Unit 6. Sequential circuits

Digital Electronic Circuits
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan-Chico <jjchico@dte.us.es> 2010-2020

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Contents

- Introduction
- Latches and flip-flops
- Synchronous Sequential Circuits (SSC) design
- SSC analysis
- SCC application examples

Bibliography

- Theory reference
  - LaMeres: 7.1, 7.2, 7.4, 7.6, 7.7
- Verilog modeling reference
  - LaMeres: 9.1, 9.2, 9.3
  - verilog-course.v: unit 6

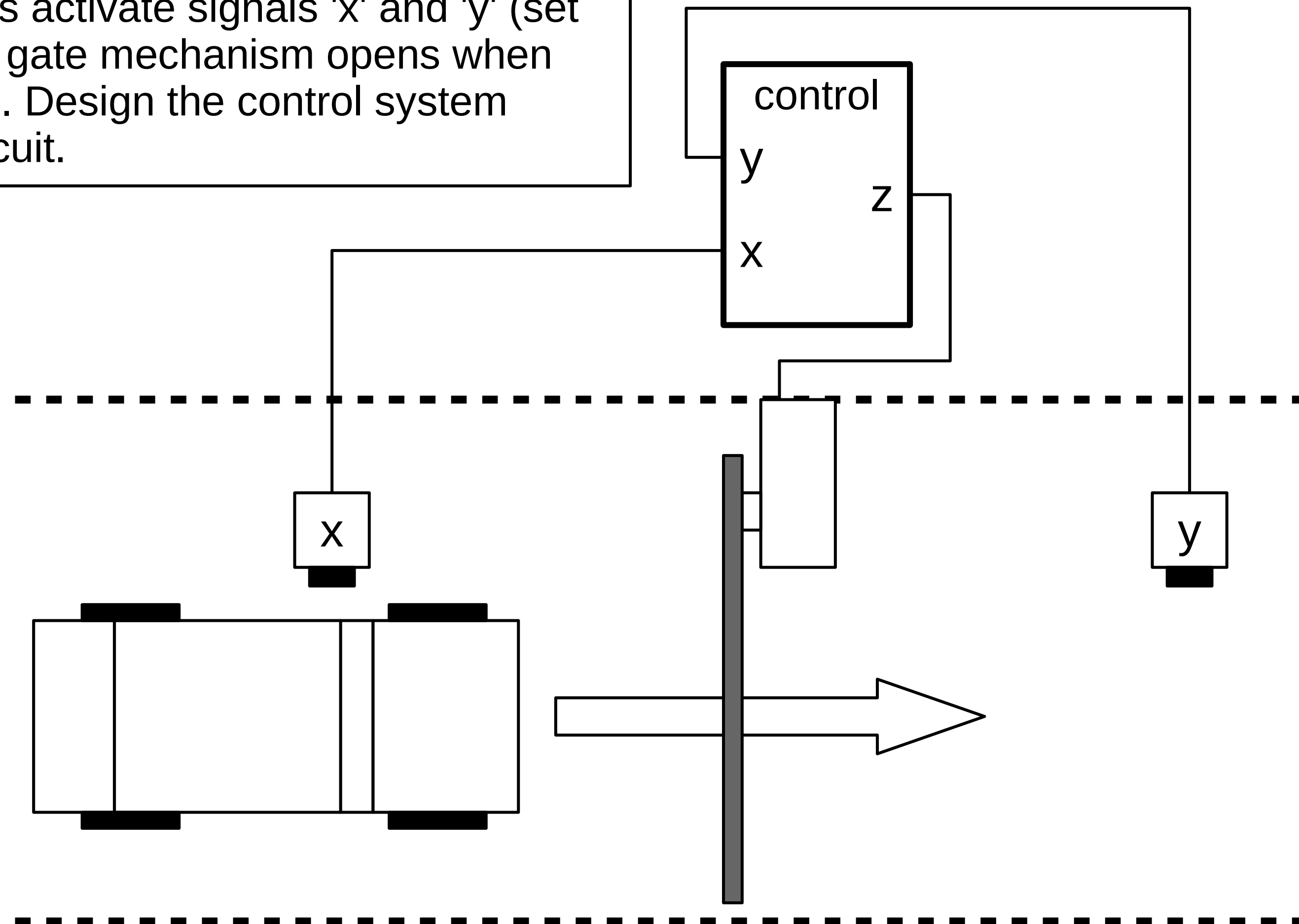
Recommended extra exercises

- Exercises from course's collection 5 (in Spanish)
  - Ex. 15. Parity detector in groups of bits. Must be designed as a Mealy machine. Why?
  - Ex. 18. Sequential two's complementer.
  - Ex. 5. Functional analysis.
  - Exs. 8 and 9. Zero-delay timing analysis.

Introduction

Example 1

A gate control system has two inputs 'x' and 'y' and one output 'z'. Inputs are connected to push buttons separated by a distance. The buttons activate signals 'x' and 'y' (set to '1') when pressed. The gate mechanism opens when z=1 and closes when z=0. Design the control system using a combinational circuit.



Introduction

- Most practical problems cannot be solved using only combinational functions.
- In many cases the action depend on the inputs and on the “state” of the system: the door is open, the light is on, etc.
- To “store” the state of the system we need new circuit elements: memory elements.
- In this unit:
  - Memory elements
  - Concept of “state” and sequential circuit
  - Techniques to design and analyze synchronous sequential circuits

Contents

- Introduction
- Latches and flip-flops
  - Latches
  - Asynchronous SR latch
  - Synchronous latches and flip-flops
  - Types of flip-flops
  - Asynchronous inputs
  - Timing restrictions
- Synchronous Sequential Circuits (SSC) design
- SSC analysis
- SCC application examples

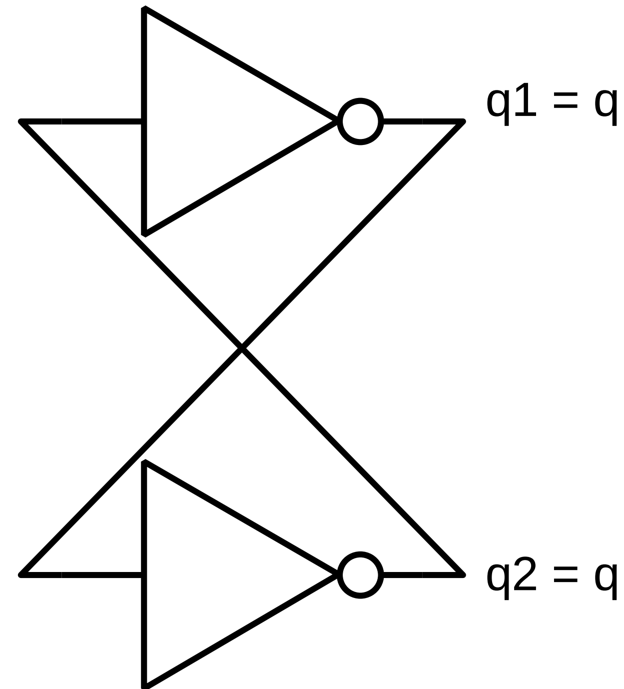
Latches

- Latches are bi-stable devices that can be placed in one of two possible stable states.
- The state the latch is in is reflected at its outputs.
- The state can be changed by actuating on some control inputs.
- Latches are the basic memory element: they store 1 bit.
- With n latches, 2^n different states can be “memorized”.

1	0	0	1	0	1	1	0

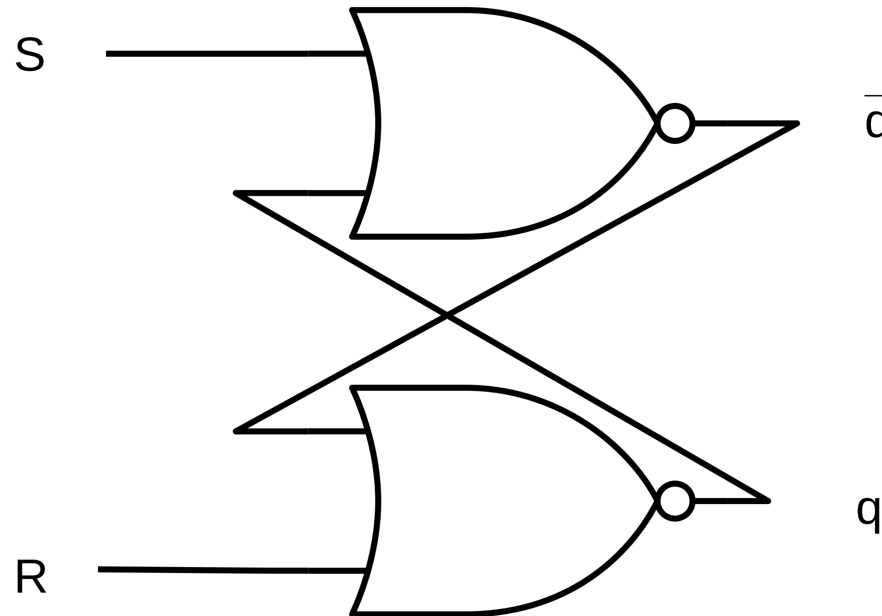
Asynchronous SR latch

- The ability to store information is obtained from the "feedback" of the outputs to the inputs: the output value maintains the input value and vice-versa
- Possible stable states:
  - q1=0, q2=1
  - q1=1, q2=0
- Notation:
  - q = q2
  - q = q1



Asynchronous SR latch

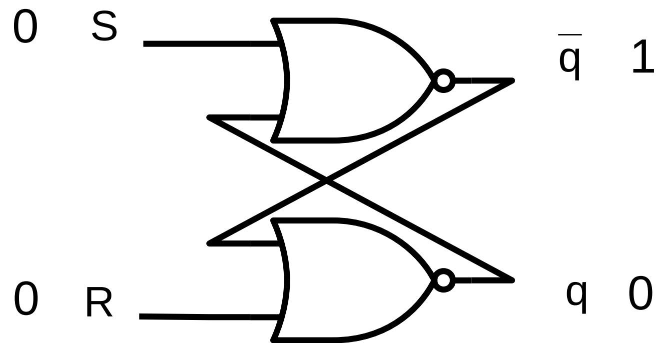
- We need a circuit with two stable states and some way to change between them.

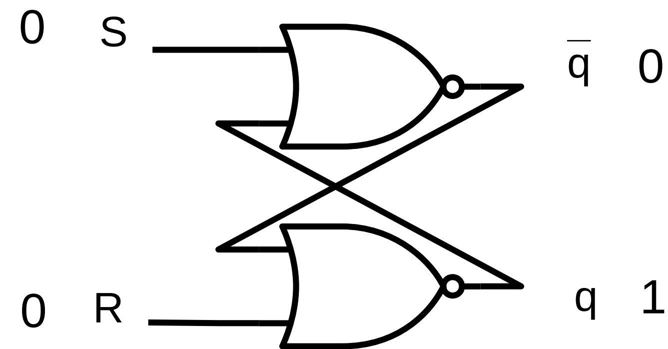


Asynchronous SR latch

Simulation

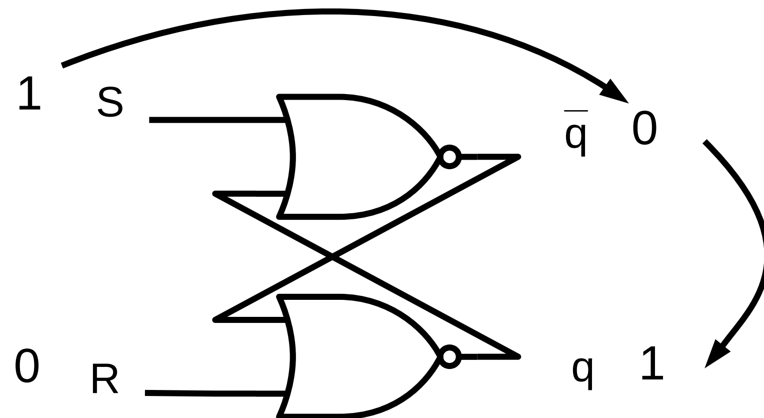
- R=S=0 the state is preserved

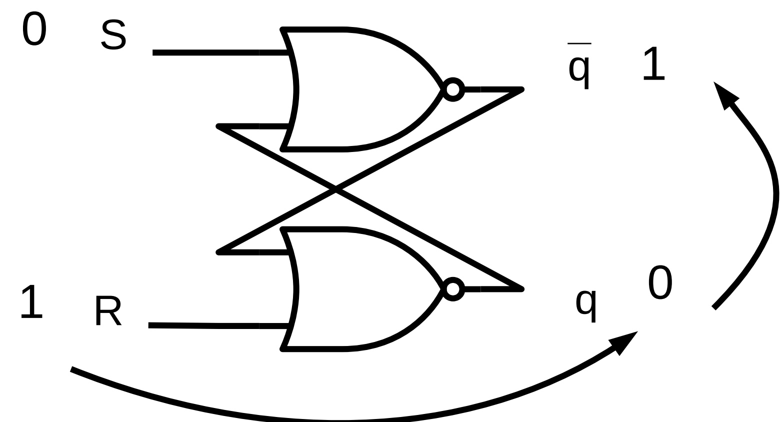




- S=1, R=0 change to 1 (set)

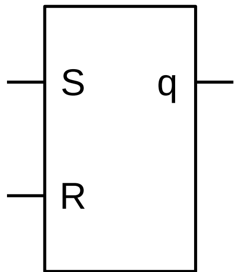
S=0, R=1 change to 0 (reset)

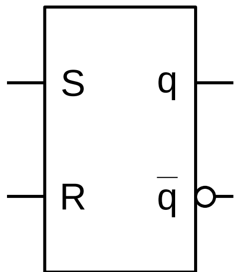




SR Latch. Formal descriptions

Symbols





State table

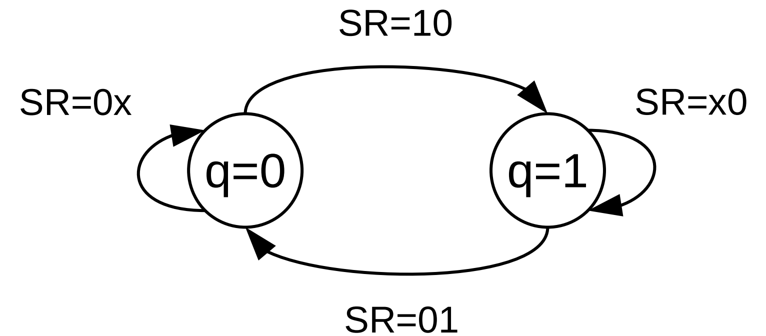
q \ SR	00	01	11	10
0	0	0	-	1
1	1	0	-	1

Q

Excitation table

q → Q	SR
0 → 0	0x
0 → 1	10
1 → 0	01
1 → 1	x0

State diagram



Verilog

```
module sra(
  input wire s,
  input wire r,
  output reg q);

  always @(s, r)
    case ({s, r})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= 1'bx;
    endcase
endmodule
```

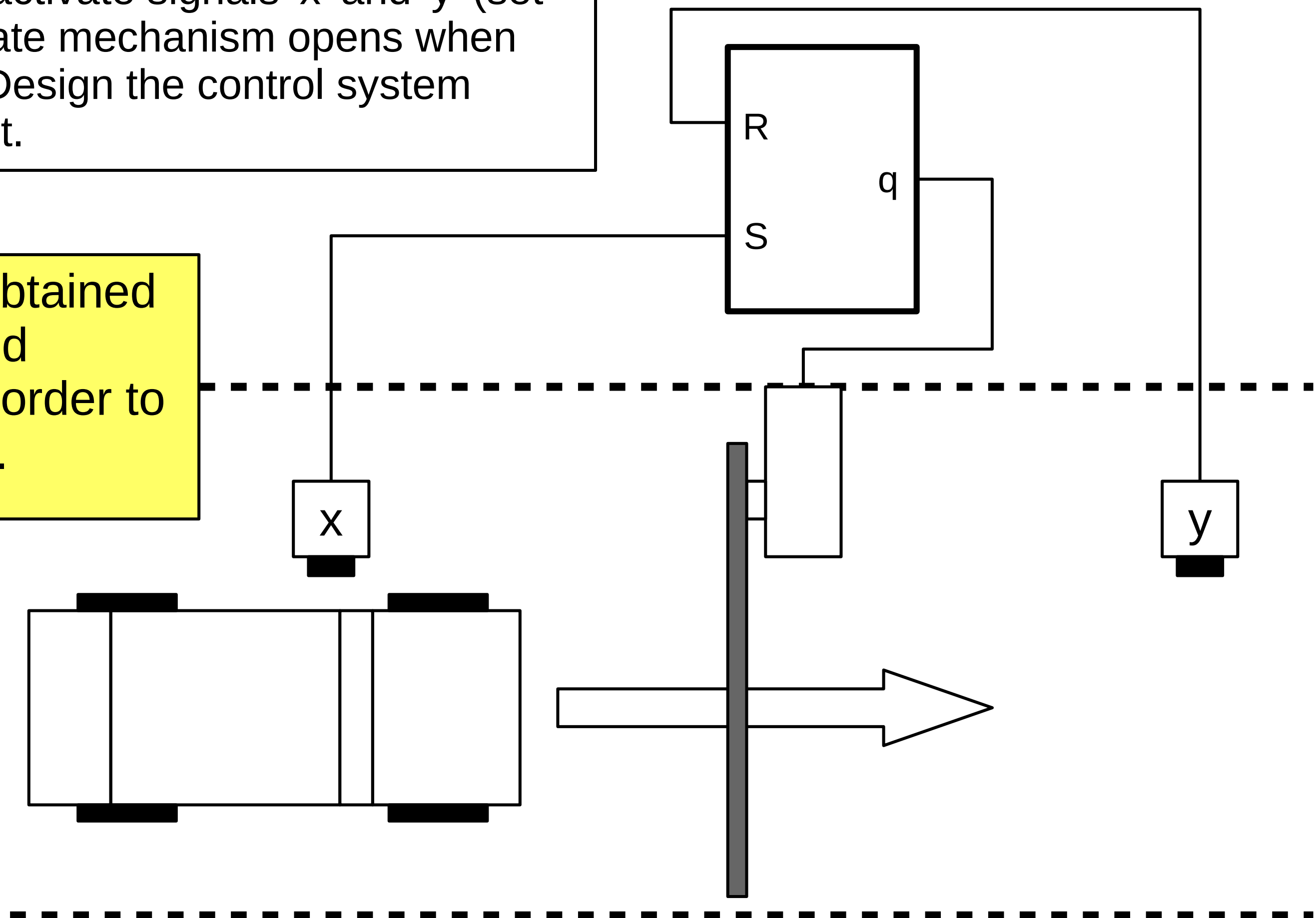
Solution to example 1

Example 1

A gate control system has two inputs 'x' and 'y' and one output 'z'. Inputs are connected to push buttons separated by a distance. The buttons activate signals 'x' and 'y' (set to '1') when pressed. The gate mechanism opens when z=1 and closes when z=0. Design the control system using a combinational circuit.

