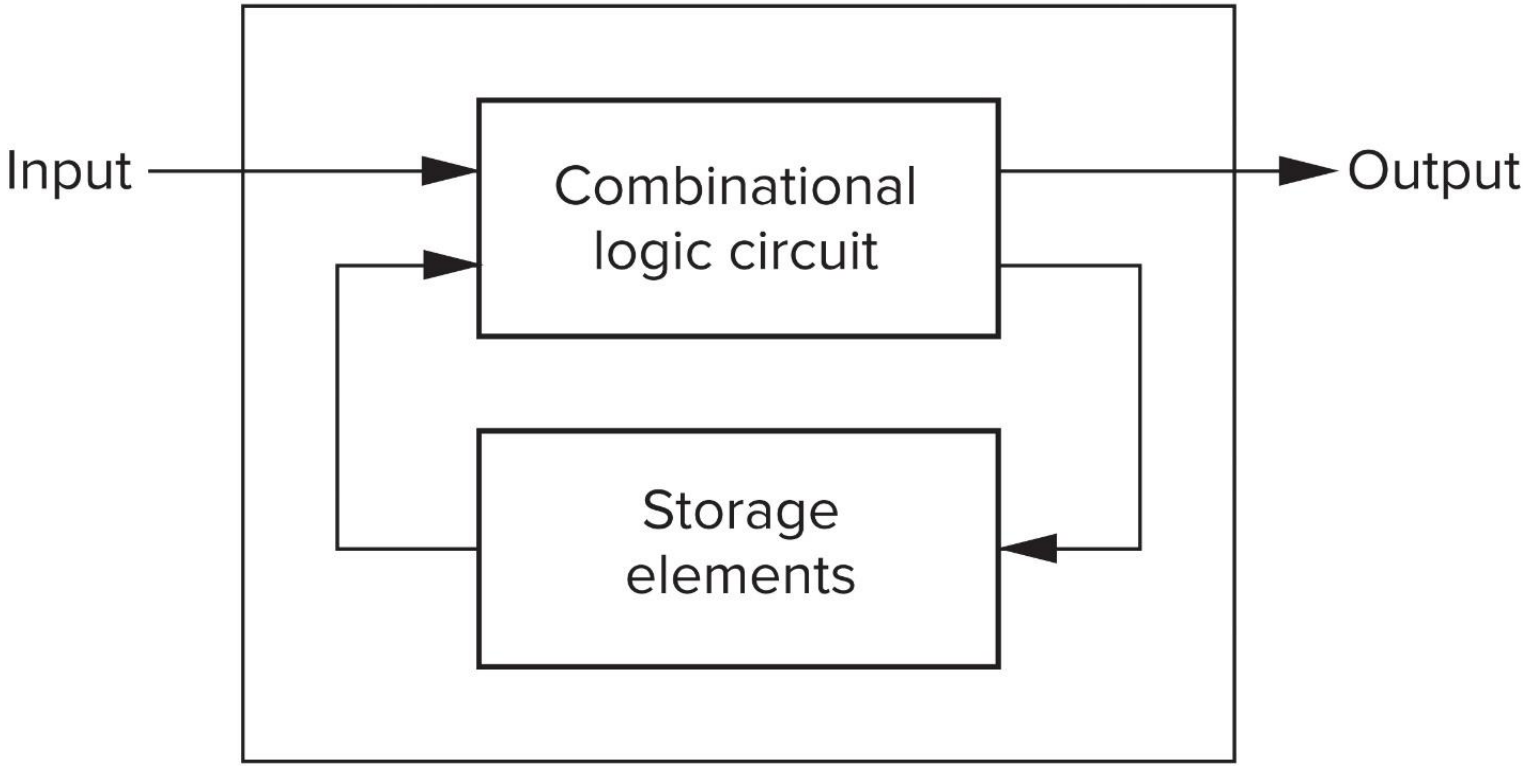

- What function is implemented here?
- This is a combinational circuit (inputs and select lines are “equivalent”)!
- Question: how many Select inputs will an 8:1 MUX have?

university of groningen

# Sequential Logic Circuit: Finite State Machine

- Uses both combinational logic and storage
- "Remembers" state, computes output (and new state) depending on previous state and new input data





university of groningen

# What is State?

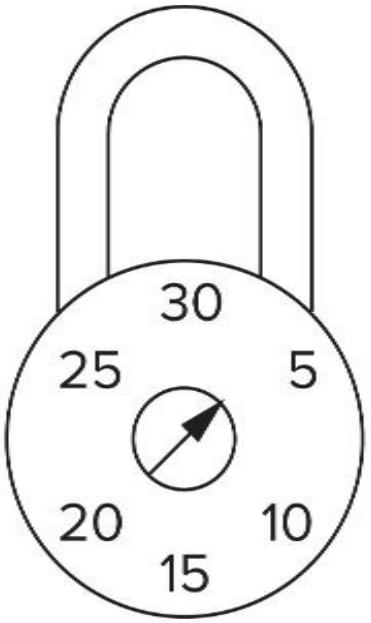
› The state of a system is a snapshot of all the relevant data that describes the system

- The state of a basketball game can be represented by the scoreboard: Number of points, time remaining, possession, etc.
- The state of a tic-tac-toe game can be represented by the placement of X's and O's on the board

university of groningen

# Example: A Combination Lock





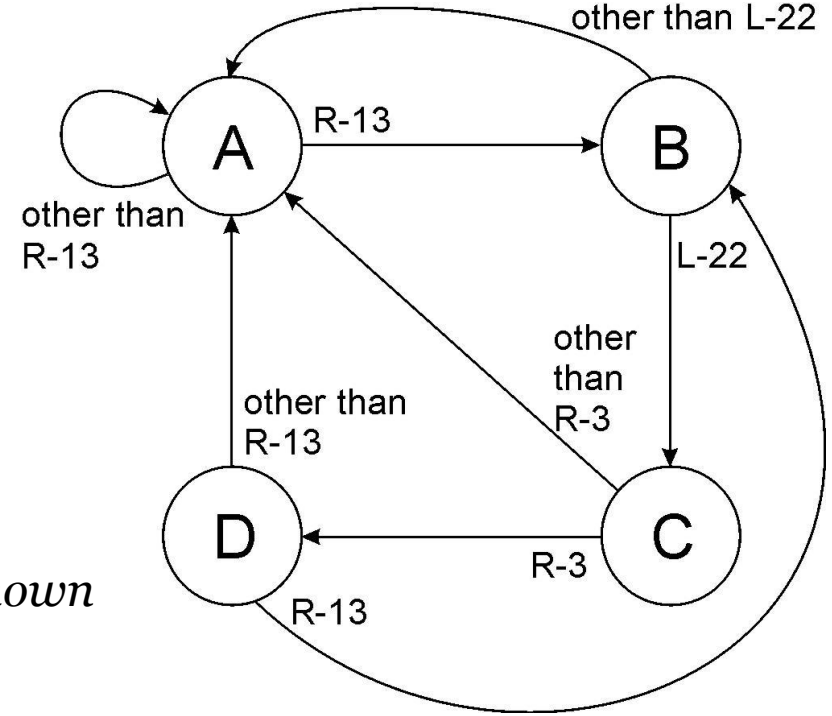

(a)

- Output (lock open / closed) depends on a sequence of actions
- For example, R-13, L-22, R-3

- Four States:
- **A:** The lock is not open, and no relevant operations have been performed
- **B:** The lock is not open, and the user has completed the R-13 operation
- **C:** The lock is not open, and the user has completed R-13, followed by L-22
- **D:** The lock is open. (The user completed R-13, L-22, and R-3)

# State Diagram

- Description of a state machine
- Shows all states (nodes), output value for each state (value in node), transitions (edges) between states for different input values (edge labels)




*The output value is not shown in this example*

# Finite State Machine: Definition

› A finite state machine (FSM) is a description of a system with the following elements:

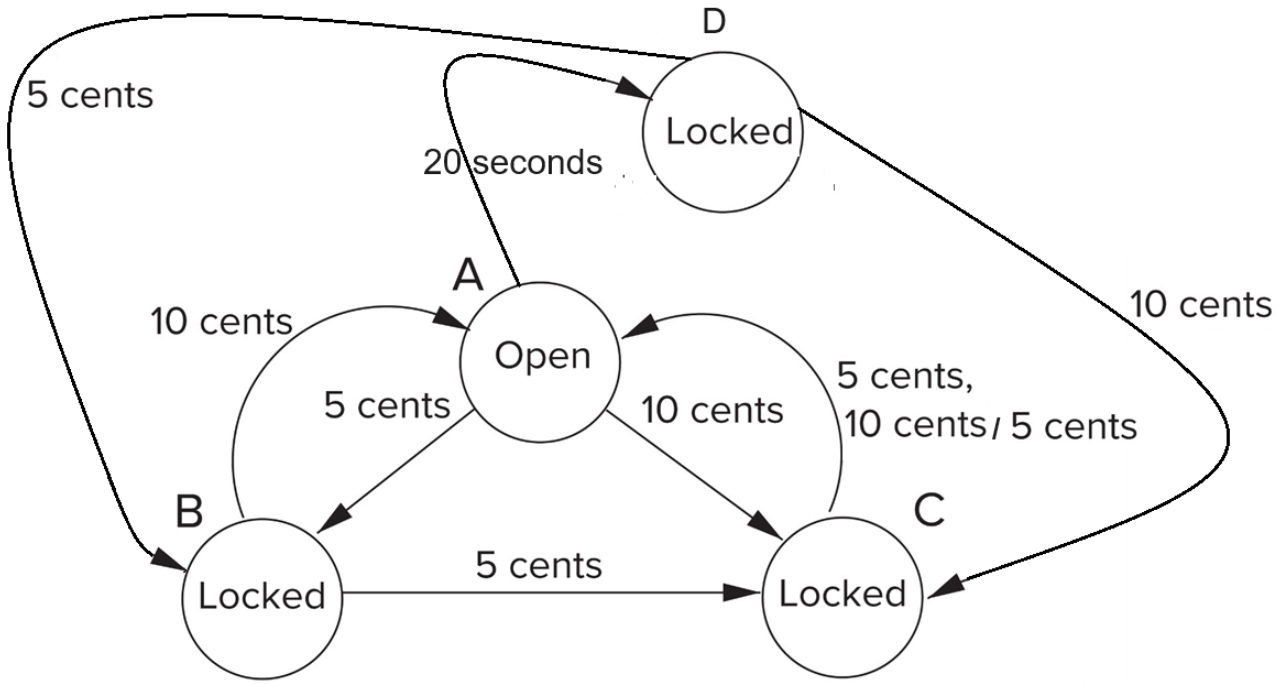
1. A finite set of states
2. A finite number of external inputs
3. A finite number of external outputs
4. An explicit specification of all state transitions
5. An explicit specification of what determines each output value

› A state diagram is one representation of an FSM

- Inputs trigger state transitions
- An output value is associated with each state (or with each transition in some specifications)

# Another Example: Drink Machine

› Suppose you have an old drink machine that accepts nickels and dimes. Inserting 15 cents will unlock the drink dispenser.




nickel = 5c
dime = 10c

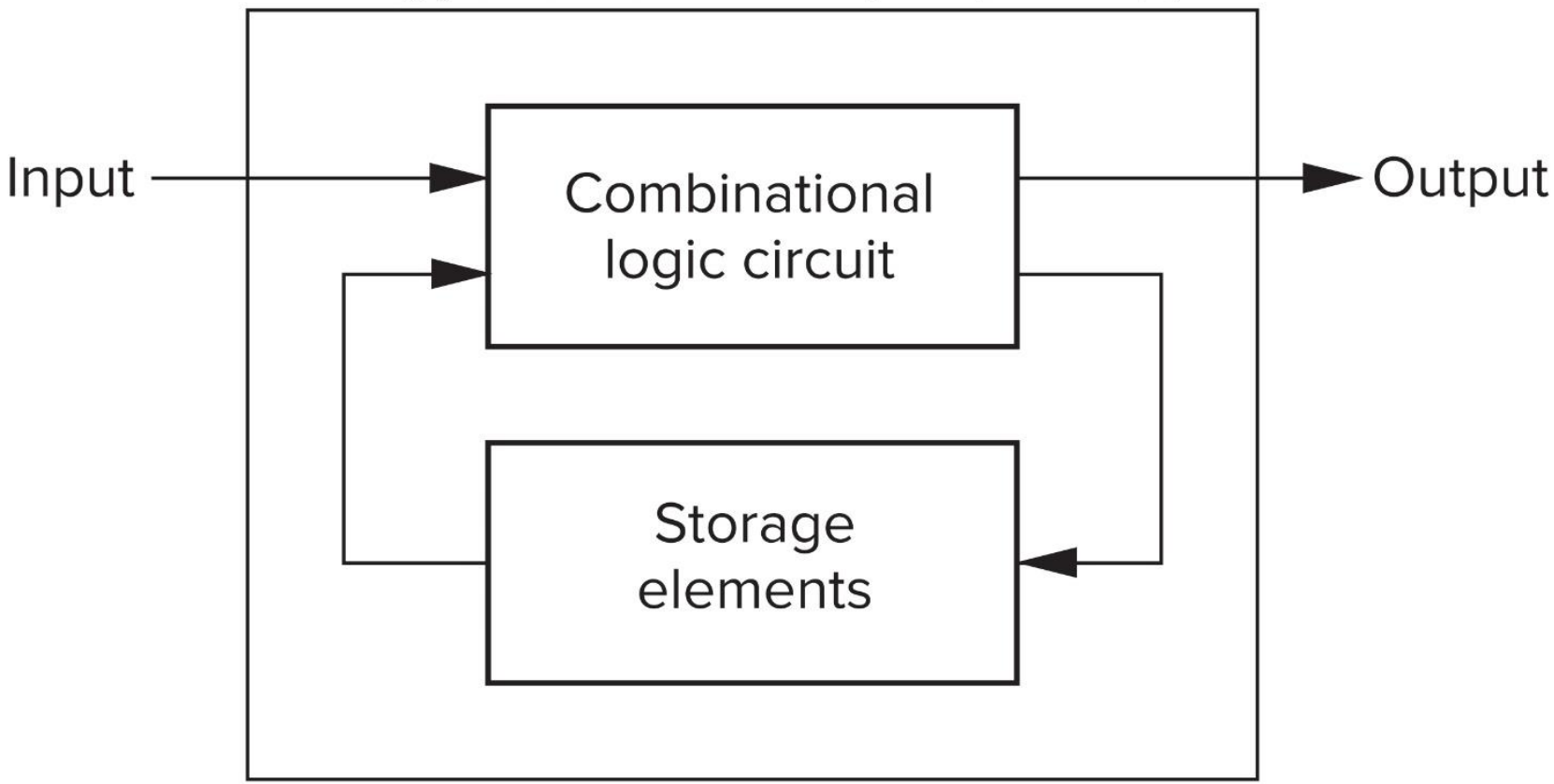
› A: Open -- one drink can be removed
› B: User has inserted five cents
› C: User has inserted ten cents
› D: Locked, no coin has been inserted

university of groningen

# Implementing a Finite State Machine

- Combinational Logic -- given current state and input, what is the next state and output?
- Storage Elements -- stores state bits until next cycle

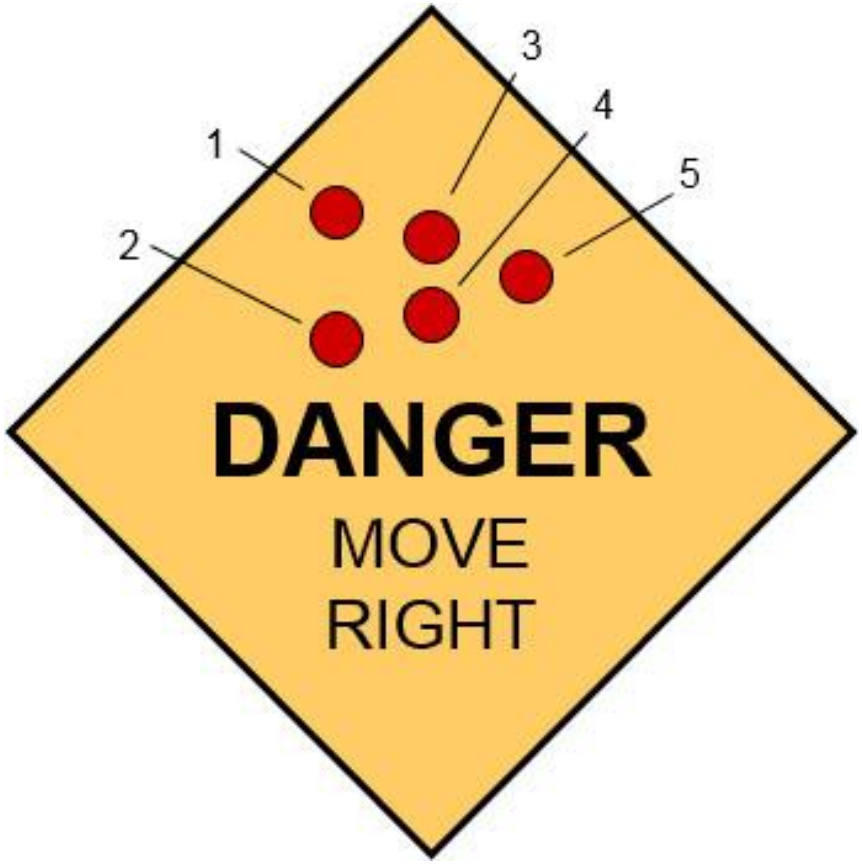




# FSM Example: Danger Sign (US)

- One input: on/off switch
- Five outputs: lights
- When sign is off, no lights
- When sign is on:

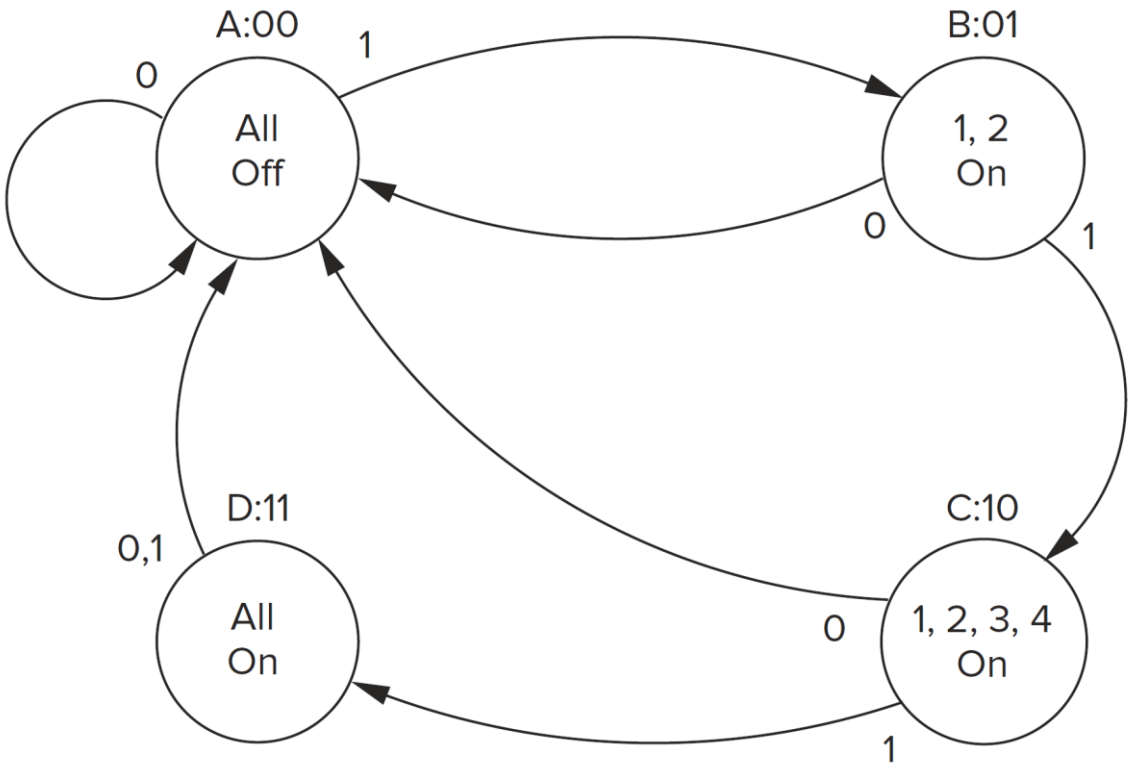
1. No lights
2. 1 and 2 on
3. 1, 2, 3, and 4 on
4. 1, 2, 3, 4, and 5 on




- Repeats states 1 to 4 as long as switch is on

university of groningen

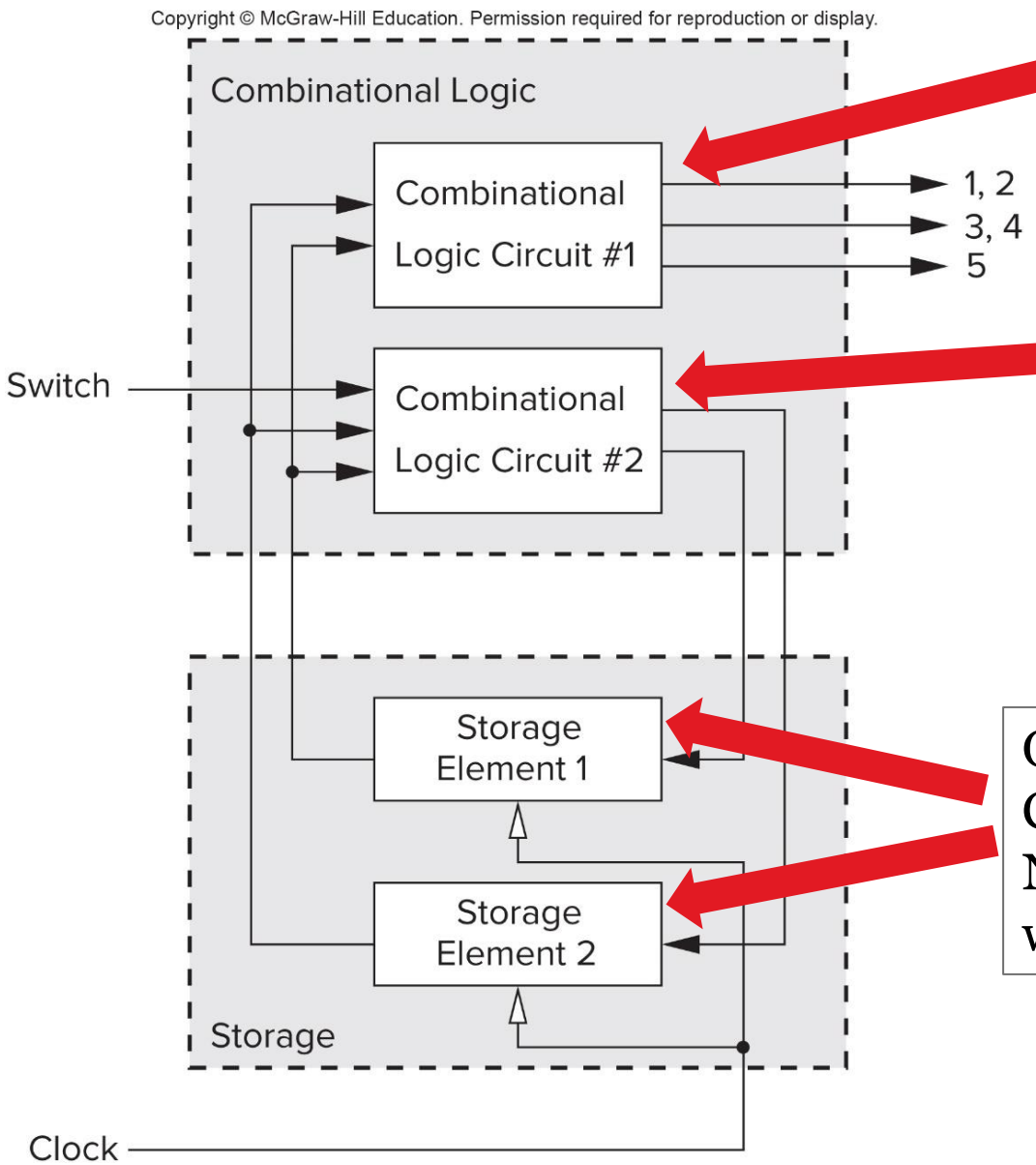
# Danger Sign: State Diagram




- Four states. Assign a two-bit label to each state.
- Input: 1 = on, 0 = off. State transitions are labeled with the input that causes that transition.
- Outputs: 1 bit per light.

university of groningen

# Danger Sign: Implementation




Given current state, what should outputs be?

Given current state and input, what should the next state be?

One storage element for each state bit. Current state (left) is input to logic. Next state (right) is output from logic, written to storage on next clock cycle.

# Danger Sign: Truth Tables

- Two truth tables: one for each combinational logic circuit.
- On left, translate state S[1:0] to output signals:
  W = lights 1 & 2, X = lights 3 & 4, V = light 5
- On right, translate input (Switch) and state S[1:0] to next state: S'[1:0]

| S[1:0] | W | X | V |
|---|---|---|---|
| 00 | 0 | 0 | 0 |
| 01 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 |
| 11 | 1 | 1 | 1 |

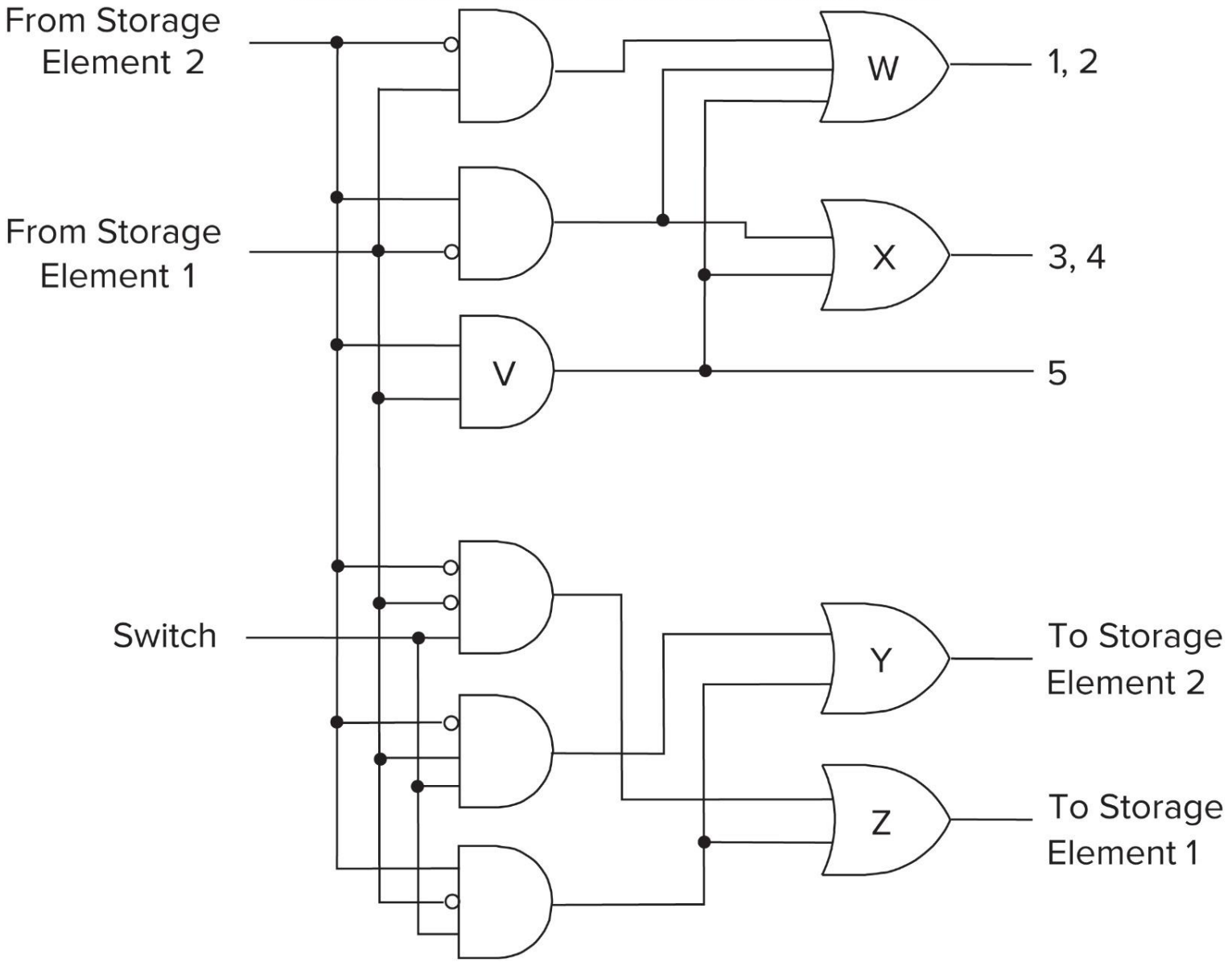
| Switch | S[1:0] | S'[1:0] |
|---|---|---|
| 0 | 00 | 00 |
| 0 | 01 | 00 |
| 0 | 10 | 00 |
| 0 | 11 | 00 |
| 1 | 00 | 01 |
| 1 | 01 | 10 |
| 1 | 10 | 11 |
| 1 | 11 | 00 |