An SR latch solves the problem.

This solution has been obtained “intuitively”. We also need “systematic” methods in order to solve complex problems.

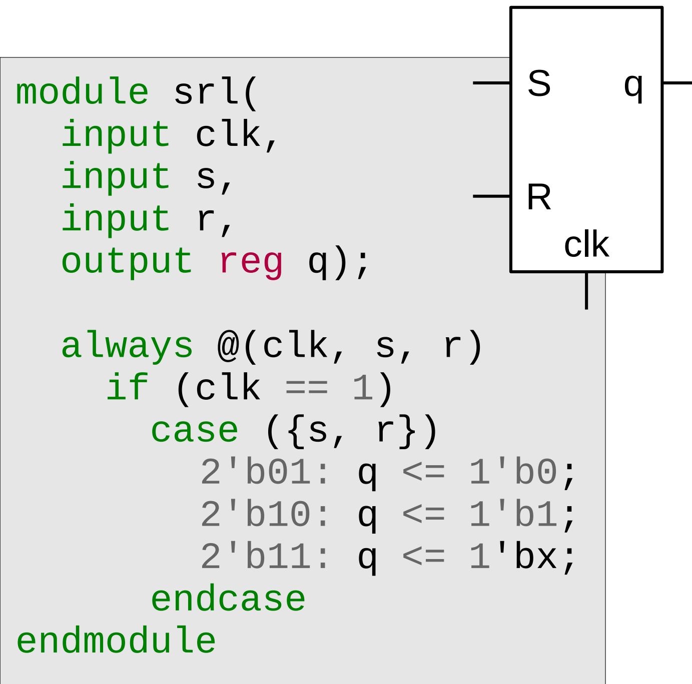


Synchronous latches

- In real circuits with thousands or million latches, it is very useful to control the state change so that it takes place in all devices at the same time.
- State change is “synchronized” to a “clock signal” (CLK)
- The clock signal is periodic with a fixed frequency: the clock frequency.
- Types of latches
  - Gated latches
    - State change is only allowed when CLK is either high (1) or low (0).
  - Edge-triggered latches (flip-flops)
    - State change is only allowed at the instant CLK changes from 0 to 1 (positive edge) or from 1 to 0 (negative edge)
    - State change is more precisely determined.
    - Make more robust and easy to design circuits

Synchronous latches

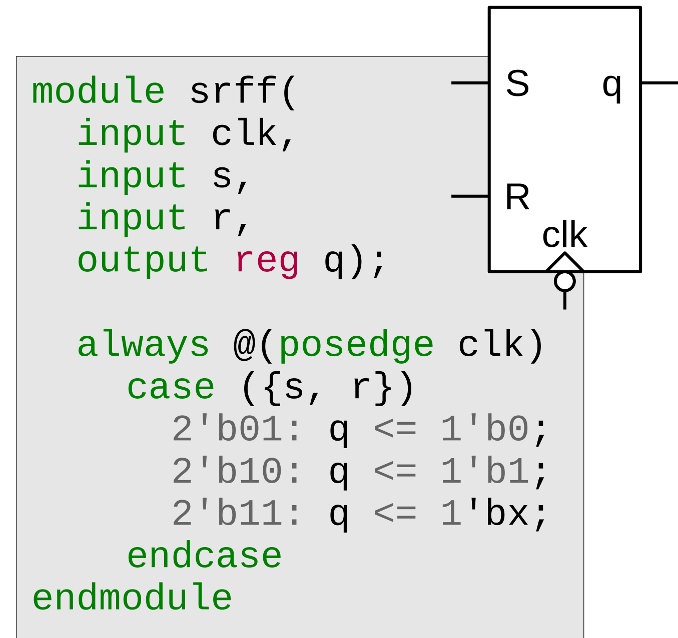
Gated latch




```
module srl(
  input clk,
  input s,
  input r,
  output reg q);

  always @(clk, s, r)
    if (clk == 1)
      case ({s, r})
        2'b01: q <= 1'b0;
        2'b10: q <= 1'b1;
        2'b11: q <= 1'bx;
      endcase
endmodule
```

State change when clk=1

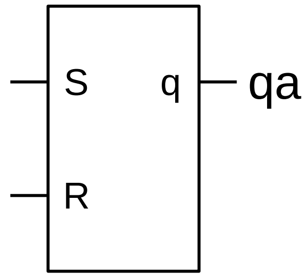
Flip-flop



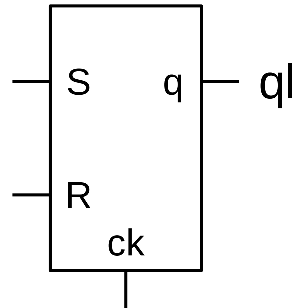

```
module srff(
  input clk,
  input s,
  input r,
  output reg q);

  always @(posedge clk)
    case ({s, r})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= 1'bx;
    endcase
endmodule
```

State change when clk changes from 0 to 1

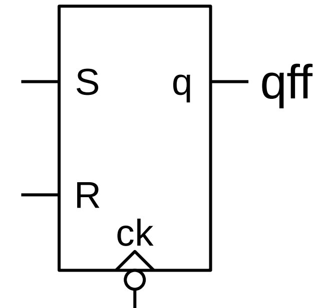
Synchronous latches



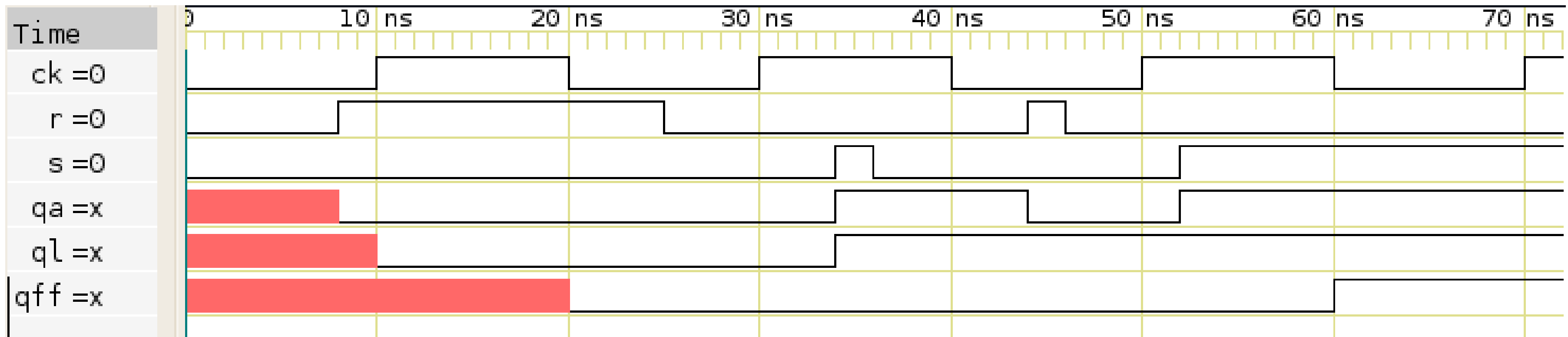
Asynchronous



Gated



Flip-flop



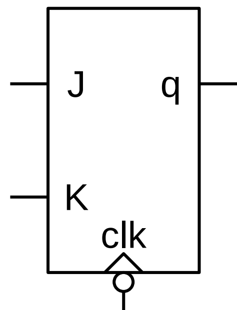
Types of flip-flops

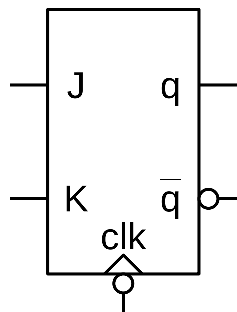
- SR
  - Simple design.
  - Undetermined state when S=R=1.
- JK
  - Similar to SR: J~S, K~R.
  - Toggle (reverse) function for J=K=1.
  - Very versatile.
- D
  - A single input equal to the next state.
  - Easy to use and implement.
  - Basic data (D) storage element.
- T
  - A single input to toggle the state
  - Very useful in some special applications: counters

JK flip-flop

Simulation

Symbols



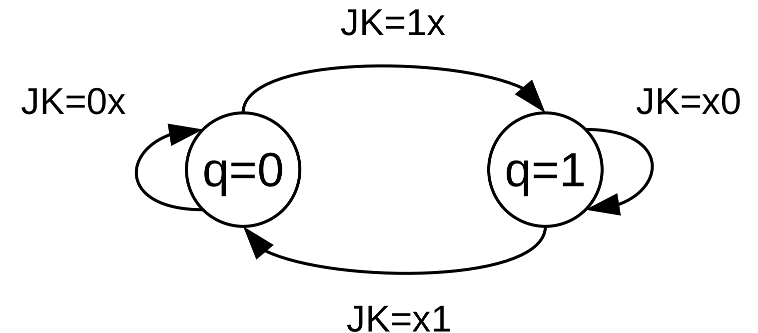


State table

q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

Q

State diagram



Excitation table

q → Q	JK
0 → 0	0x
0 → 1	1x
1 → 0	x1
1 → 1	x0

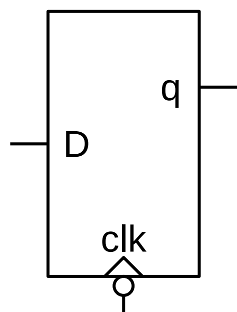
Verilog

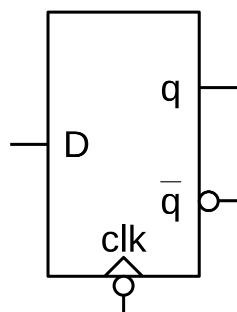
```
module jkff(
  input clk,
  input j,
  input k,
  output reg q);

  always @(negedge clk)
    case ({j, k})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= ~q;
    endcase
endmodule
```

Simulation

D flip-flop

Symbols



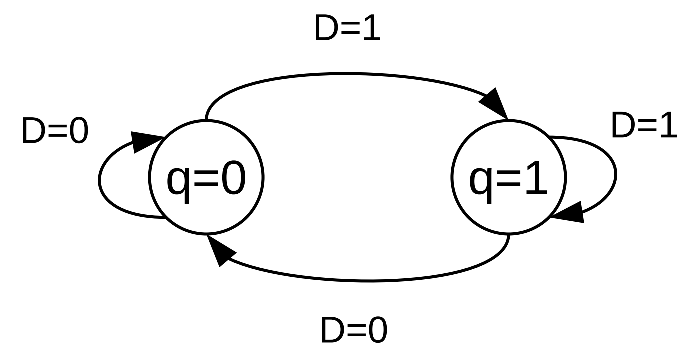


State table

q \ D	0	1
0	0	1
1	0	1

Q

State diagram



Excitation table

q → Q	D
0 → 0	0
0 → 1	1
1 → 0	0
1 → 1	1

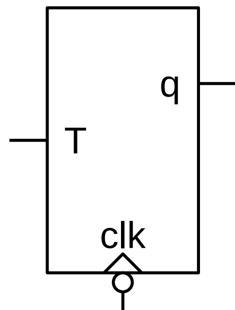
Verilog

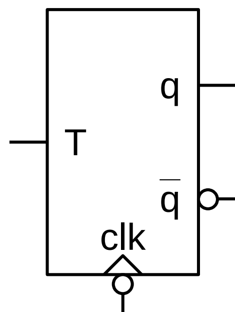
```
module dff(
  input clk,
  input d,
  output reg q);

  always @(negedge clk)
    q <= d;

endmodule
```

Simulation

T flip-flop

Symbols



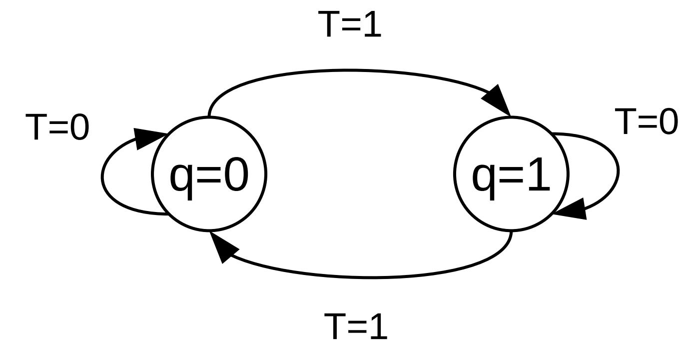


State table

q \ T	0	1
0	0	1
1	1	0

Q

State diagram



Excitation table

q → Q	T
0 → 0	0
0 → 1	1
1 → 0	1
1 → 1	0

Verilog

```
module tff(
  input clk,
  input t,
  output reg q);

  always @(negedge clk)
    if (t == 1)
      q <= ~q;

endmodule
```

About latch/flip-flop usage

Technology	SR	JK	D	T	Notes
7400 family		X	X		JK as general-purpose flip-flop and D as storage element.
FPGA			X		CLB regularity. Good for storage and registers (will see).
ASIC	X	X	X	X	All types available depending on application.

Latch/flip-flop questions

- Does it make sense an asynchronous JK, D or T latch?

No way.

- And a gated JK, D or T latch?

D: yes.
JK: what if J=K=1, CLK=1?
T: what if T=1, CLK=1?

- Can we design a flip-flop using another type of flip-flop?

Absolutly!

- Can it be done systematically?

Yes. Calculate the excitations to obtain the transition of the target FF.

- Is it useful?

Yes. Eg. the latch available in your technology is different from what you need.

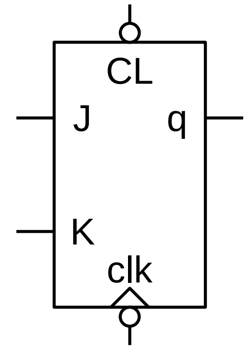
Obtaining a flip-flop from another

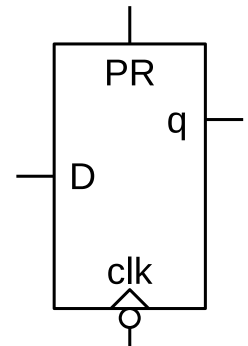
Example 2

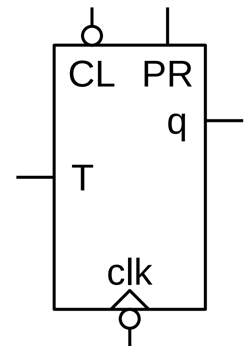
Design SR, D and T flip-flops by using a JK flip-flop (and gates).