- All information can be read from the **state diagram**

university of groningen

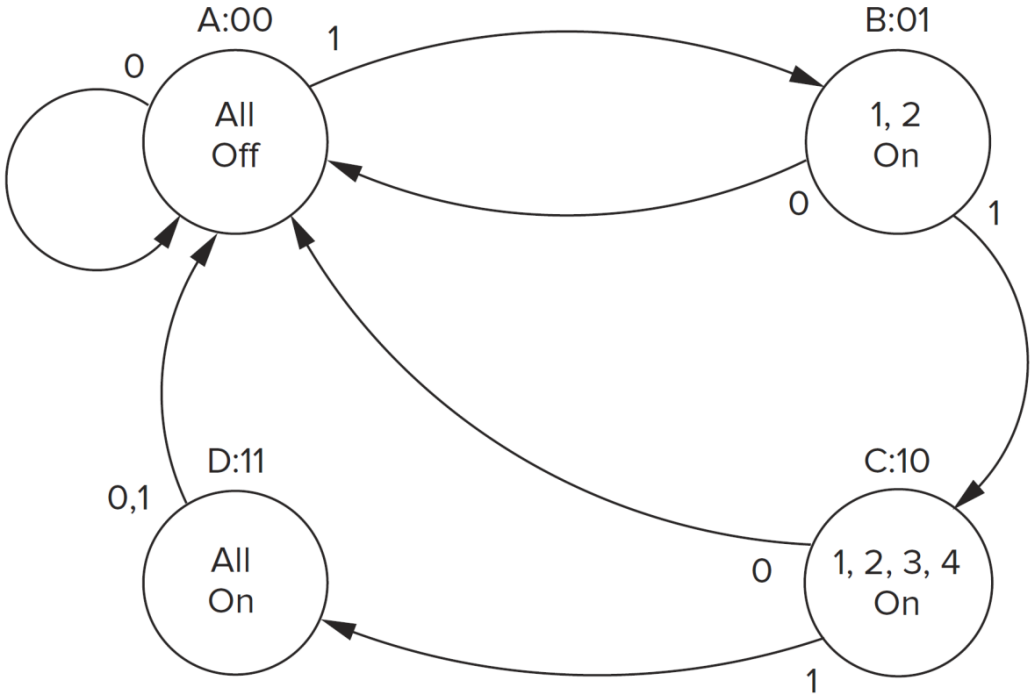
# Danger Sign: Combinational Logic

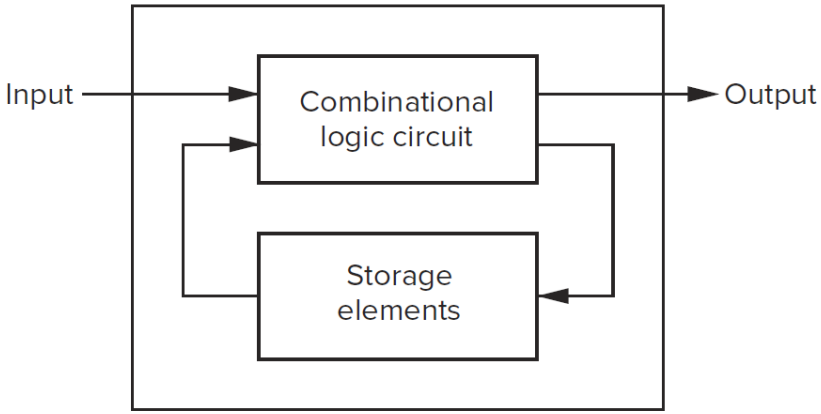




# FSM Summary

› State diagram, truth tables, and circuit are three different representations of the state machine. Given one, we can easily derive the other two.




| S[1:0] | W | X | V |
|---|---|---|---|
| 00 | 0 | 0 | 0 |
| 01 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 |
| 11 | 1 | 1 | 1 |

| Switch | S[1:0] | S'[1:0] |
|---|---|---|
| 0 | 00 | 00 |
| 0 | 01 | 00 |
| 0 | 10 | 00 |
| 0 | 11 | 00 |
| 1 | 00 | 01 |
| 1 | 01 | 10 |
| 1 | 10 | 11 |
| 1 | 11 | 00 |

# Synchronous versus Asynchronous

- In examples so far, no fixed notion of time.
- The outputs may occur any time between inputs.
- This is an **asynchronous** system. The inputs change the output whenever they happen…

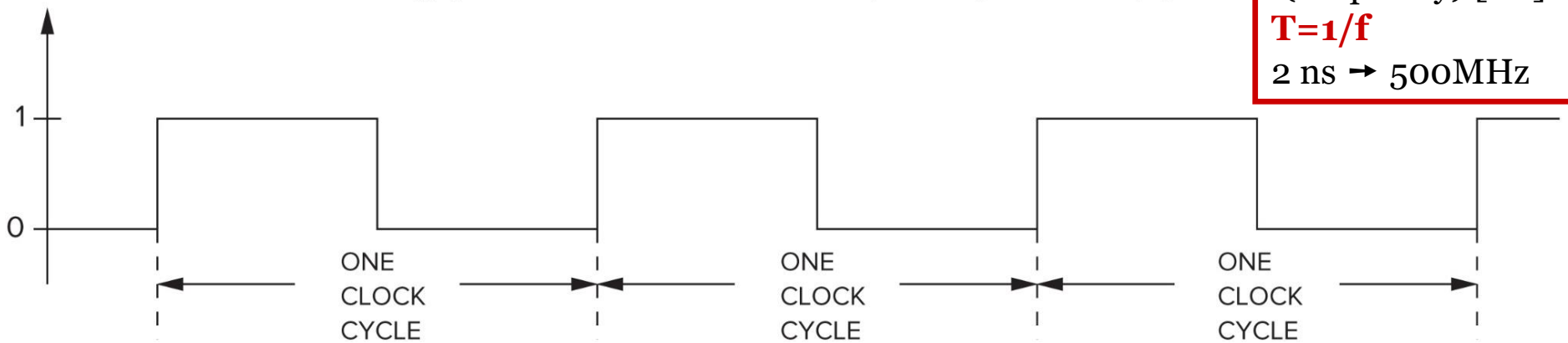
- In a **synchronous** system, state transitions happen at fixed time intervals. At each interval, the input and state determine the transition.

university of groningen

# Synchronous versus Asynchronous

- A **clock signal** is used to trigger the state transitions.
- For example, a transition happens whenever the clock changes from 0 to 1.






T (period) [s]
f(frequency) [Hz]
**T=1/f**
2 ns → 500MHz

# Storage Elements: D-Latch?

- Suppose we use a D-Latch to store each data bit.
- We want the new state to be written into the latch each clock cycle. So we can use the **Clock** signal as write-enable (**WE**).

- **Problem:**
- As long as WE = 1, the new data will change the stored data.

- This means that we need WE (Clock) to go back to zero **before** the next state is prepared by the logic circuit. Otherwise, we would transition through two or more states in a single clock cycle.