Asynchronous inputs

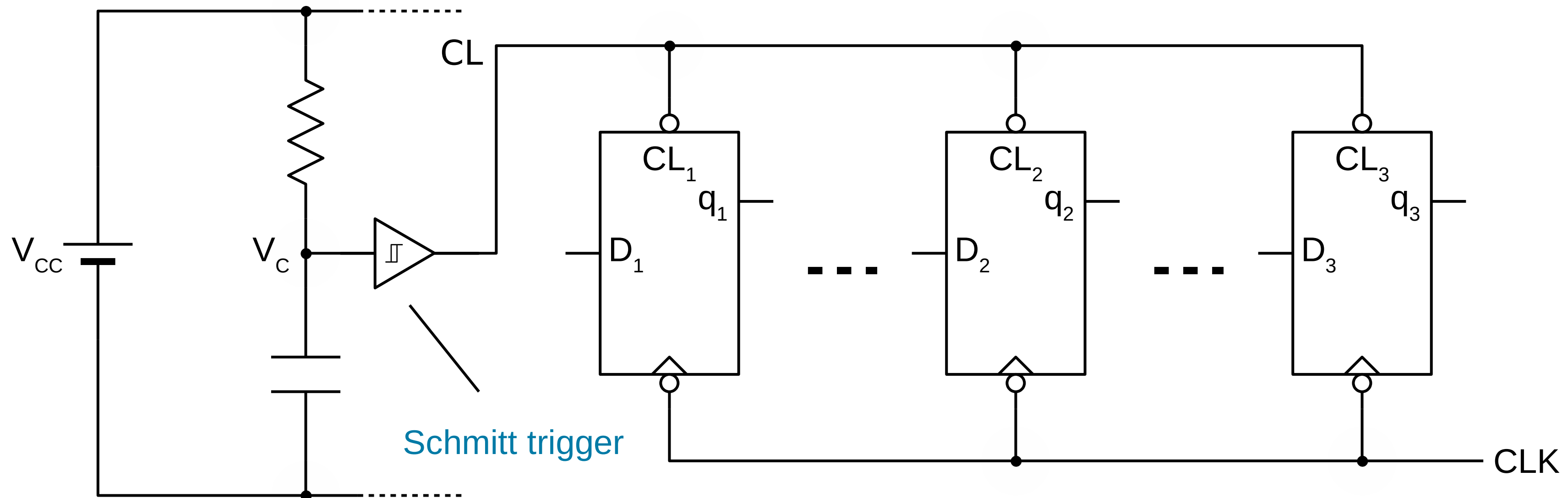
- Easy way to force a given state
  - CL (clear): set to 0
  - PR (preset): set to 1
- Immediate effect after activation:
  - Active low (0)
  - Active high (1)
- Higher priority than synchronous inputs
  - J, K, D, T, ...
- Solution to the problem of initiating the state in complex digital systems
  - Millions of flip-flops
  - Need to start from a known state

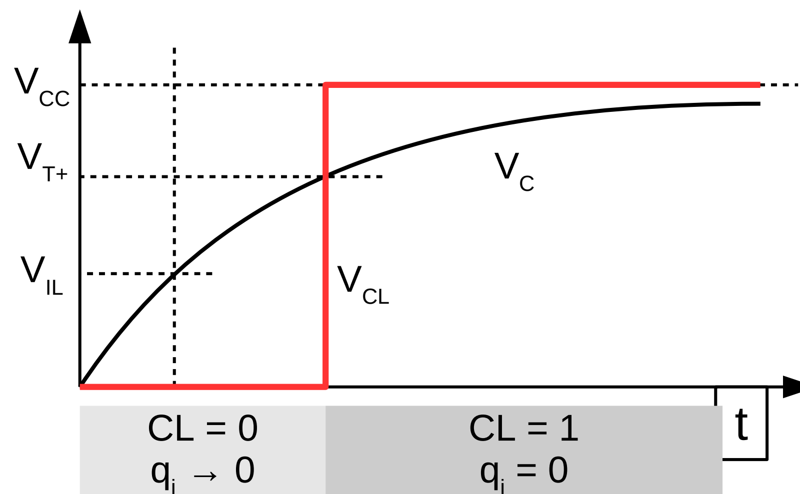




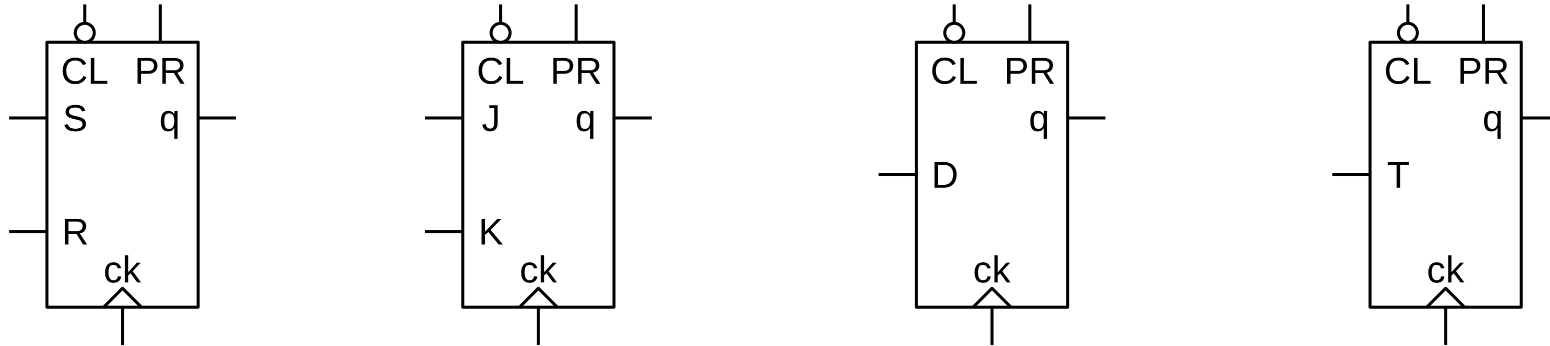


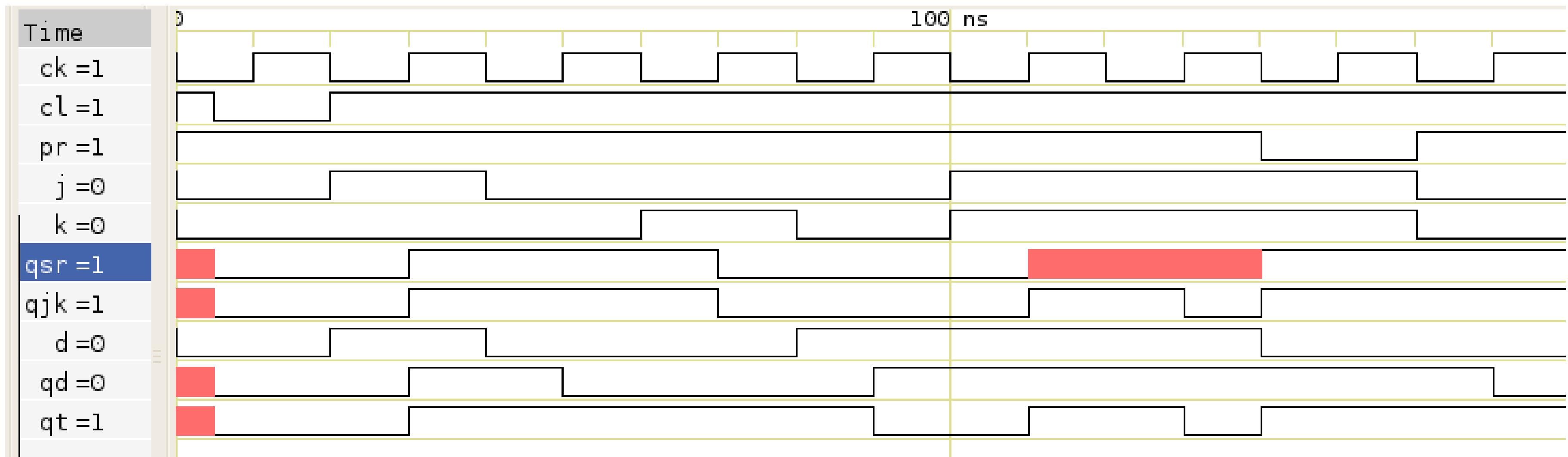
Asynchronous inputs
Sample startup circuit with schmitt trigger





Asynchronous inputs

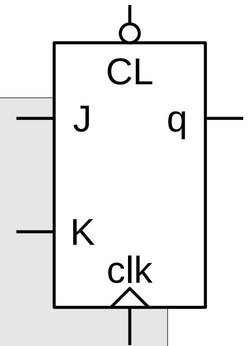




S=J, R=K, T=D

Asynchronous inputs. Verilog example

- JK flip-flop
  - Rising edge-triggered
  - Active-low clear (CL)
- Why “negedge cl”?

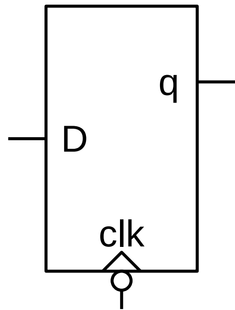



```
module jkff(
  input clk,
  input j,
  input k,
  output reg q);

  always @(posedge clk, negedge cl)
    if (cl == 1'b0)
      q <= 1'b0;
    else
      case ({j, k})
        2'b01: q <= 1'b0;
        2'b10: q <= 1'b1;
        2'b11: q <= ~q;
       endcase
endmodule
```

Timing restrictions

- Like logic gates, flip-flops present a delay from the clock edge to the output: t_{CQ}
- Synchronous inputs should not change close to the active edge of the clock signal to avoid an unpredictable state change.
  - Set-up time (t_s)
    - Time **before** the active edge in which inputs should not change.
  - Hold time (t_h)
    - Time after the active edge in which inputs should not change.
- An input change in the forbidden zone should be avoided:
  - The output may change to an undetermined state.
  - The output may take an undetermined time to change (metastable state).

Timing restrictions example



t_s t_h

CLK

D

t_{CQ}

q Q=1

D

?

q ? Q=?

D

?

q ? Q=?

Contents

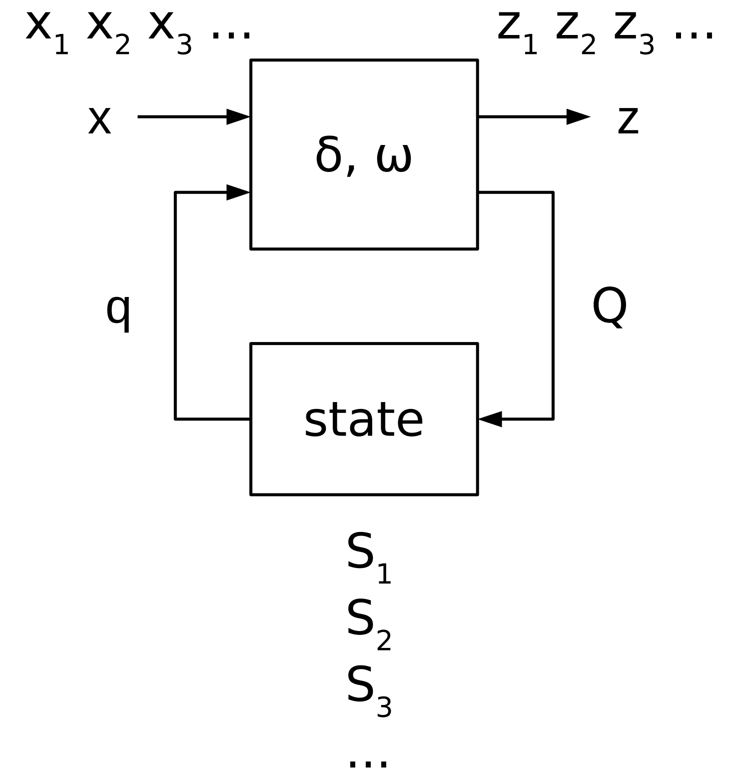
- Introduction
- Latches and flip-flops
- Synchronous Sequential Circuits (SSC) design
  - Synchronous sequential circuits
  - Finite state machines (FSM)
  - SSC design objective and procedure
  - Manual SCC design procedure
  - Mealy vs Moore and input synchronization
  - Automatic SCC design procedure using Verilog
- SSC analysis
- SCC application examples

Synchronous Sequential Circuits (SSC)

- SSC combine combinational elements and flip-flops.
- In general, the output value depends on the inputs and the internal state of the flip-flops.
- In a SSC all flip-flops are triggered in the same way by the same clock signal: they change states at the same time (synchronously).
- The synchronous restriction allows:
  - A more simple procedure to design sequential circuits.
  - The automation of the design process (done by automatic synthesis tools).
  - More robust circuits against device and signal variations.
- The clock frequency becomes a key performance factor:
  - Determines the number of operations per second that the circuit can do.
  - The maximum frequency is limited by the propagation delay of devices and interconnection lines, the maximum power that can be dissipated and other factors.

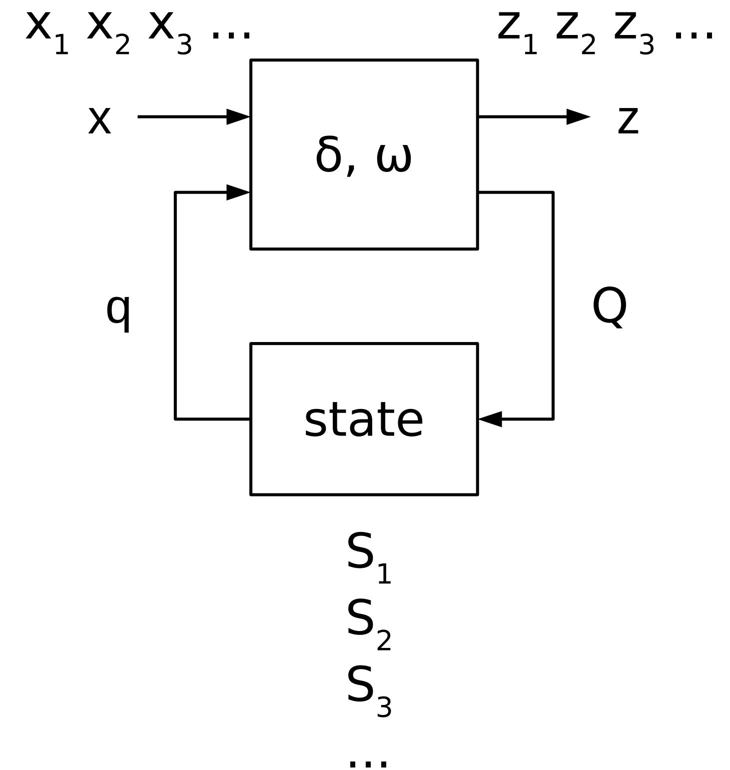
Finite State Machines (FSM)

- Finite State Machines are an useful tool to model many different problems, including problems that can be solved using Synchronous Sequential Circuits.
- A FSM is formed by:
  - A finite set of states (S)
  - A set of input symbols (Σ)
  - A set of output symbols (Γ)
  - A next state function (δ)
    - $Q = \delta(q, x)$
  - An output function (ω)
    - Mealy's machine: $z = \omega(q, x)$
    - Moore's machine: $z = \omega(q)$



Finite State Machine operation

- A new symbol (x) arrives to the input.
- A new output symbol is calculated.
  - $z = \omega(q, x)$
- The next state of the machine is calculated.
  - $Q = \delta(q, x)$
- The next calculated state becomes the current state.
- The cycle is repeated.

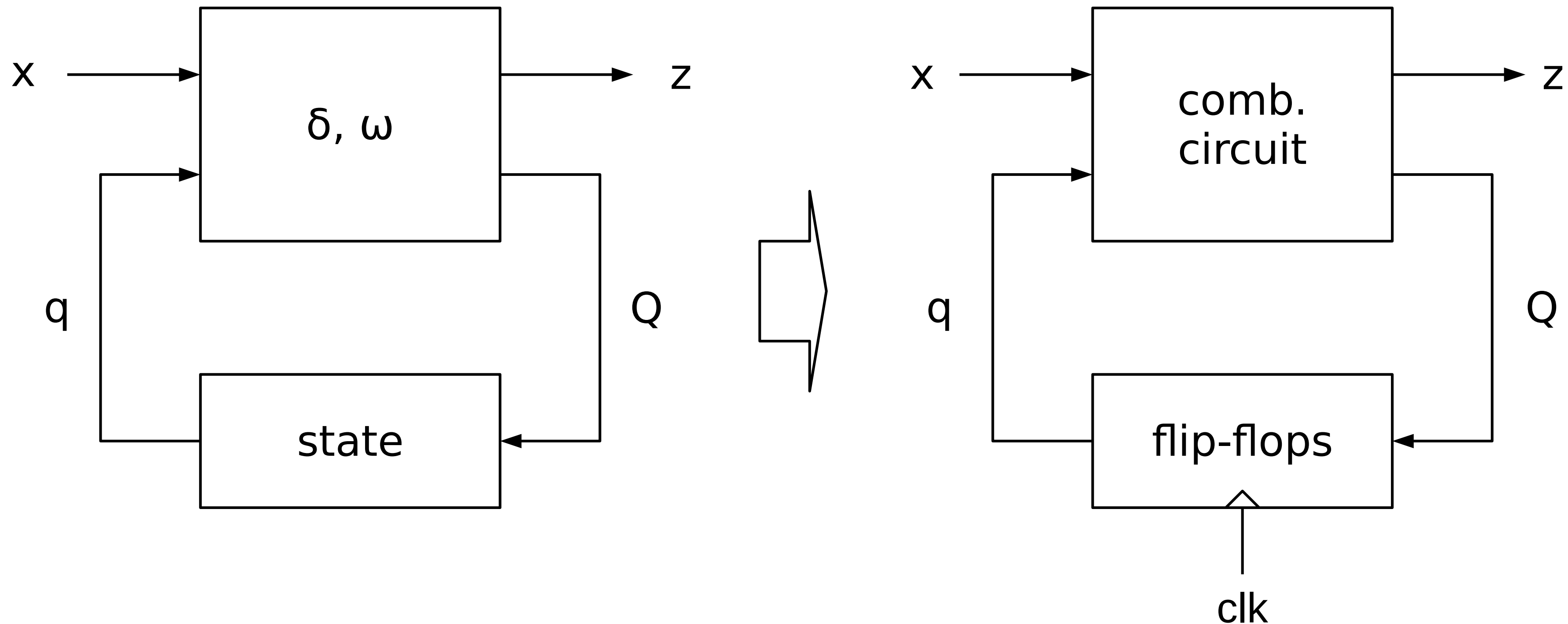


Finite State Machine properties

- Starting at the same state, deterministic FSM's always generate the same output sequence for the same input sequence.
- Two FSM's are equivalent if they generate the same output sequence for the same input sequence.
- For every Mealy's machine, there is an equivalent Moore's machine and vice-versa.
- FSM's can be optimized: an equivalent FSM with a reduced number of states.
- The state of the machine changes depending on the input sequence so the state represents the whole set of old input symbols (history of the machine)
- FSM's may not be completely specified: a next state may not be defined for a given current state and input value.

SSC design using FSMs

- FSMs can be implemented as a SSCs systematically.
  - The state of the FSM is stored with flip-flops in the SSC.
  - Next state and output functions of the FSM are implemented as logic functions using combinational circuits.



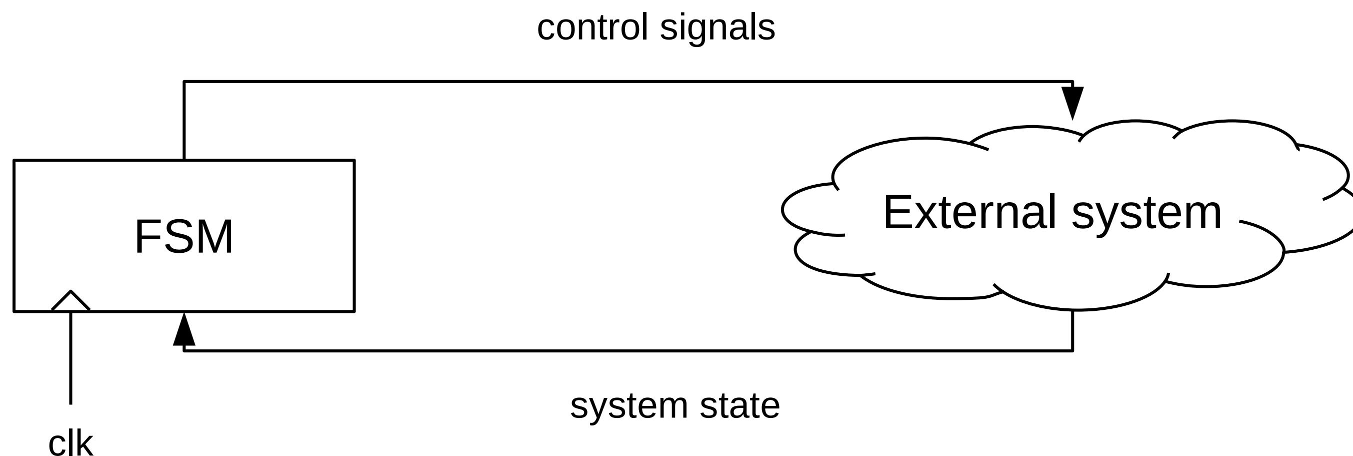
SSC Applications

- Sequence detectors
  - The output is activated when the input receives a specific sequence of symbols.
- Sequence generators
  - The output generates a fixed or variable sequence of symbols depending on the value of the inputs.

Many practical problems may be described as sequence detectors or generators

SSC Applications

- Control units
  - Inputs take information from an external system (system state, temperature, position, etc.)
  - Output signals activate operation in the external system (heating, movement, etc.)
  - The FSM implements the control algorithm.
  - The frequency of the clock signal determines the frequency at which the external system is sampled and actions are taken.



SSC Applications

- Sequential calculations
  - The output is the result doing some calculations on the input data.
  - Sometimes calculations are performed one bit at a time, using as many clock cycles as necessary.
  - It is an alternative of doing the operation on all bits at the same time using a combinational circuit.
  - Why do sequential calculations instead of combinational?
    - Saves hardware: the same circuit can operate many bits, one bit at a time.
    - Drawback: needs more time than a combinational circuit.
    - Sometimes bits arrive one at a time (serial communication, streaming): calculations can be done as bits arrive.
  - Calculations example:
    - Parity calculation.
    - Sequential arithmetic operations.
    - Sequential coding and decoding.
    - Etc.

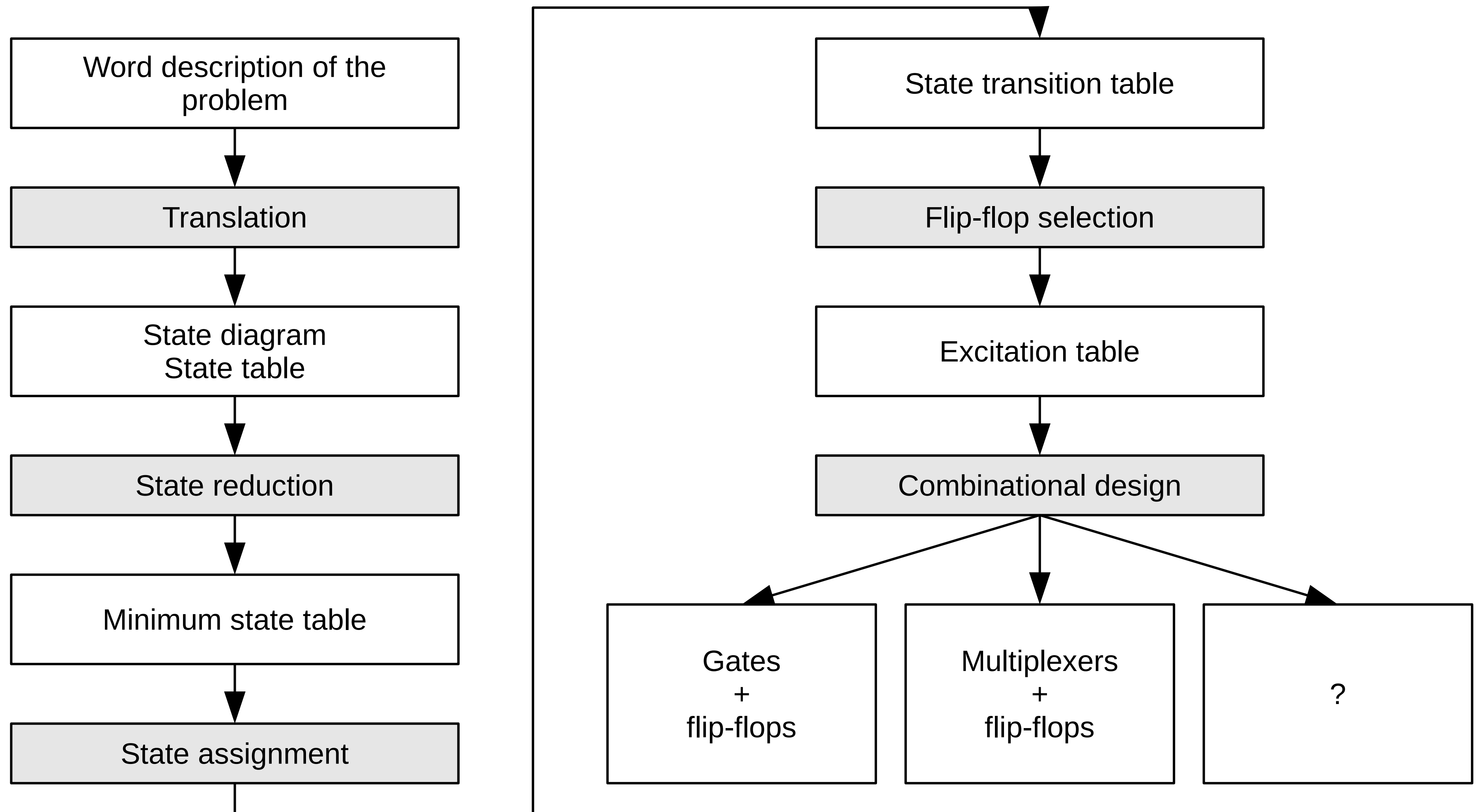
SSC design objective

- Objective
  - Define a FSM that solves a given problem.
  - Implement the FSM using a synchronous sequential circuits (SSC).
- Cost criteria
  - Minimizing the number of memory elements
  - Minimizing the number of devices
  - Operation frequency
  - Energy consumption
  - Etc.
- Need to compromise different criteria
- Procedures
  - Manual: using pen and paper.
  - Automatic synthesis: using CAD tools starting with a HDL description

Manual SSC design procedure

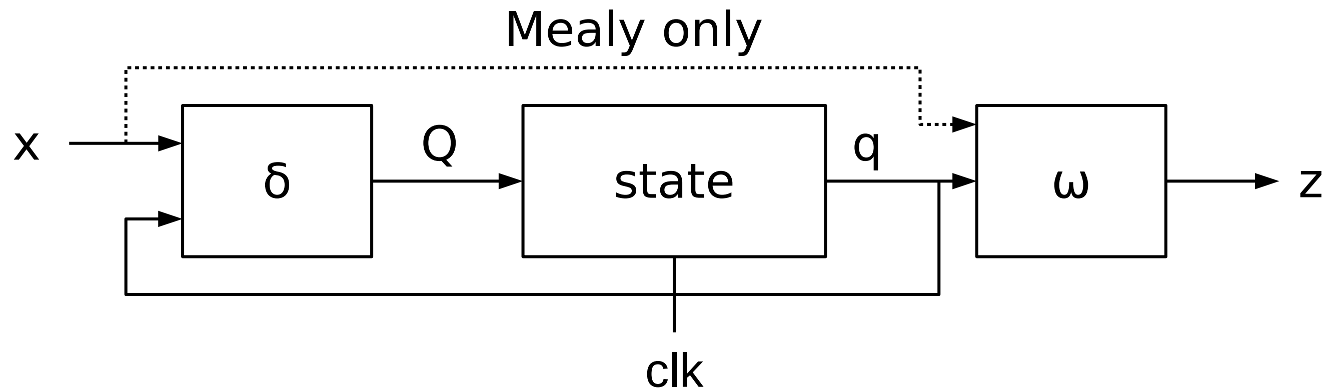
- Hand process
  - Can be done with paper and pencil.
  - Starts with a description of the problem using a state diagram or a state table.
  - The states table is transformed in different steps to lead to a circuit representation.
- Design process using Computer Aided Design (CAD) tools
  - The problem is translated to a formal description using a hardware description language.
  - Simulation tools are used to check that the operation of the described system is correct.
  - Automatic synthesis tools are used to implement the actual circuit.

Manual SSC design procedure



Mealy vs. Moore

- Mealy
  - Different output at the same state is possible: may save states.
  - Output is generated as soon as the input is available: less latency.
- Moore
  - Output always synchronized to the clock: less timing failures.
  - The output only depends on the state: more simple output functions.



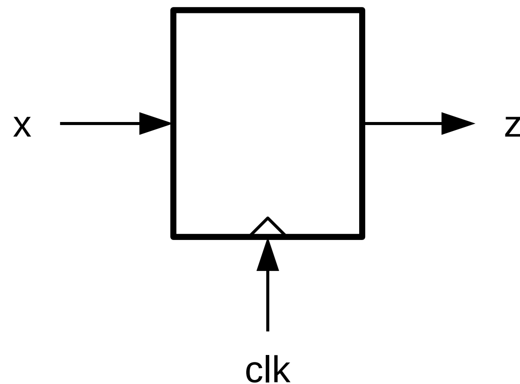
Mealy's FSM

Example 3

Design a synchronous sequential circuit with a 1-bit input 'x' and a 1-bit output 'z' that detects when the sequence of bit '1001' arrives at the input. When the sequence is detected, 'z' will be set to '1' for one clock cycle and will remain '0' otherwise.

The sequences may start anytime and may overlap (the last '1' in a sequence may be the first '1' in the next sequence).

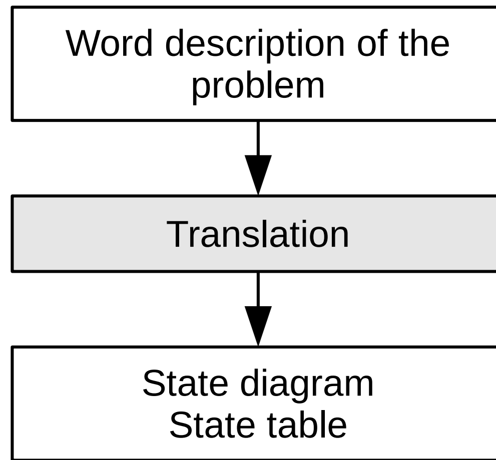
Design the circuit using a Mealy FSM and implement it with logic gates and JK flip-flops.



Sample sequence and expected output

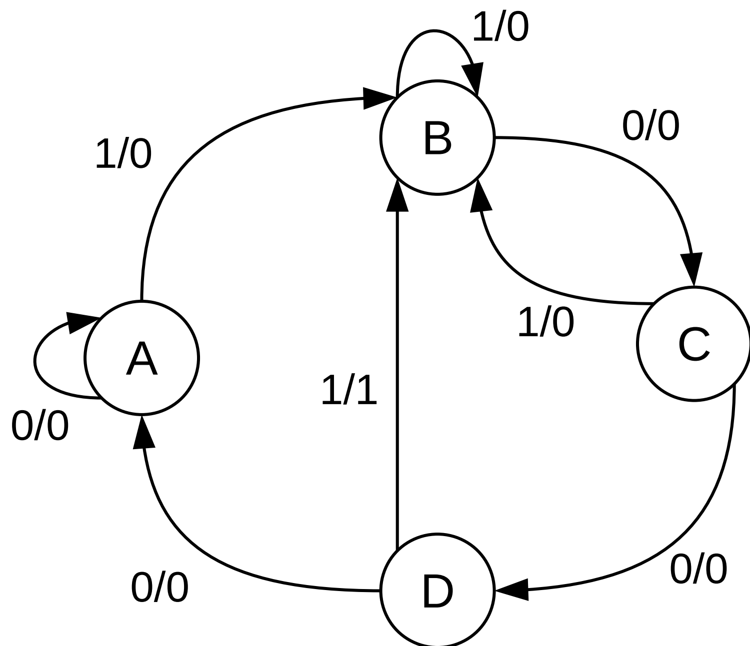
```
x: 00100111000110111001001001010011...
z: 00000100000000000001001001000010...
```

Translation



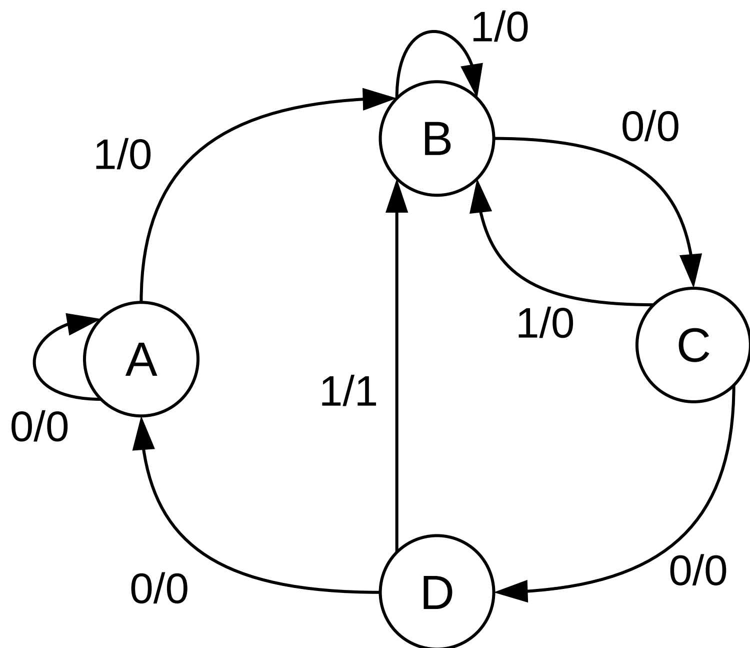
- Most important part of the design process
- Non-systematic
- Clues/advice:
  - Clearly define inputs and outputs
  - Select Mealy or Moore depending on the problem: output synchronization, …
  - Define appropriate states in the most general way: similar states may be the same state.
  - Calculate the necessary transitions and output values.
  - Capture all the details of the problem in the state diagram/table.
  - Check the diagram/table with a sample input sequence.

Translation. State diagram



- Nodes
  - Represent states
  - Use intuitive names (sometimes)
    - {A, B, C, …}
    - {S0, S1, S2, …}
    - {wait, start, receiving, ...}
- Arcs
  - Represent possible state transitions from every state (S).
  - Named as x/z, where:
    - x: input value that triggers the transition from state S.
    - z: output value of the machine when in state S and input is x.

Translation. State diagram

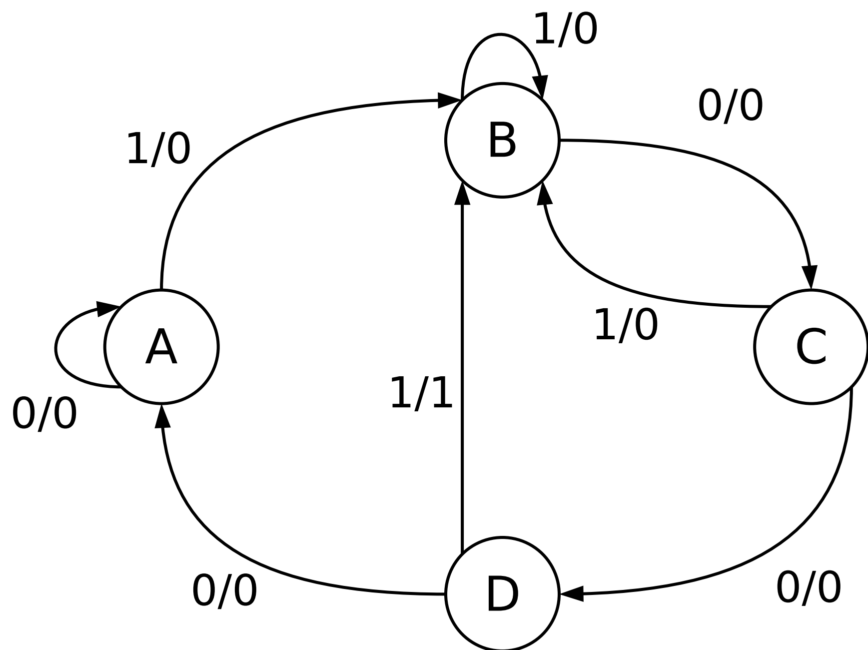


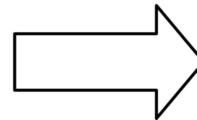
- A: waiting for first bit of sequence ('1')
  - While the input is '0', stay at A and keep the output at '0'.
- B: first bit ok, waiting for a '0'.
  - If the input is '0', move to next state. It it is '1', the second bit is not correct, but is a correct first bit.
- C: second bit ok, waiting for a '0'.
  - If the input is '1', move to the next state. If it is '1' move to B because it is a correct first bit.
- D: third bit correct, waiting for a '1'.
  - If the input is '0', the sequence is not correct: output a '0' and start again at A. If the input is '1', the sequence is correct: output a '1' and move to B because it is also a correct first bit and we are considering overlapping.

Translation. States table

- Double input table (rows and columns) with information equivalent to the states diagram.
  - Rows: possible states.
  - Columns: possible input values.
  - In each cell: corresponding next state and output.
- Each node in the diagram and the arcs starting at that node correspond to a row in the states table.
- Converting a states diagram to a states table and vice-versa is trivial.