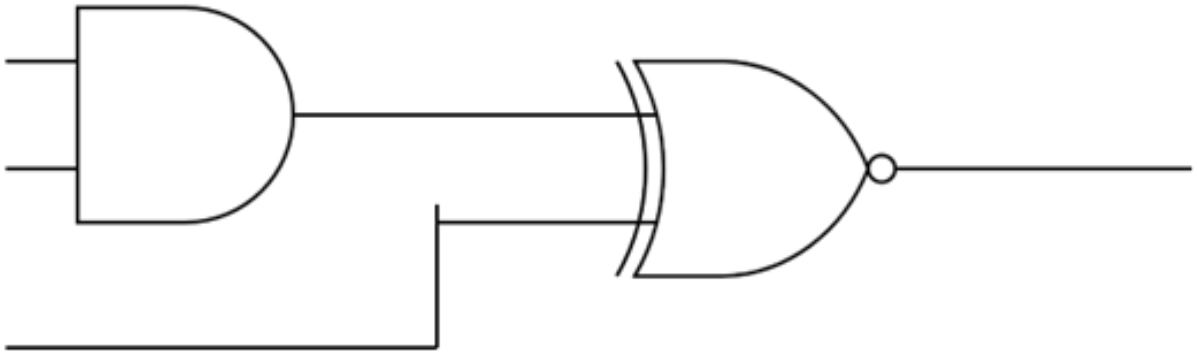
- **Solution: flip-flop!**

# Signal Propagation Delay

- Propagation delay is the amount of time required after an input signal is applied until the output of the circuit has stabilized to the correct output signal.
- Each gate adds a short delay to the signal.
- The longer paths experience larger delays.

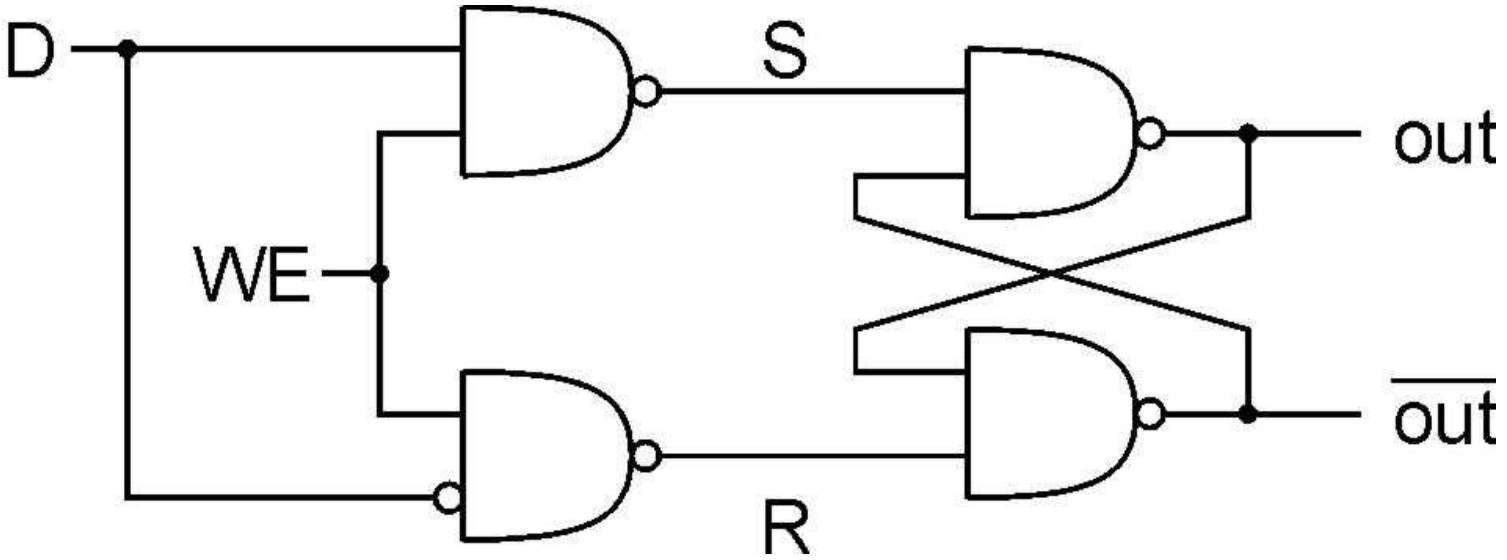
# Example: Asynchronous State Transition

- In the following example, the initial input is (0,0,0).
- The input changes to (1,1,1).
- We expect the same output however, there will a transient output due to the propagation delay.



# D-Latch: Simpler Control for One-Bit Latch (Recap)

- D = data input, WE = write enable
- When WE = 0, latch output does not change (D is irrelevant)
- When WE = 1, latch output = D

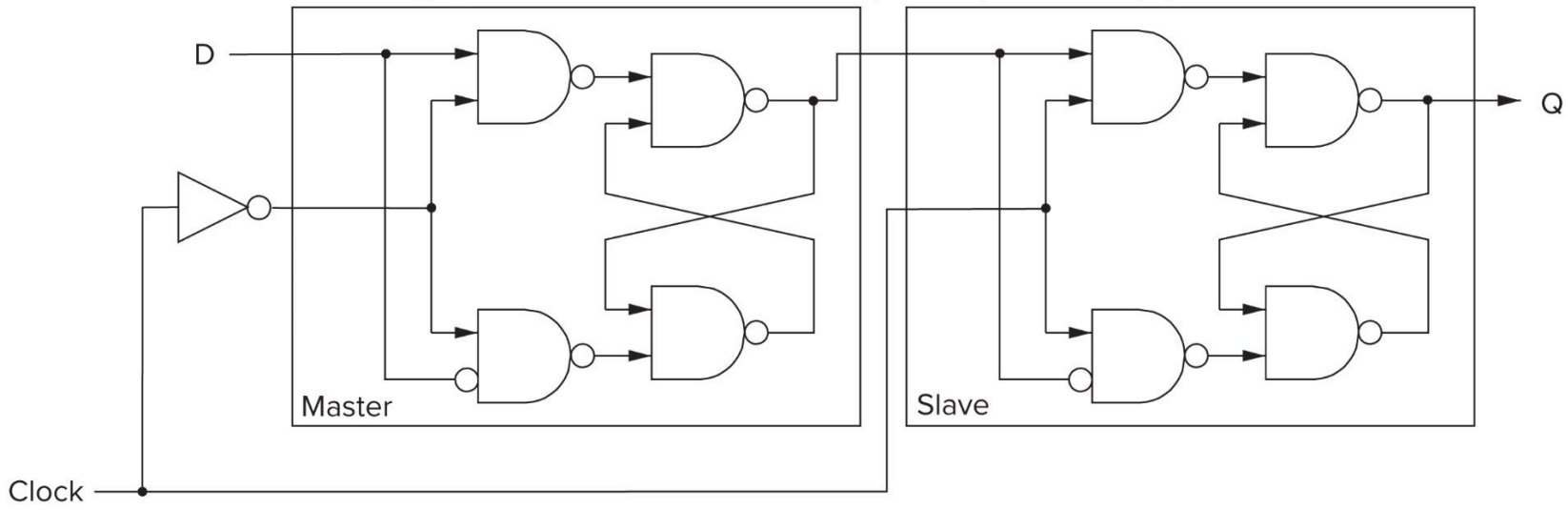


university of groningen

# Storage Elements: Flip-flop

› A flip-flop uses two latches to separate the reading phase and the writing phase of the clock cycle



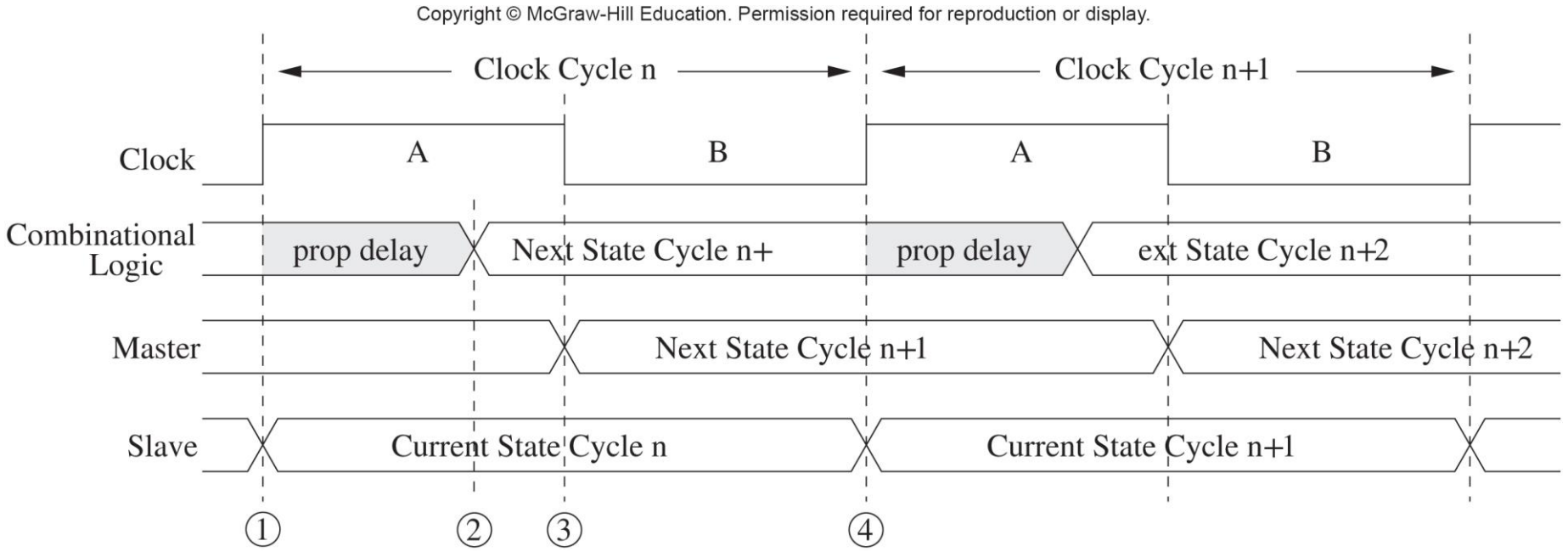



When Clock = 0 (write phase), new data (D) is stored in the Master latch, but output (Q) **does not change**.