Translation. States table

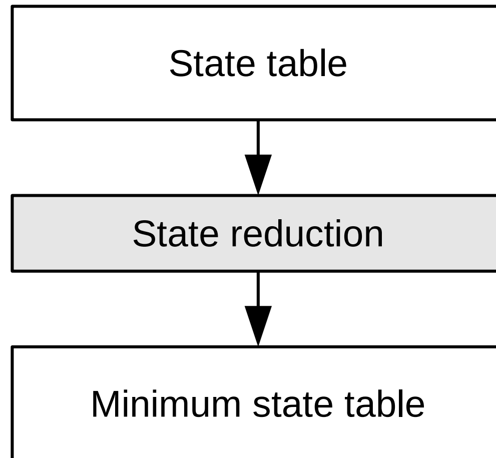




S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

NS,z

States reduction



- Objective:
  - Eliminate redundant states
  - Reduce the cost in flip-flops and combinational logic.

Equivalent states:

Two states p and q are equivalent if any input sequence that is applied to the machine starting at state p produces exactly the same output sequence when applied starting at state q.

Two states p and q are equivalent if and only if:
•All the next states to p and q are identical or equivalent for every input value.
•All the output values are identical for every input value.

In a minimum state table there are no equivalent states.

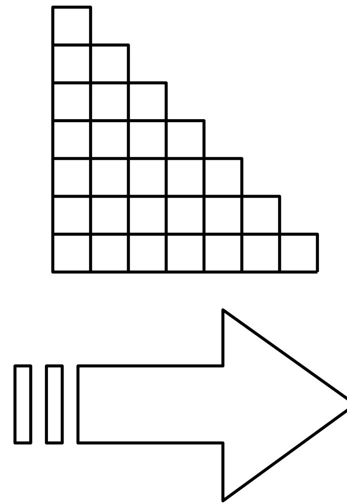
States reduction. Procedure

- Starting with the state table, possible compatible states are identify by comparing all the possible state pairs.
- The **compatible states table** helps identifying compatible states and the compatibility conditions.
- Once identified all the compatible states, the compatible states are grouped in equivalence classes.
- A new state table is written by selecting one representative from every class.

States reduction

S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

Q,z

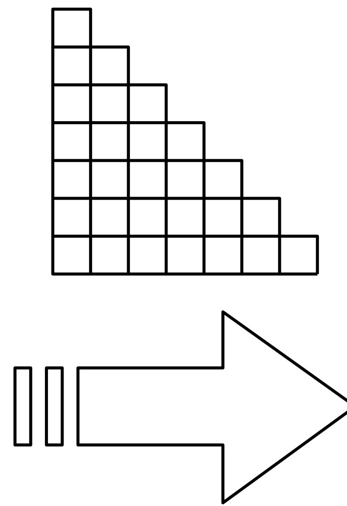


Minimum states table already!

States reduction (Example 4)

S \ x	0	1
A	B,0	C,0
B	D,0	E,0
C	G,0	E,0
D	H,0	F,0
E	G,0	A,0
F	G,1	A,0
G	D,0	C,0
H	H,0	A,0

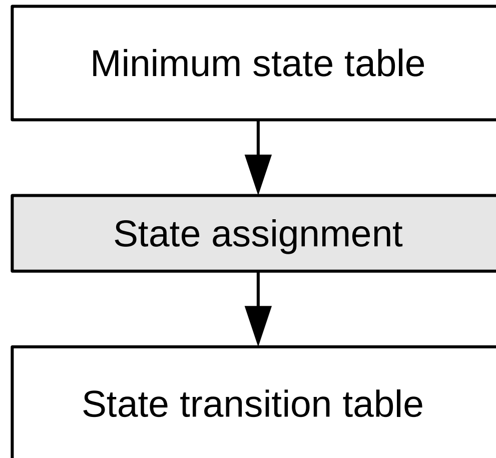
NS, z



S \ x	0	1
a	b,0	a,0
b	d,0	a,0
d	h,0	f,0
f	b,0	a,0
h	h,0	a,0

NS, z

States assignment



- Objective:
  - Assign binary values to states (state coding).
  - Allow state storage in flip-flops.
- Coding selection:
  - Affects the final result: number of devices, size, speed, energy consumption, ...
  - Selection depends on cost criteria.
- Typical options
  - Complex assignment algorithms: optimize the final circuit (no. of devices, speed, cost, etc.)
  - Arbitrary (random) assignment: in simple circuits or when cost/optimization is not important.
  - One flip-flop per state (one-hot encoding): used when minimizing the number of flip-flops is not important. Typically used in FPGA's.
  - Gray encoding: used when states are normally reaches in sequence. Reduces signal transitions and power consumption.

States assignment (Gray encoding)

Gray encoding: “contiguous” states have “adjacent” codes. Minimizes the number of signal transitions.

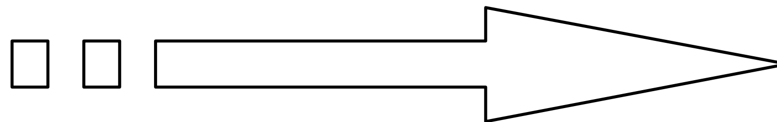
States table

S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

NS,z

State assignment

S	q_1q_0
A	00
B	01
C	11
D	10

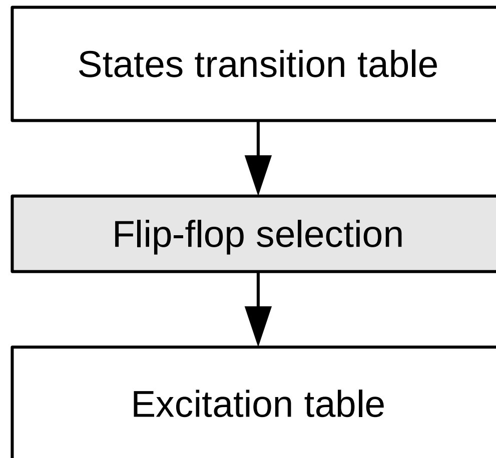


States transition table

q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q_1Q_2,z

Flip-flop selection



- Objective
  - Select the type of flip-flops that will be used to store the codified state.
- Options:
  - JK: reduces the cost of the combinational part but needs two control inputs (more connections).
  - RS: more simple than JK internally but less flexible.
  - D: simplify the design and reduces the number of connections (one control input).
  - T: easy to toggle the state. Specially useful when state toggling is frequent.
- Restrictions
  - The selection depends on the available technology.
    - MSI (74xx): JK may reduce the number of chips.
    - FPGA: only D FF available, typically.
    - ASIC: any type available. Depends on application.

Flip-flop selection. Eg. JK

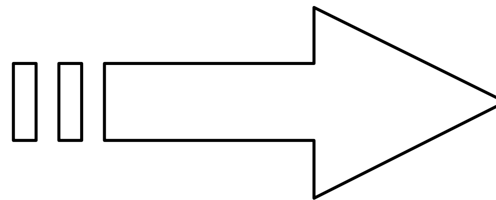
States transition table

q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q_1Q_2,z

JK excitation table

q → Q	JK
0 → 0	0x
0 → 1	1x
1 → 0	x1
1 → 1	x0



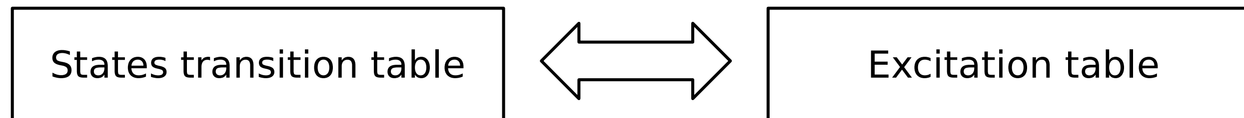
Excitation table

q_1q_2 \ x	0	1
00	0x,0x,0	0x,1x,0
01	1x,x0,0	0x,x0,0
11	x0,x1,0	x1,x0,0
10	x1,0x,0	x1,1x,1

J_1K_1,J_2K_2,z

Flip-flop selection. Eg. D

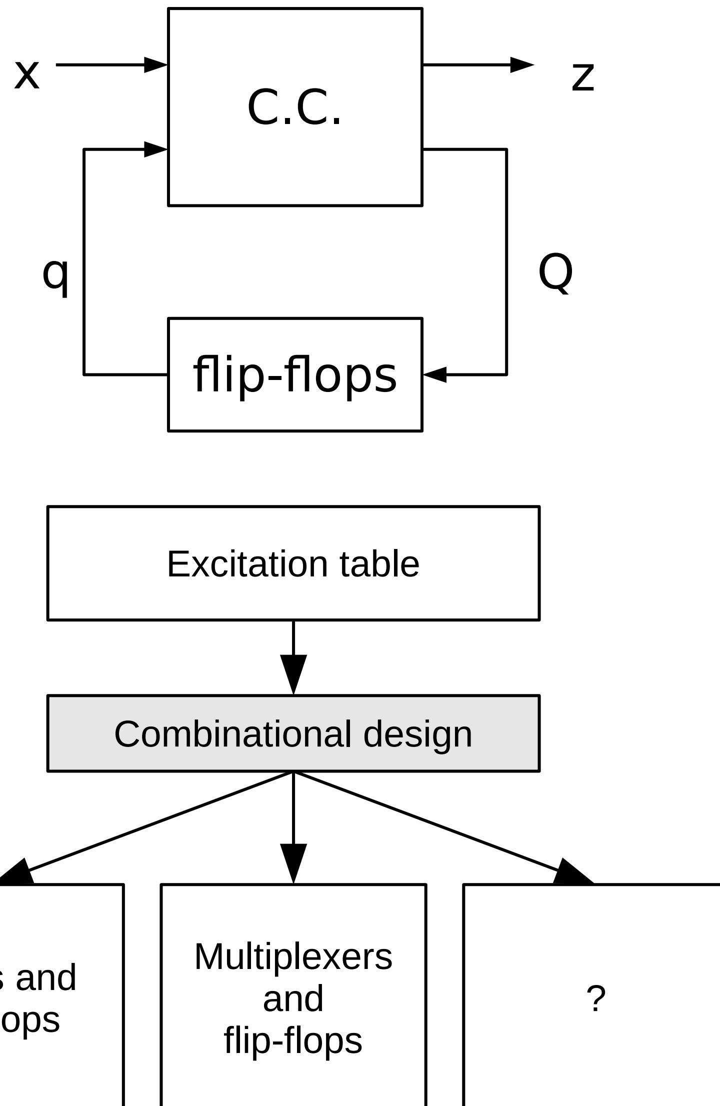
- Flip-flop D:
  - Next state is equal to D
  - Excitation to achieve next state Q is: D = Q



q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q_1Q_2,z
D_1D_2,z

Combinational part design



- The excitation table is a specification of the combinational part.
- The implementation of the combinational part may use any techniques available for combinational circuits:
  - Two-level logic gate design (K-map).
  - Decoders + logic gates.
  - Multiplexers.
  - Others.

Combinational part design using logic gates

q_1q_2 \ x	0	1
00	0x,0x,0	0x,1x,0
01	1x,x0,0	0x,x0,0
11	x0,x1,0	x1,x0,0
10	x1,0x,0	x1,1x,1

J_1K_1,J_2K_2,z

J_1:

q_1q_2 \ x	0	1
00	0	0
01	1	0
11	x	x
10	x	x

K_1:

q_1q_2 \ x	0	1
00	x	x
01	x	x
11	0	1
10	1	1

z:

q_1q_2 \ x	0	1
00	0	0
01	0	0
11	0	0
10	0	1

J_2:

q_1q_2 \ x	0	1
00	0	1
01	x	x
11	x	x
10	0	1

K_2:

q_1q_2 \ x	0	1
00	x	x
01	0	0
11	1	0
10	x	x

$J_1 = \overline{x}q_2$

$K_1 = \overline{x}+q_2$

$J_2 = x$

$K_2 = \overline{x}q_1$

$z = xq_1\overline{q_2}$

Circuit implementation

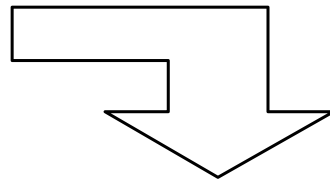
$J_1 = \overline{x}q_2$

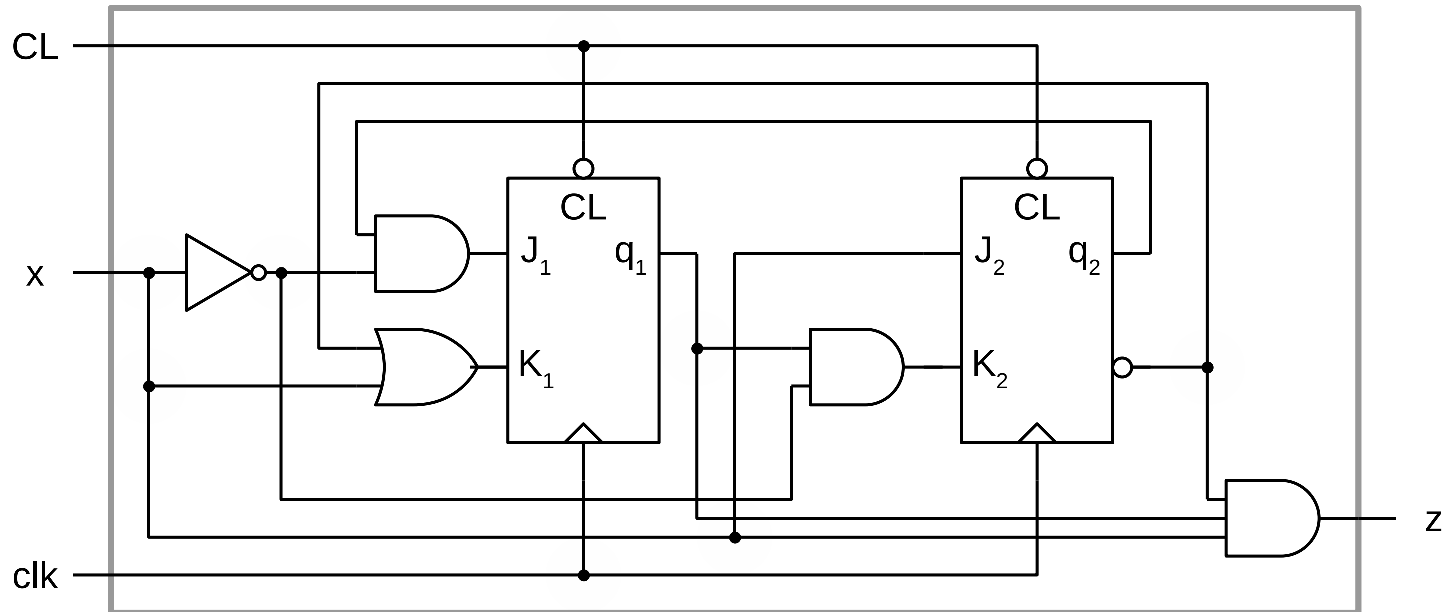
$K_1 = \overline{x}+q_2$

$J_2 = x$

$K_2 = \overline{x}q_1$

$z = xq_1\overline{q_2}$





Manual SSC design procedure
Example 3 summary

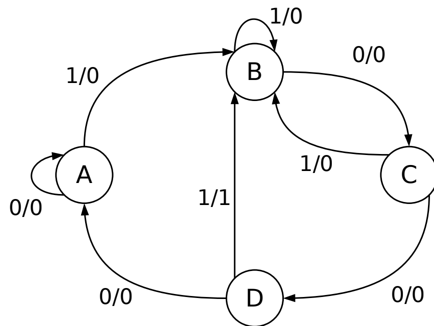
States diagram

States table

States transition table

Word description

Bla, bla, bla, ...



States table (NS,z)

S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

States transition table (Q_1Q_2,z)

q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Excitation table

Excitation/output equations

Circuit: Gates + flip-flops

Excitation table (J_1K_1,J_2K_2,z)

q_1q_2 \ x	0	1
00	0x,0x,0	0x,1x,0
01	1x,x0,0	0x,x0,0
11	x0,x1,0	x1,x0,0
10	x1,0x,0	x1,1x,1

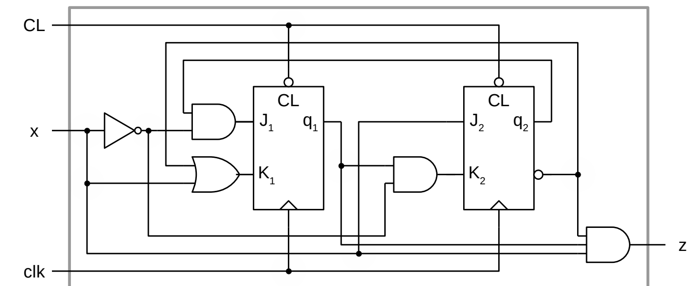
$J_1 = \overline{x}q_2$

$K_1 = \overline{x}+q_2$

$J_2 = x$

$K_2 = \overline{x}q_1$

$z = xq_1\overline{q_2}$



Examples

Example 5

Implement the combinational part of example 3 using multiplexers. Draw the complete circuit.

Example 6

Implement the FSM in example 3 using D flip-flops and:

a) logic gates.

b) multiplexers.

Moore's FSM

Example 7

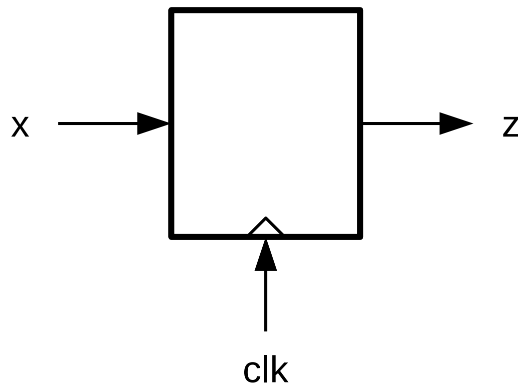
Design a synchronous sequential circuit with a 1-bit input 'x' and a 1-bit output 'z' that detects when the sequence of bit '1001' arrives at the input. When the sequence is detected, 'z' will be set to '1' for one clock cycle and will remain '0' otherwise.

The sequences may start anytime and may overlap (the last '1' in a sequence may be the first '1' in the next sequence).