When Clock = 1 (read phase), data from Master is stored into the Slave and the output (Q) **changes** to the new value.

# Timing Diagram

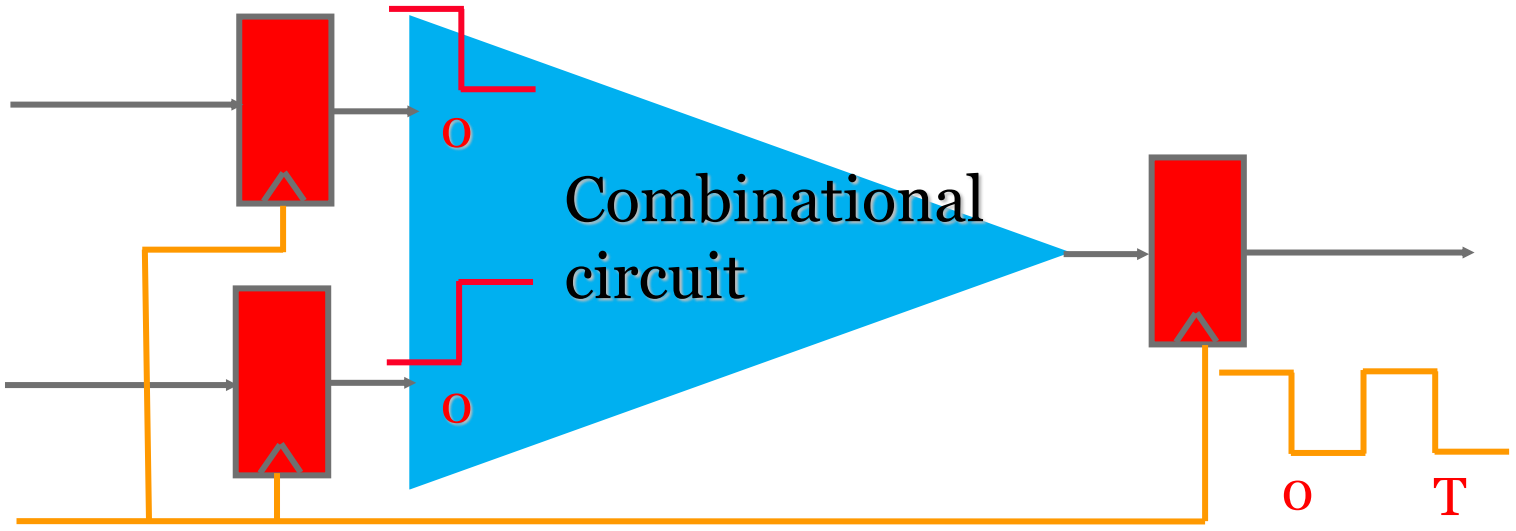
Transistors (logic gates, latches) and wires are not ideal. It takes time for the electrons to do the work or travel.




- As long as worst-case propagation delay through the logic is less than half of the clock cycle, state will properly change each clock cycle.
- Because of this behavior, flip-flop is considered an *edge-triggered* storage element: its state changes on the clock edge. A latch is *level-triggered*, because it changes as long as the WE signal is enabled.

# Clock Frequency

› Critical path



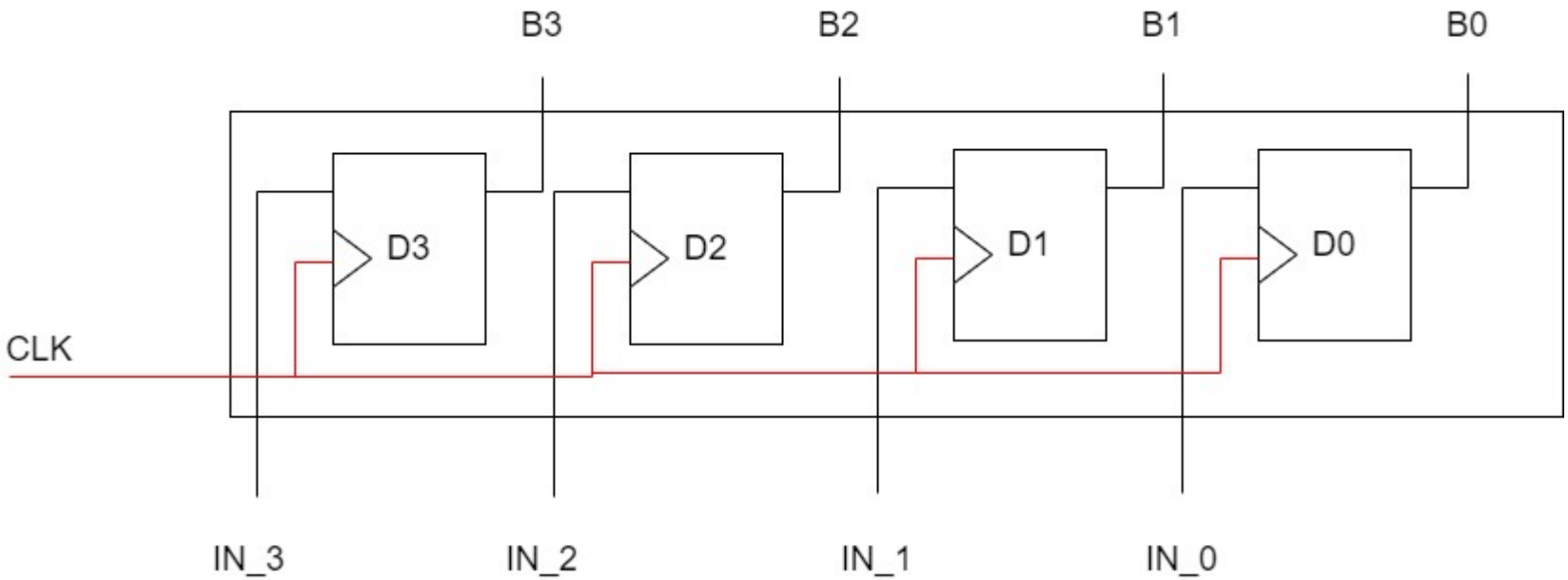

How fast can the clock be?

FYI: T= 1/f
where T is period [s] and f is frequency [Hz]
The period of 1GHz clock signal is 1 ns

university of groningen

# Registers

- Registers are a type of computer memory used by the CPU.
- It acts like an N-bit memory
- They can be built out of latches/flip flops

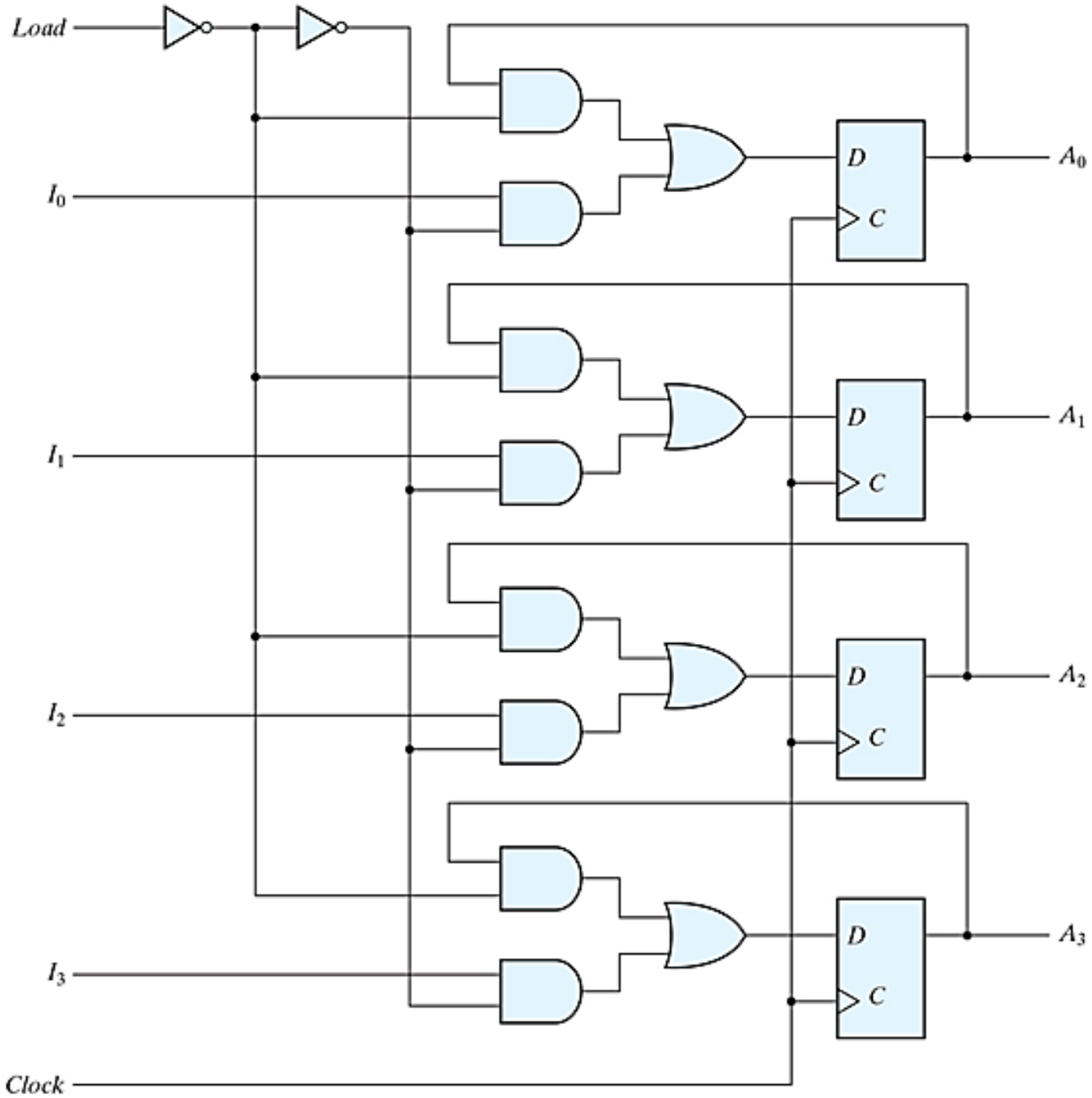


# Registers with Load Control

- The input to the memory cells (flip-flops) of a register may change with changes in the outputs of the combinational circuits.