Design the circuit using a Moore's FSM. Implement it using:

a) JK flip-flops and logic gates.

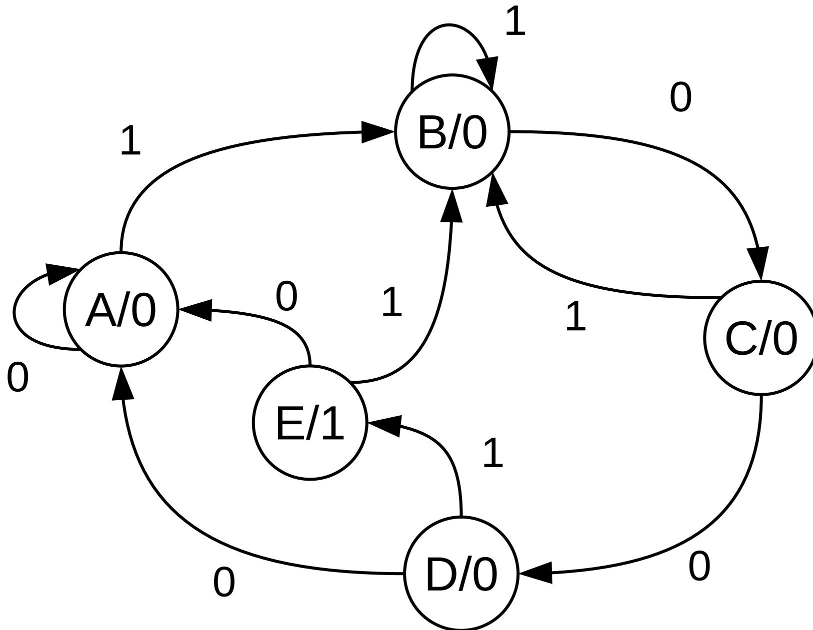
b) D flip-flops and multiplexers



Sample sequence and expected output

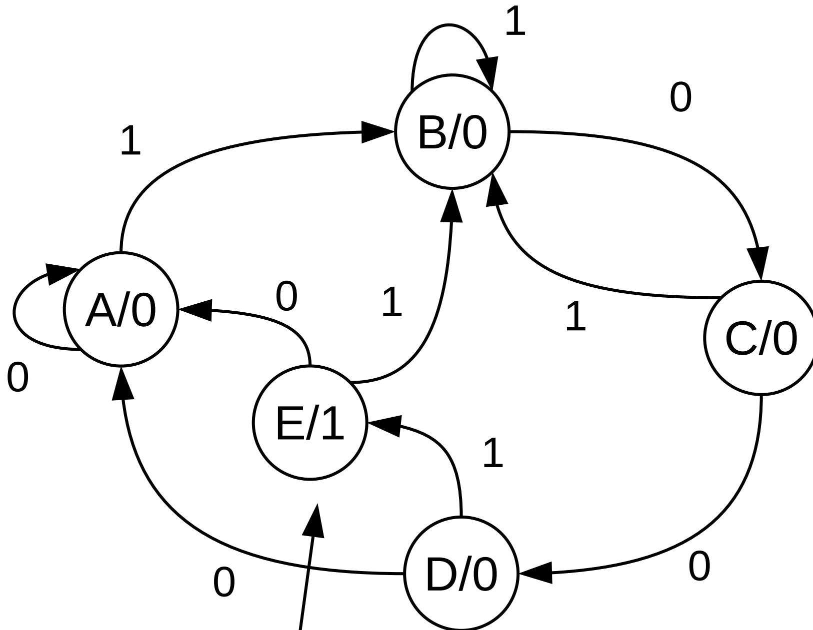
```
x: 001001110001101110010010010100110...
z: 000000100000000000001001001000010...
```

States diagram. Moore



- Nodes
  - Represent states
  - Use intuitive names (sometimes)
    - {A, B, C, …}
    - {S0, S1, S2, …}
    - {wait, start, receiving, ...}
  - Each node includes the output value corresponding to each state.
- Arcs
  - Represent possible state transitions from every state (S).
  - Named as 'x', where 'x' is the input value that triggers the transition from state S.

States diagram. Moore



We need a new state to activate the output, that we did not need in the Mealy's machine.

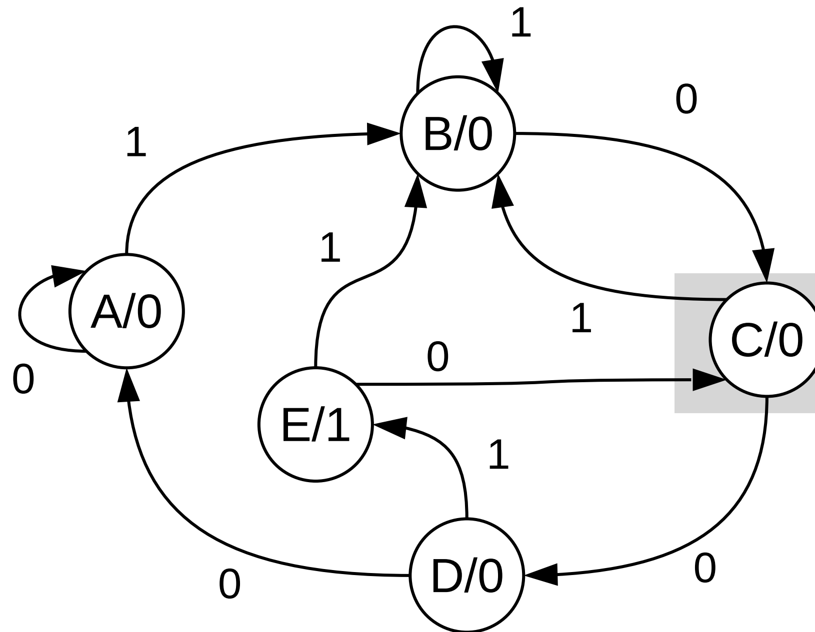
- A: waiting for first bit ('1')
- B: 1st bit correct, waiting for '0'
- C: 2nd bit correct, waiting for '0'
- D: 3rd bit correct, waiting for '1'
  - If the input is '0', the sequence is incorrect and go back to A.
  - If the input is '1', move to E to activate the output.
- E: activate the output
  - If the input is '0', move to A and start again.
  - If the input is '1', first bit is correct so move to B.

Are there any errors? Check with the sample sequence and expected output.

States table. Moore

- Double input table (rows and columns) with information equivalent to the states diagram.
  - Rows: possible states.
  - Columns: possible input values.
  - Output associated to every state in last column because this time the output does not depend on the input. Optionally, output value in each cell like Mealy.
- Each node in the diagram and the arcs starting at that node correspond to a row in the states table.
- Converting a states diagram to a states table and vice-versa is trivial.

States table. Moore



S \ x	0	1	z
A	A	B	0
B	C	B	0
C	D	B	0
D	A	E	0
E	C	B	1

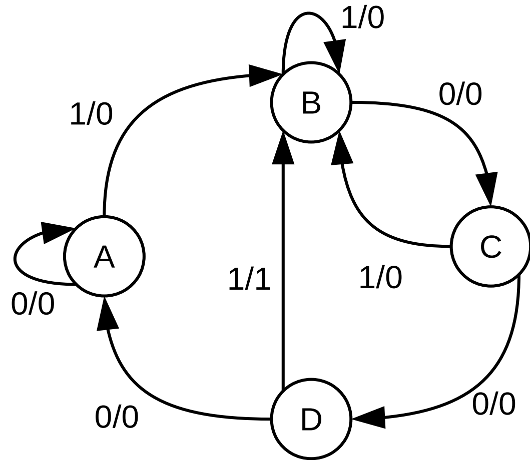
NS

Implementation

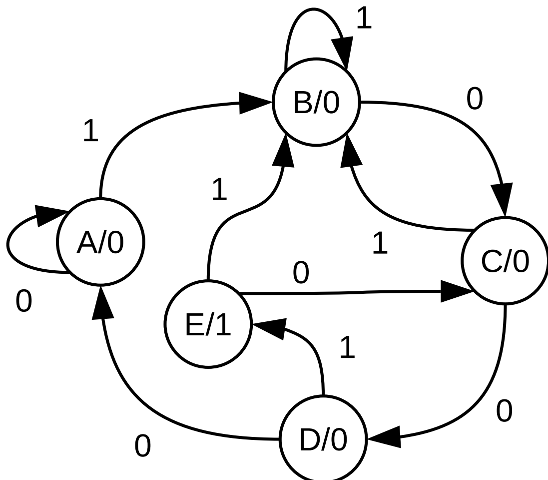
- Example 7
  - a) JK flip-flops and logic gates.
  - b) D flip-flops and multiplexers.

- Questions
  - How many flip-flops do we need?
  - What happen with the non-assigned flip-flop states?
  - May the machine get stuck in a non-assigned flip-flop state? (blocking states).
  - Can these blocking states be resolved?

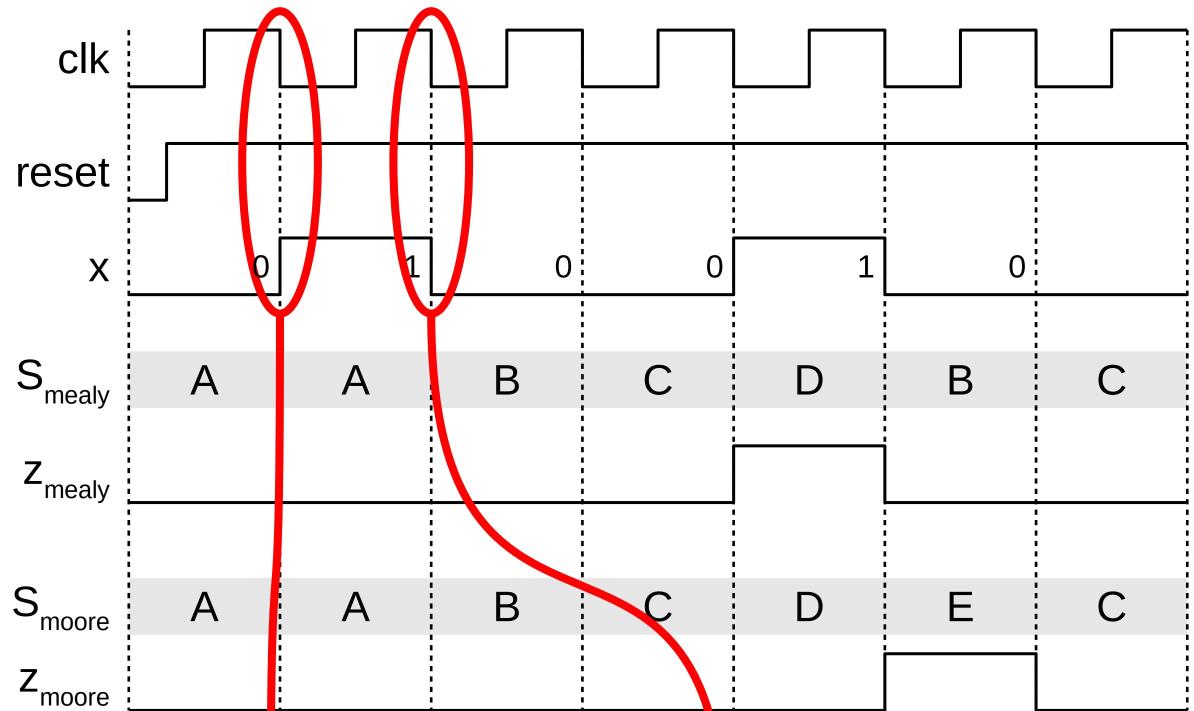
Mealy vs Moore
Synchronized inputs

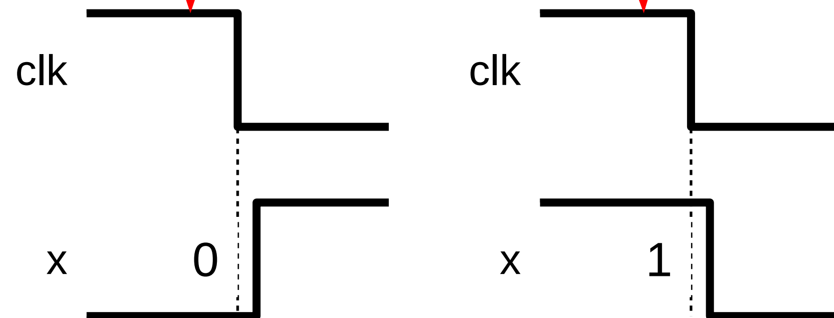


Mealy

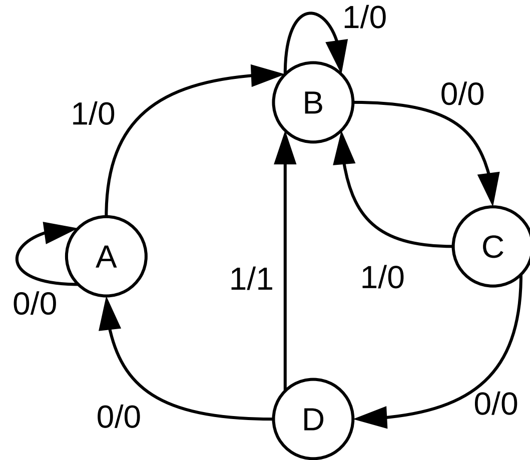


Moore

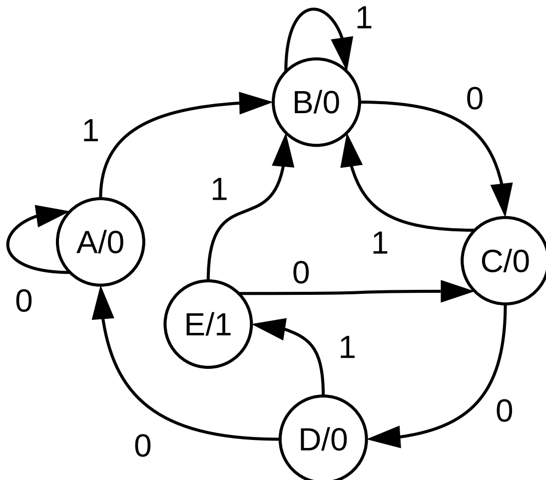




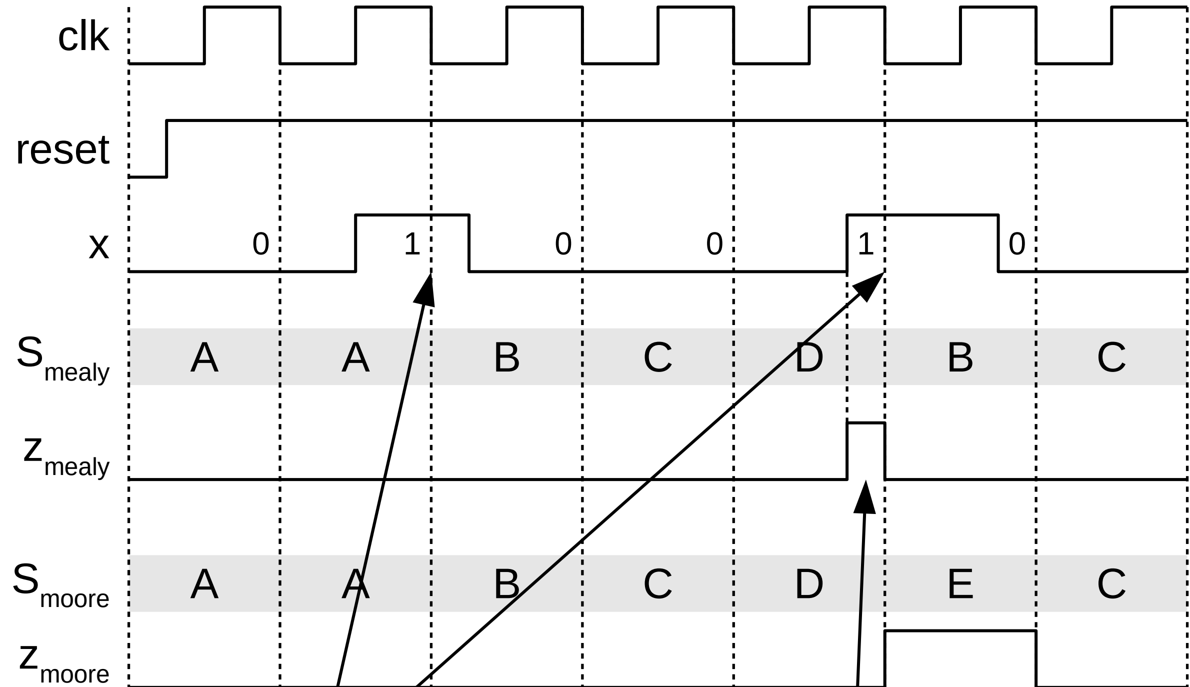
Mealy vs Moore
Non-synchronized inputs



Mealy



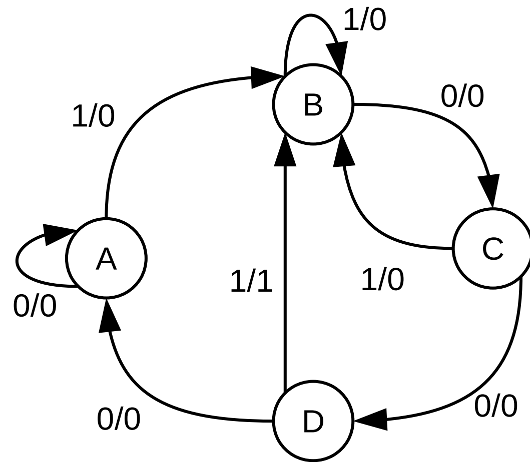
Moore



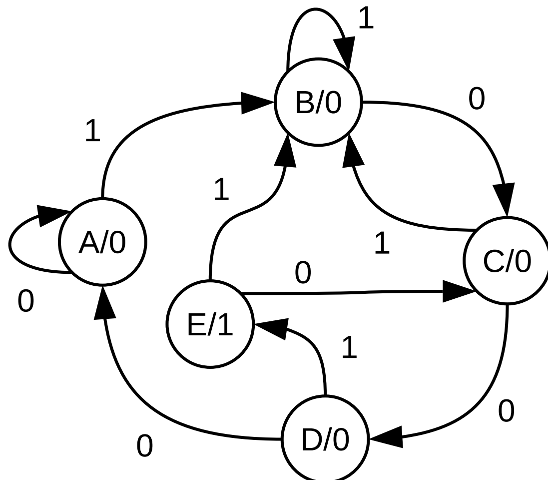
x may not meet the timing constrains (t_s, t_h)

Z_{mealy} activates during less than a clock cycle.