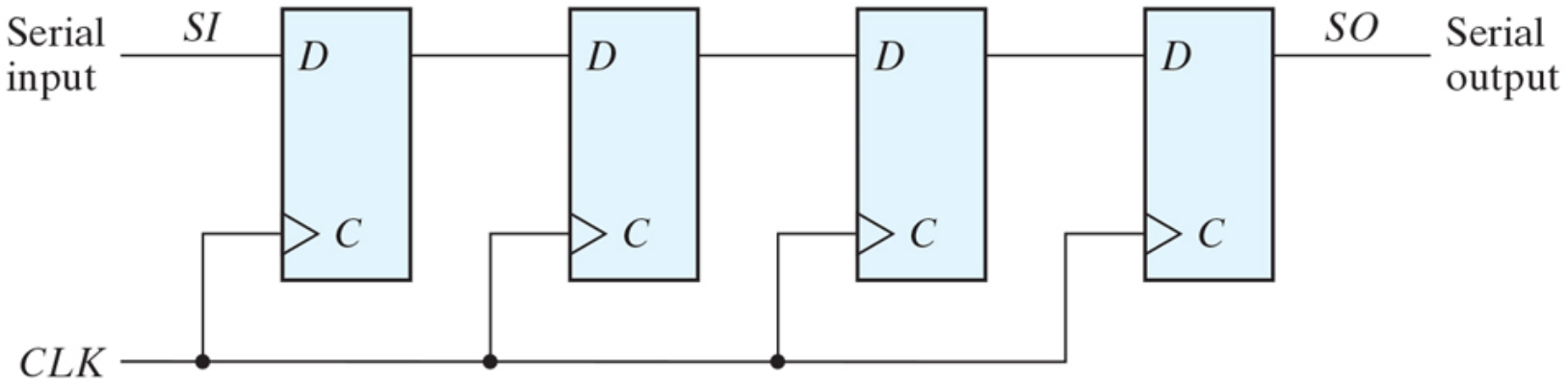
- To control when data should be loaded into a register, we use a load input

university of groningen



# Shift Registers

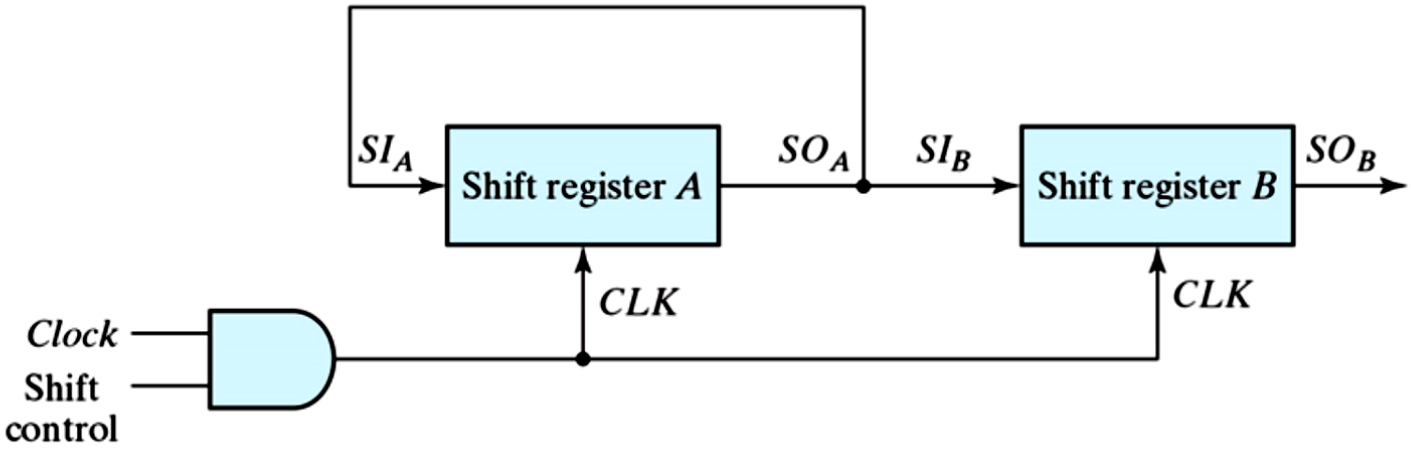
- A register may have only serial inputs, where the input is loaded bit by bit, and is moved from one flip-flop to the next (hence shift registers)
- Reading the data from these registers is also done serially.
- These registers are called serial-in, serial-out, shift registers.



university of groningen

# Serial Transfer

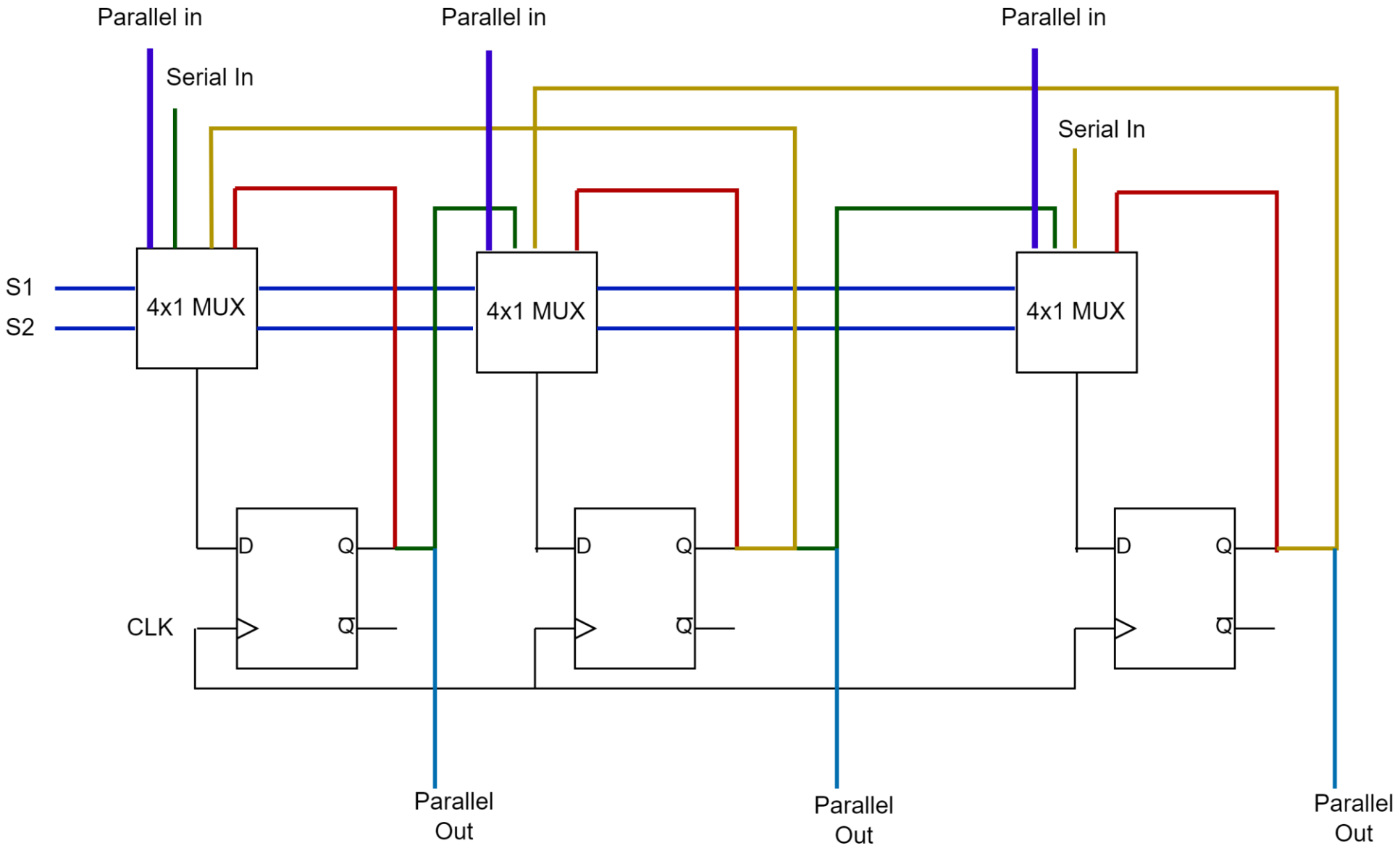
- The content of a register can be copied into a second register serially.
- This is called serial transfer operation.



# General Shift Registers

- A general shift register has the capabilities of:
  - Parallel data load
  - Serial data load by
    - Shift to left and insert a bit from right
    - Shift right and insert a bit from left
  - Reset (load logical zero)
  - Preserve current data
- A 4x1 mux can be used to decide which action should be carried out.

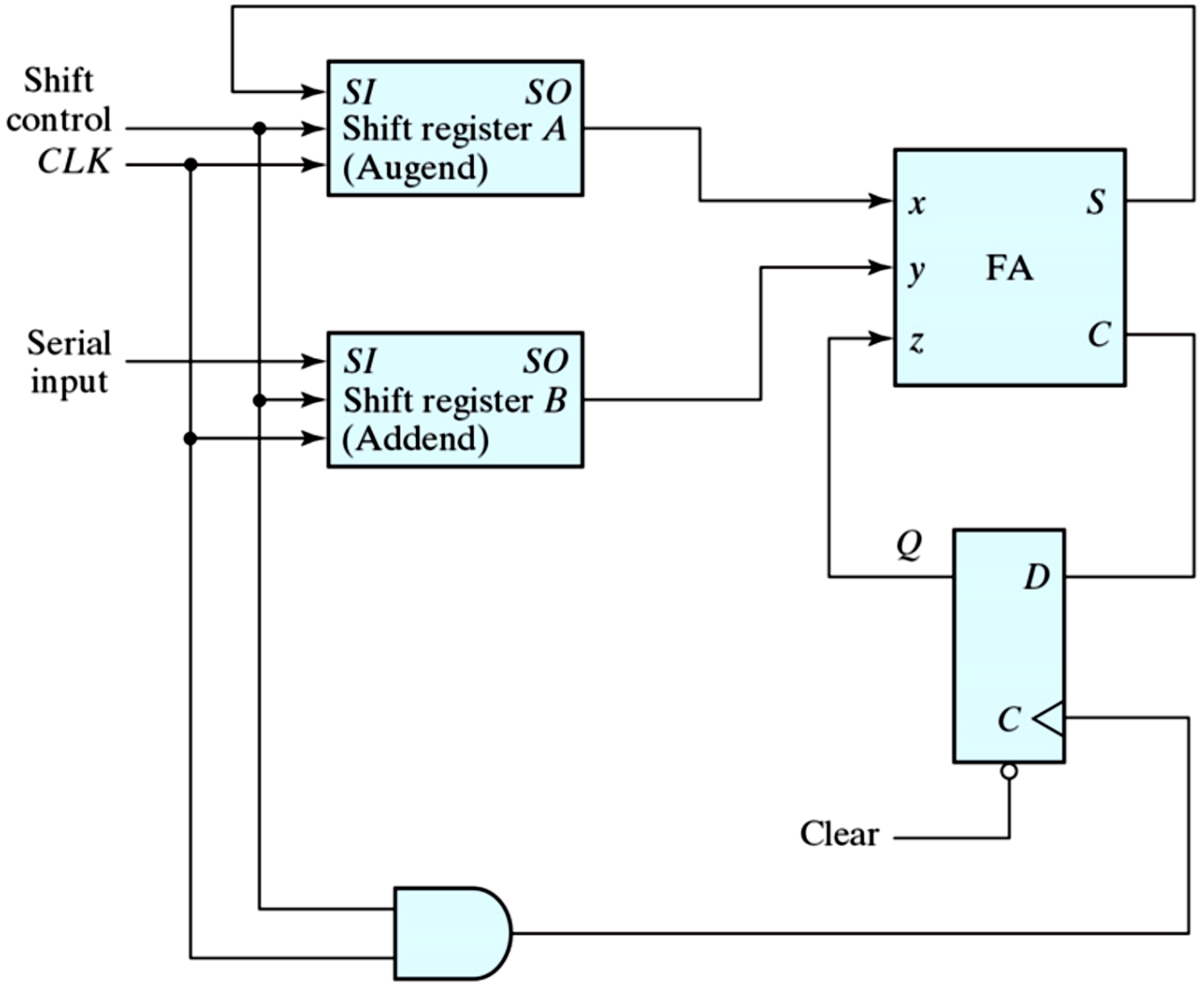
| S1 | S2 | Action |
|---|---|---|
| 0 | 0 | Keep Value |
| 0 | 1 | Shift Left |
| 1 | 0 | Shift Right |
| 1 | 1 | Parallel Load |


Parallel in
Parallel in
Parallel in
Serial In
Serial In
S1
S2
4x1 MUX
4x1 MUX
4x1 MUX
D
Q
D
Q
D
Q
CLK
Q
Q
Q
Parallel Out
Parallel Out
Parallel Out

# Serial Adder

- Assume we have two registers A, and B.
- We want to add the values in these registers, putting the result in register A.
- We want to use only one full adder, and a flip-flop.

# Serial Adder

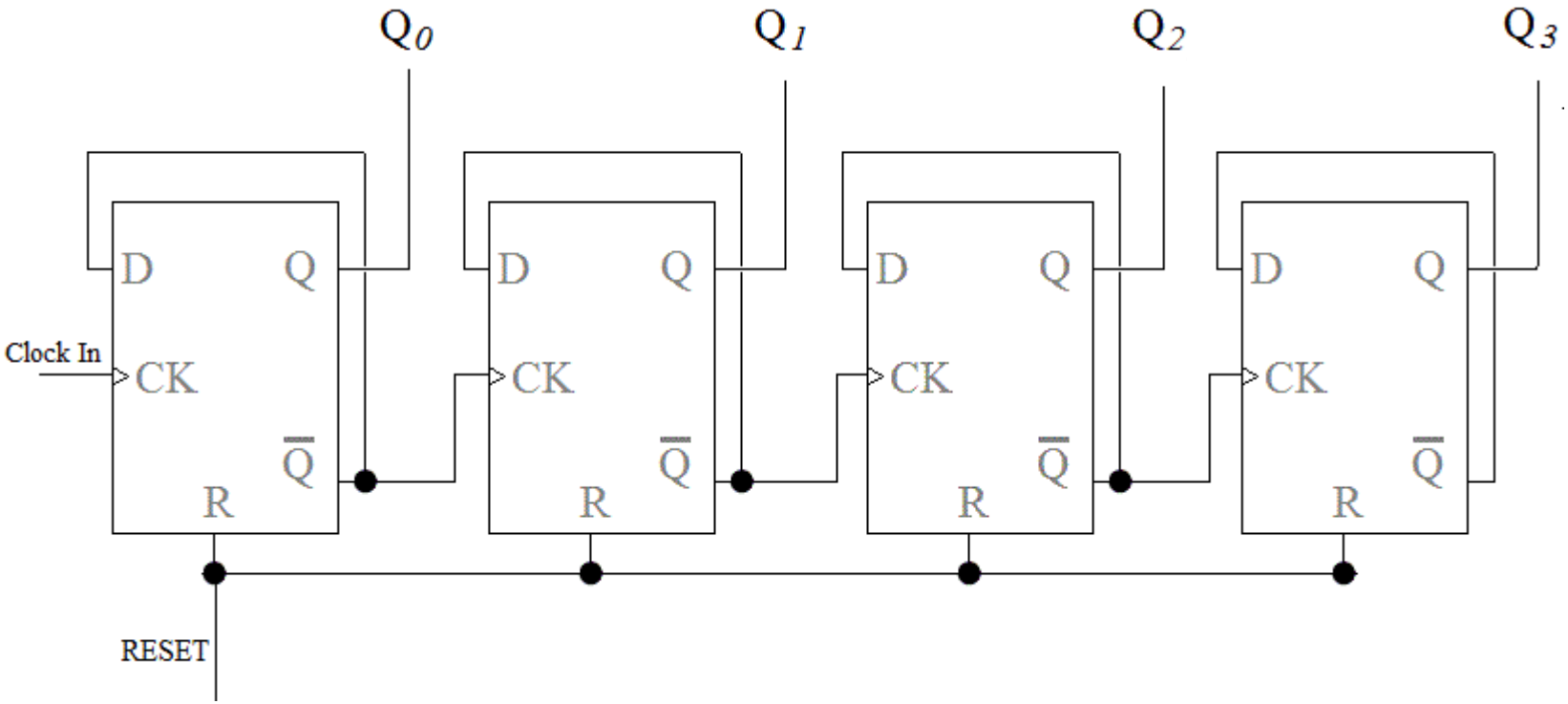


# Counters

- A counter is a sequential circuit to count numbers from zero, to a maximum defined for the circuit.
- Counters can be designed as synchronous, or asynchronous circuits.
- Ripple counter is an example of an asynchronous counter.

university of groningen

# Ripple Counter



# Summary

- Sequential logic circuits are used to combine the current input and the past output to generate new outputs.
- The current output is considered as the state of the circuit.
- Finite State Machine are used to define the transitions from one state to the next based on the given inputs.
- Synchronized circuits limit the changes in the output to discrete points in time. In this way, the oscillation at the output is prevented.

university of groningen

# Questions?