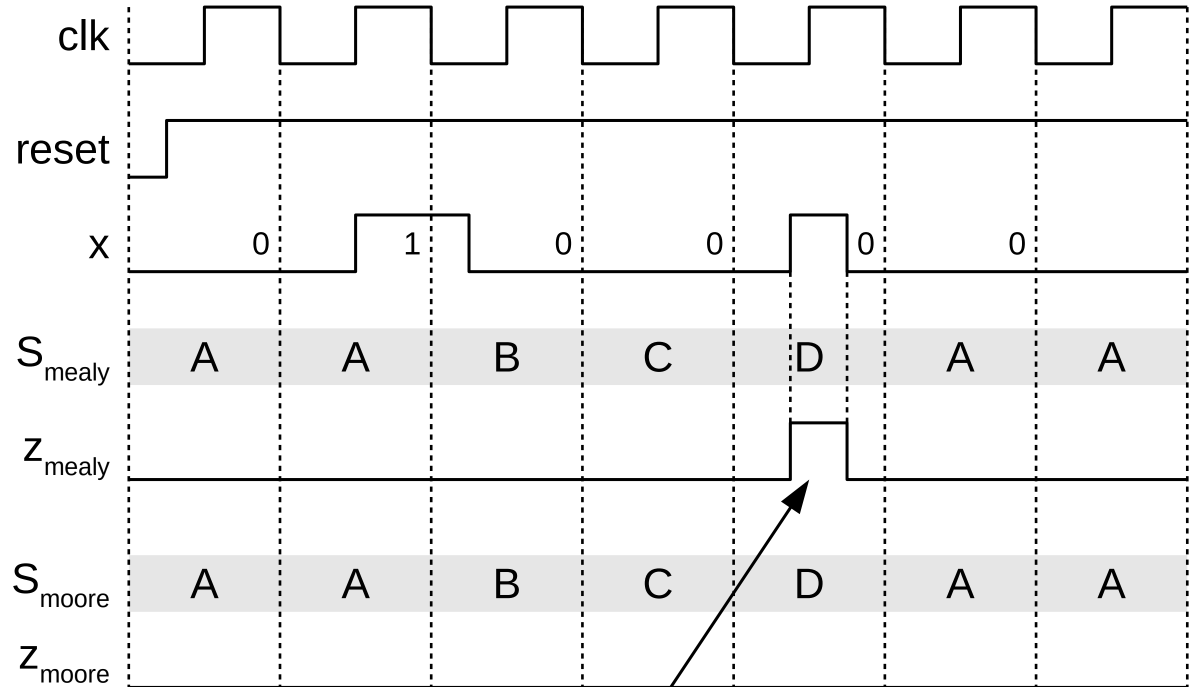
Mealy vs Moore
Non-synchronized inputs



Mealy



Moore

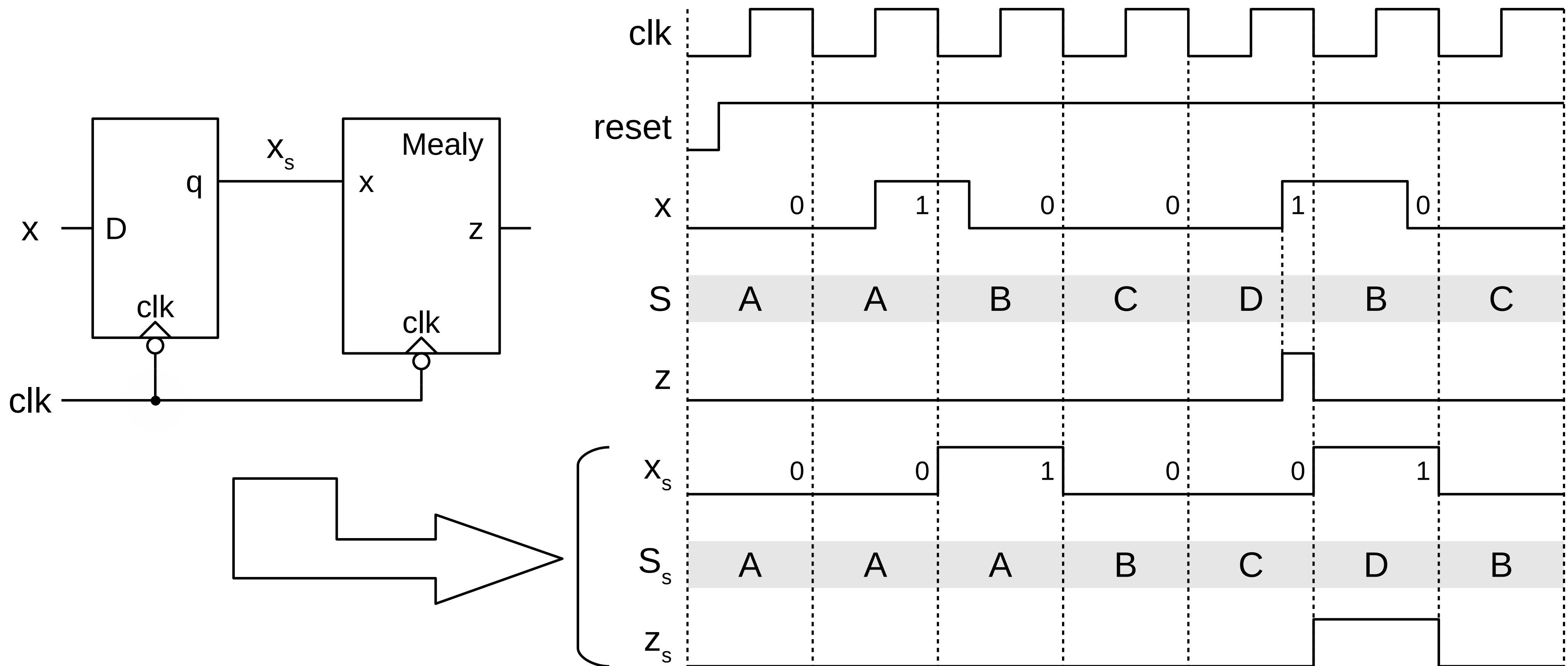


Z_{mealy} activates but it is not the correct sequence.

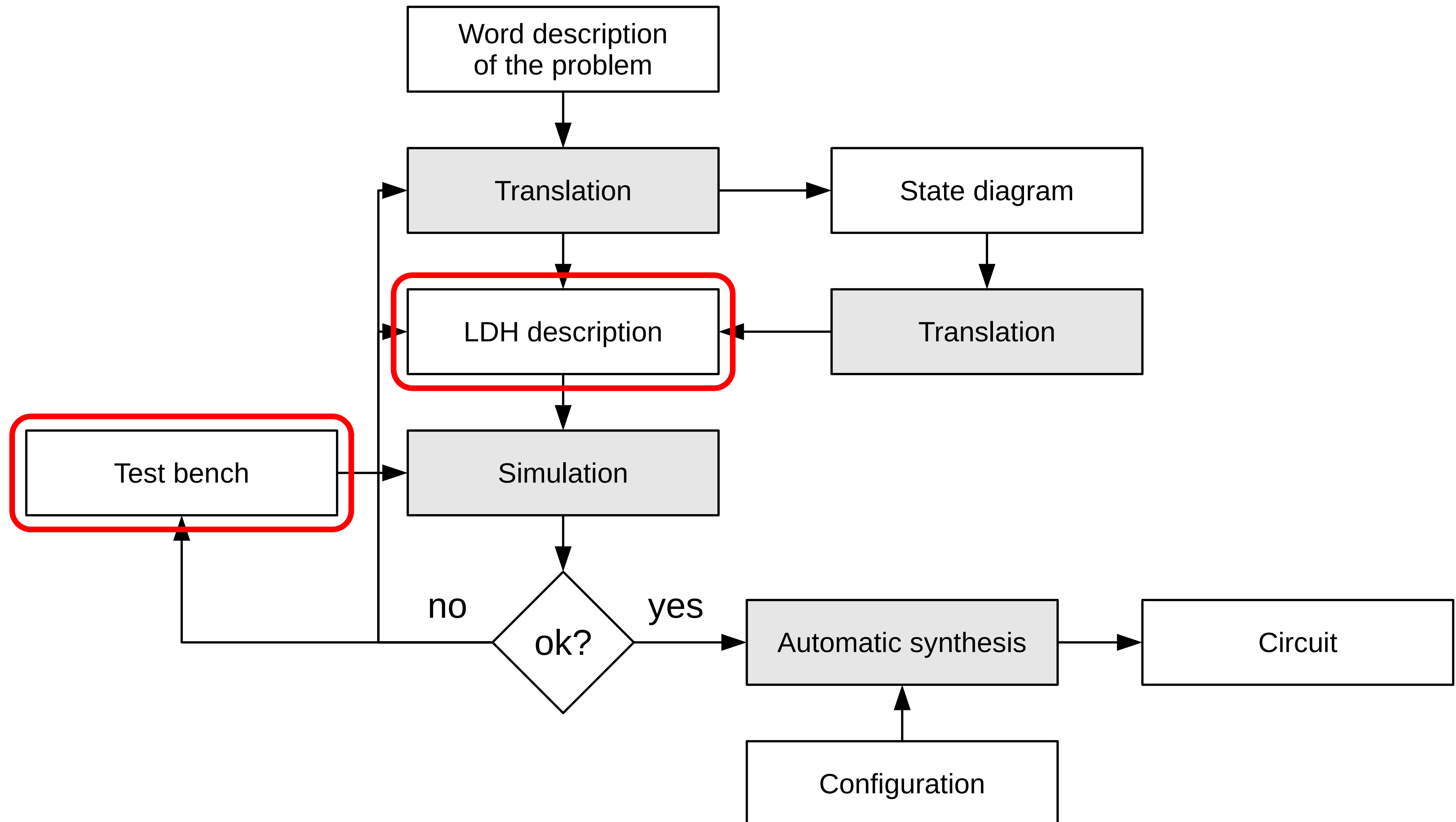
Input signal synchronization

- Asynchronous external input signals may change at any time (not synchronized with the clock).
- Asynchronous inputs may lead to strange behavior in Mealy machines.
  - Better use Mealy machines only with synchronized inputs.
- Asynchronous input may violate the timing constraints of the latches (setup and hold times) and may cause:
  - Excessive state change delay.
  - Changes to the wrong state.
  - Changes to blocking states.
- Non-synchronized inputs can be synchronized using D flip-flops.
  - Always synchronize external asynchronous signals if you want a robust circuit.
  - Drawback: the input is read with a delay of one cycle.

Input signal synchronization



Automatica SCC design procedure using Verilog

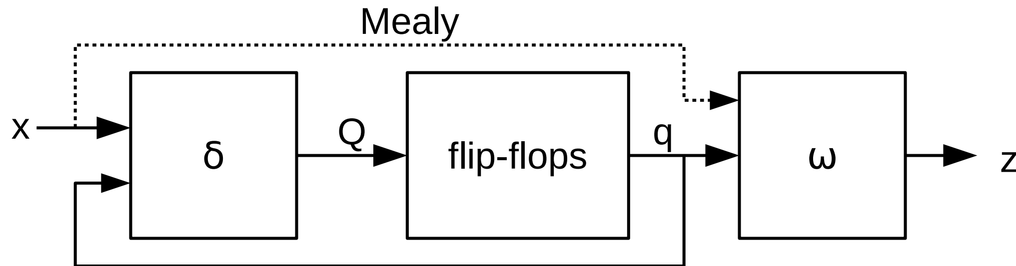


Verilog FSM descriptions

- States encoding
  - Parameters are used for state encoding.
  - Synthesis tools may change the encoding to optimize the implementation.
- State variable
  - Stores the current state (q)
- Next state variable
  - Stores the calculated next state (Q).

```
// States encoding

parameter [1:0]
    A = 2'b00,
    B = 2'b01,
    C = 2'b11,
    D = 2'b10;

// State (q): two flip-flops

reg [1:0] state;

// Next state (Q): two bits

reg [1:0] next_state;
```

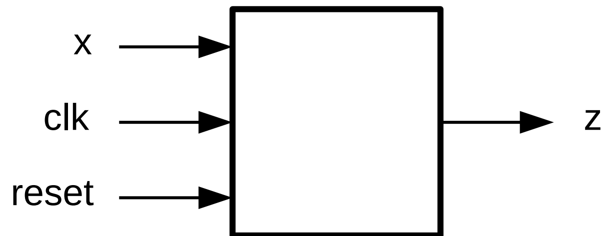
Verilog FSM descriptions



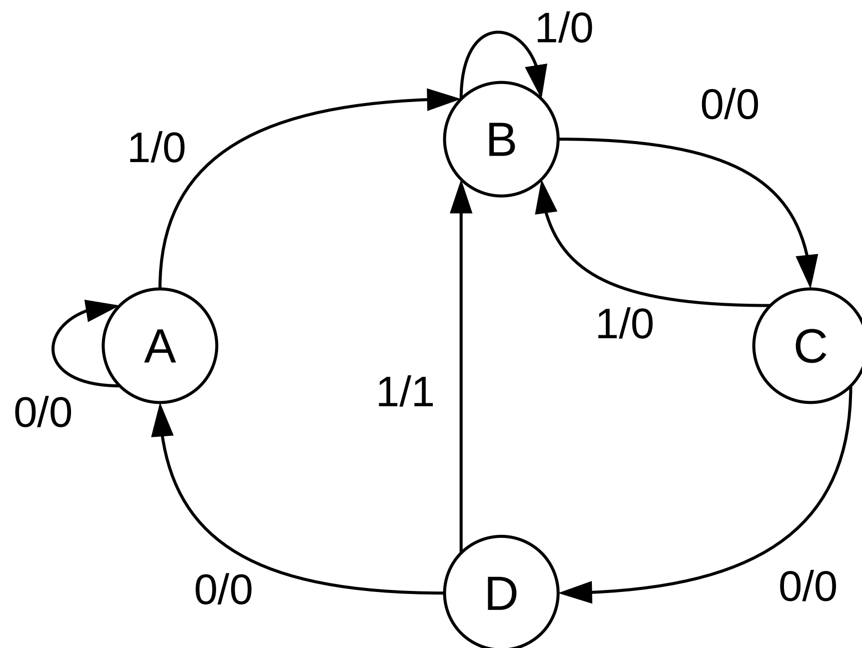
- Two/three processes
  - Process 1 (State change): represents the flip-flop block.
  - Process 2 (Next state calculation): excitation equations (δ).
  - Process 3 (Output calculation): output equations (ω).
    - Can be with process 2.
- Only the state change process is sequential (includes memory elements).

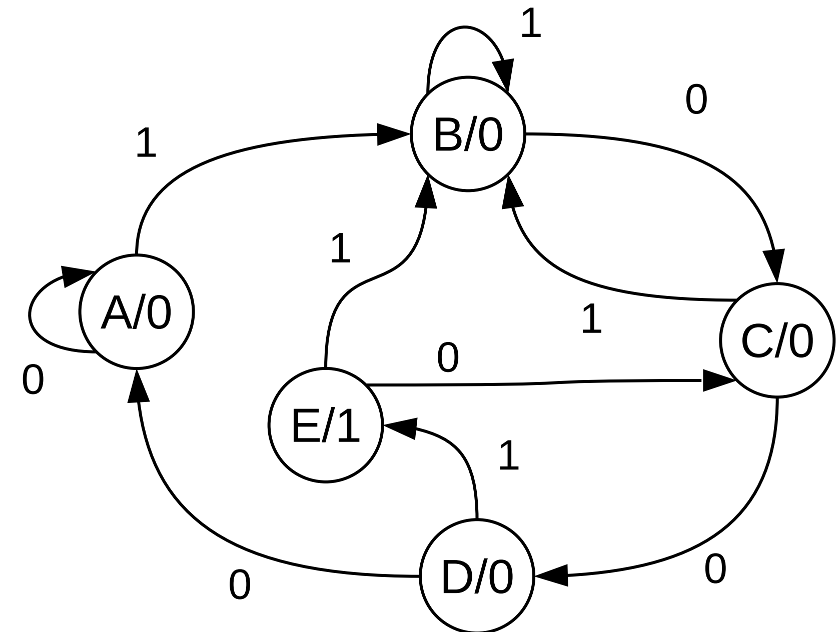
```
// State change process (sequential)
always @(posedge ck, posedge reset)
    if (reset)
        state <= A;
    else
        state <= next_state;

// Next state calculation process
// (combinational)
always @* begin
    case (state)
    A:
        next_state = . . .;
    B:
        next_state = . . .;
    . . .
    endcase
end

// Output calculation process
// (combinational)
always @* begin
    z = . . .;
end
```

Verilog FSM descriptions. Example



See examples in verilog-course.v





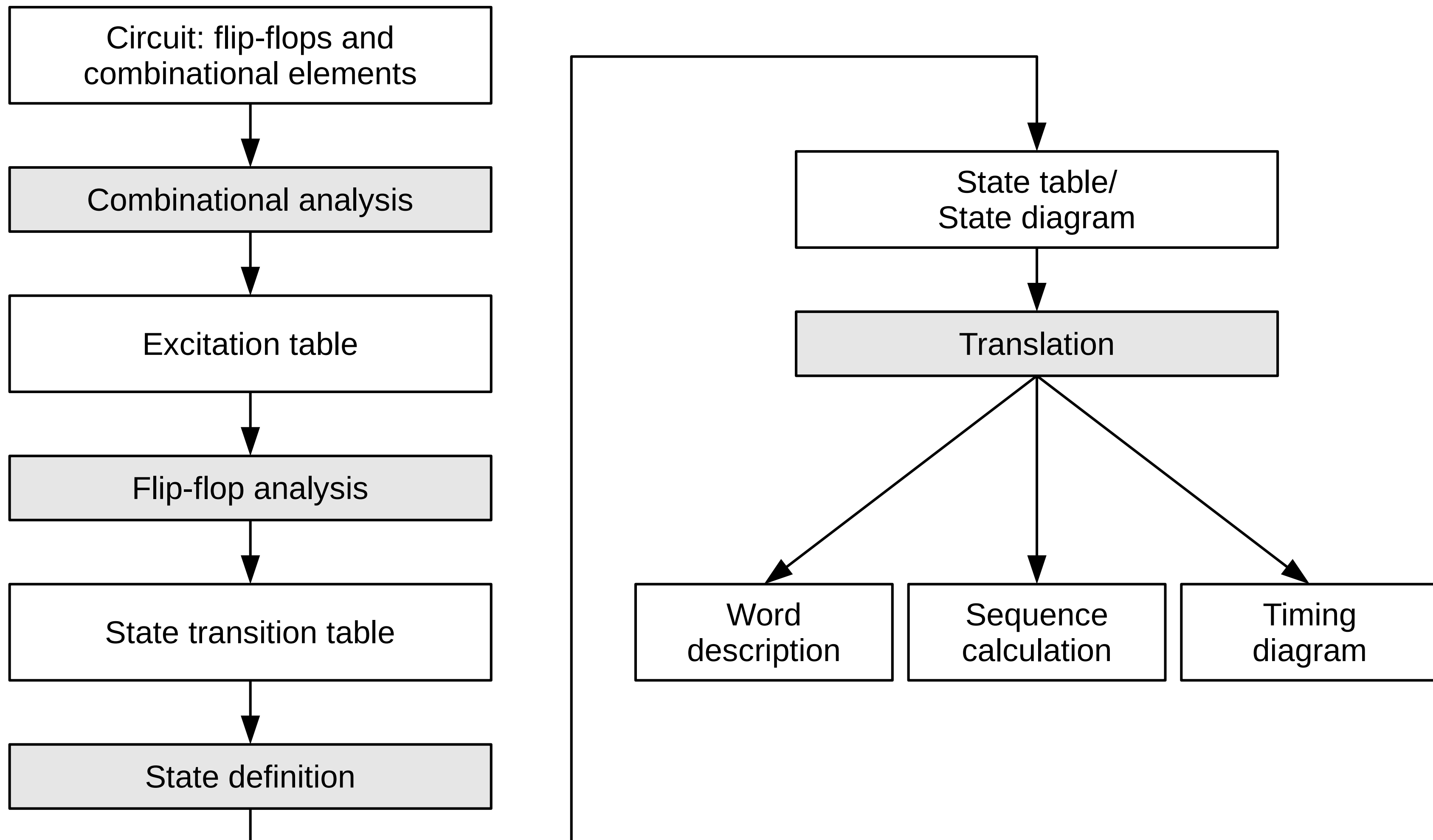
Contents

- Introduction
- Latches and flip-flops
- Synchronous Sequential Circuits (SSC) design
- SSC analysis
  - SSC (FSM) functional analysis
  - SSC timing analysis
    - Manual timing analysis
    - Automatic timing analysis using Verilog
- SCC application examples

SSC (FSM) functional analysis

- Opposite process to the synthesis
- Objective:
  - Starting from an implemented circuit (circuit diagram), describe its operation and usefulness.
- Procedure
  - Description of the synchronous behavior by obtaining the states table or states diagram. It just takes the same steps than the synthesis in reverse order.
  - Description of the asynchronous behavior: reset, etc.
  - Description of the circuit's operation. It is not systematic and depends on additional information:
    - Where does the circuit come from?
    - Do we know its purpose? Alarm system, control, etc.

SSC (FSM) functional analysis



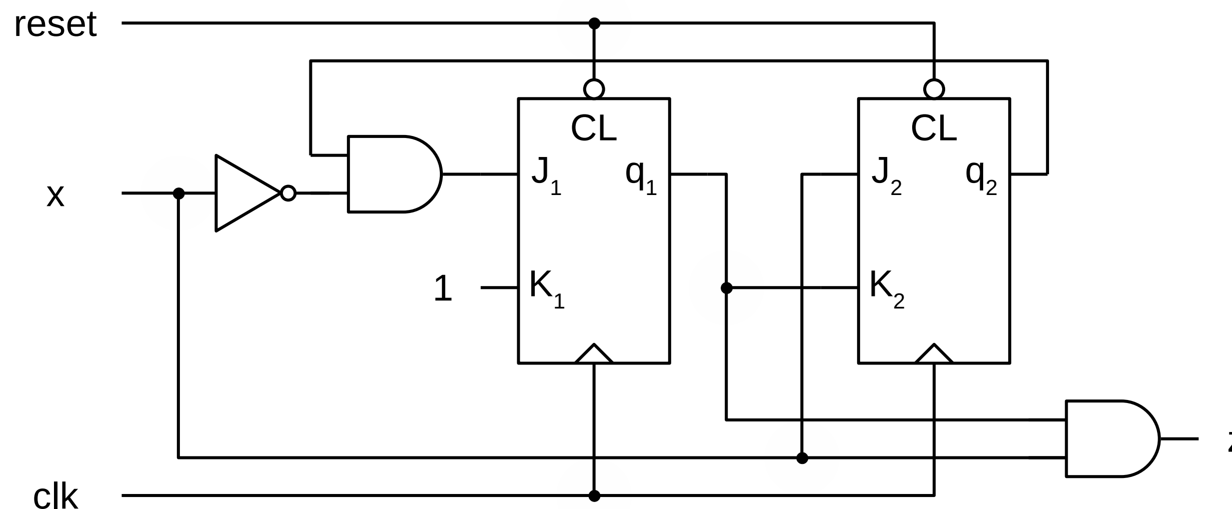
SSC (FSM) functional analysis

Example 8

It is known that the circuit in the figure activates output 'z' when a 3-bit sequence is detected at input 'x'. Analyze the circuit and determine:

1) What is the sequence that activates the output.
2) The active value of the output.
3) If consecutive activation sequences may overlap.
4) The sequence of states and output values for the following input sequence:

   x: 0001100101011100100



SSC timing analysis

- Objective
  - Given a SSC and a set of input waveforms, obtain the waveforms at the circuit's outputs.
- Things to consider
  - It is possible to analyze circuits with flip-flops that are not SSC.
  - If it is a SSC, the timing analysis should correspond to the expected result from the FSM that the circuit implements.
- Manual procedure is very similar to the timing analysis of CC.
  - Combinational part: identical
  - Flip-flops (FF)
    - Synchronous behavior: update the state at every active clock edge according to the states table of the flip-flop.
    - Asynchronous behavior: apply the new state immediately according to asynchronous signal activation.
  - Delay:
    - From the clock edge to the output: t_{ck-q}
    - From asynchronous inputs to the output. Eg.: t_{CL-q}
- Automatic procedure using Verilog
  - Structural description of the circuit (possible including delays).
  - Write a test bench representing the input waveforms.
  - Simulate with a logic simulator.

Manual timing analysis. Example

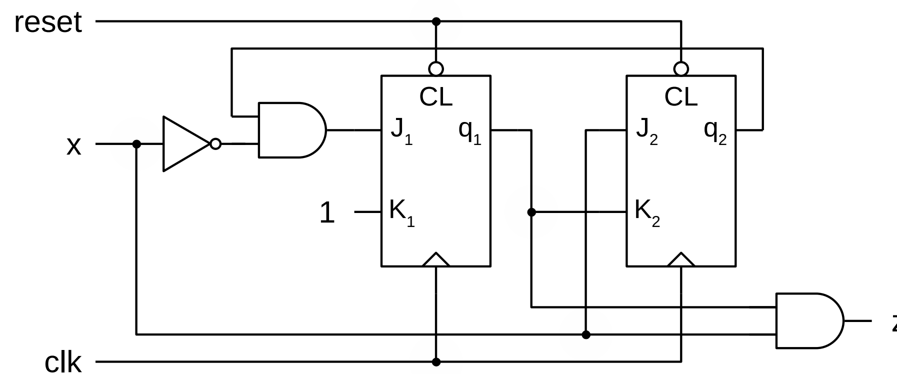
Example 9

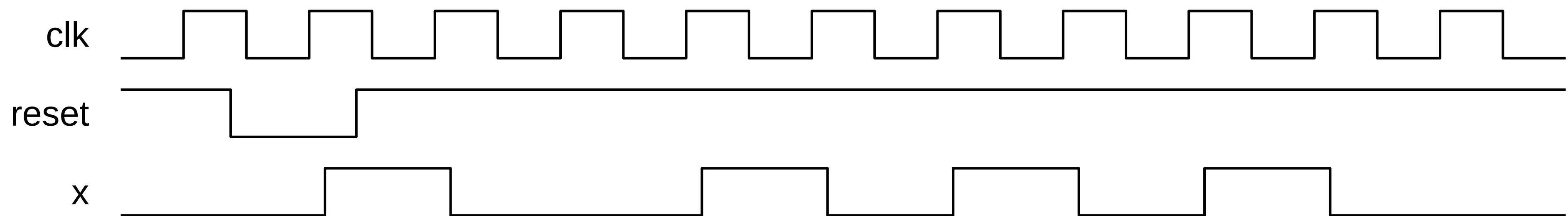
Obtain the output waveform for the input waveforms below and verify that the output activates as expected from the previous functional analysis. Do two analysis:

a) Without considering delays.

b) Considering all delays are the same ($t_p = t_{ck\text{-}q} = t_{CL\text{-}q} = \Delta$).

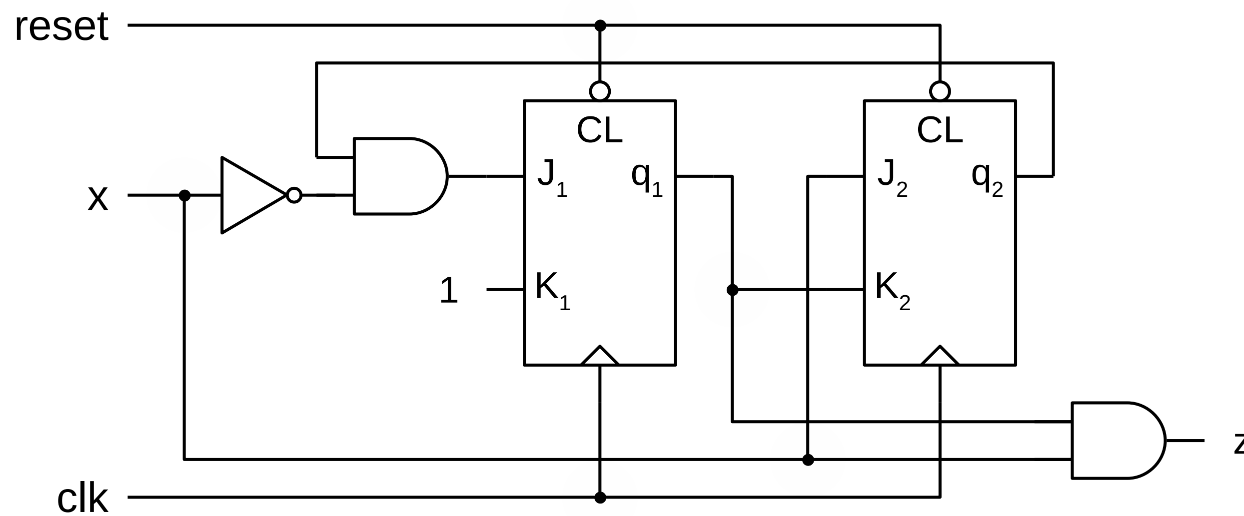
Note that 'x' is synchronized to the clock's active edge.





Manual timing analysis. Example without delay

Obtain the equations of the flip-flops inputs.



$J_1 = \overline{x}\, q_2$

$K_1 = 1$

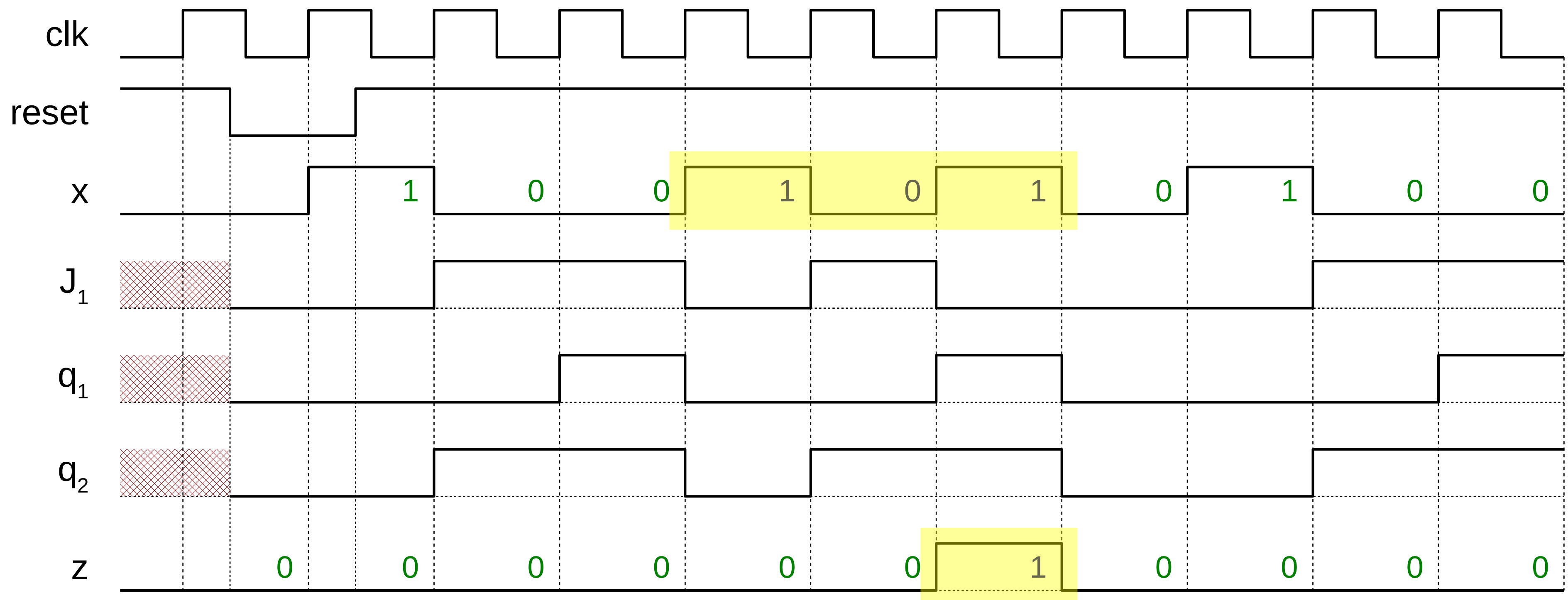
$J_2 = x$

$K_2 = q_1$

$z = x\, q_1$

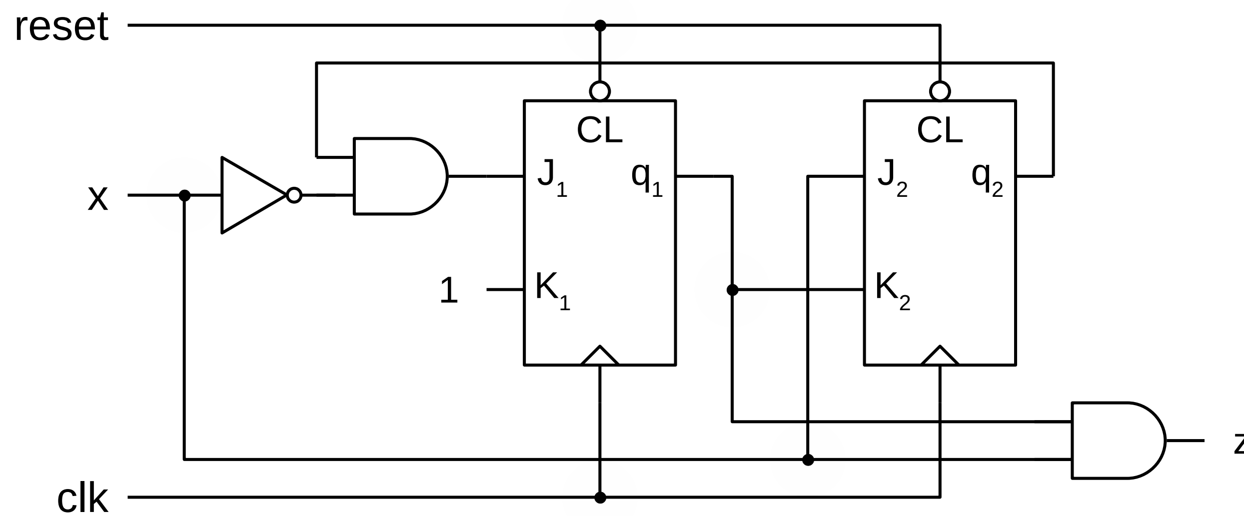
Manual timing analysis. Example without delay

$J_1 = \overline{x}\, q_2$ $K_1 = 1$ $J_2 = x$ $K_2 = q_1$ $z = x\, q_1$



Manual timing analysis. Example with delay

Obtain the equations of the signals at all the internal nodes.
Do not substitute signals. We need them to calculate delays.



$x_n = \overline{x}$

$J_1 = x_n\, q_2$

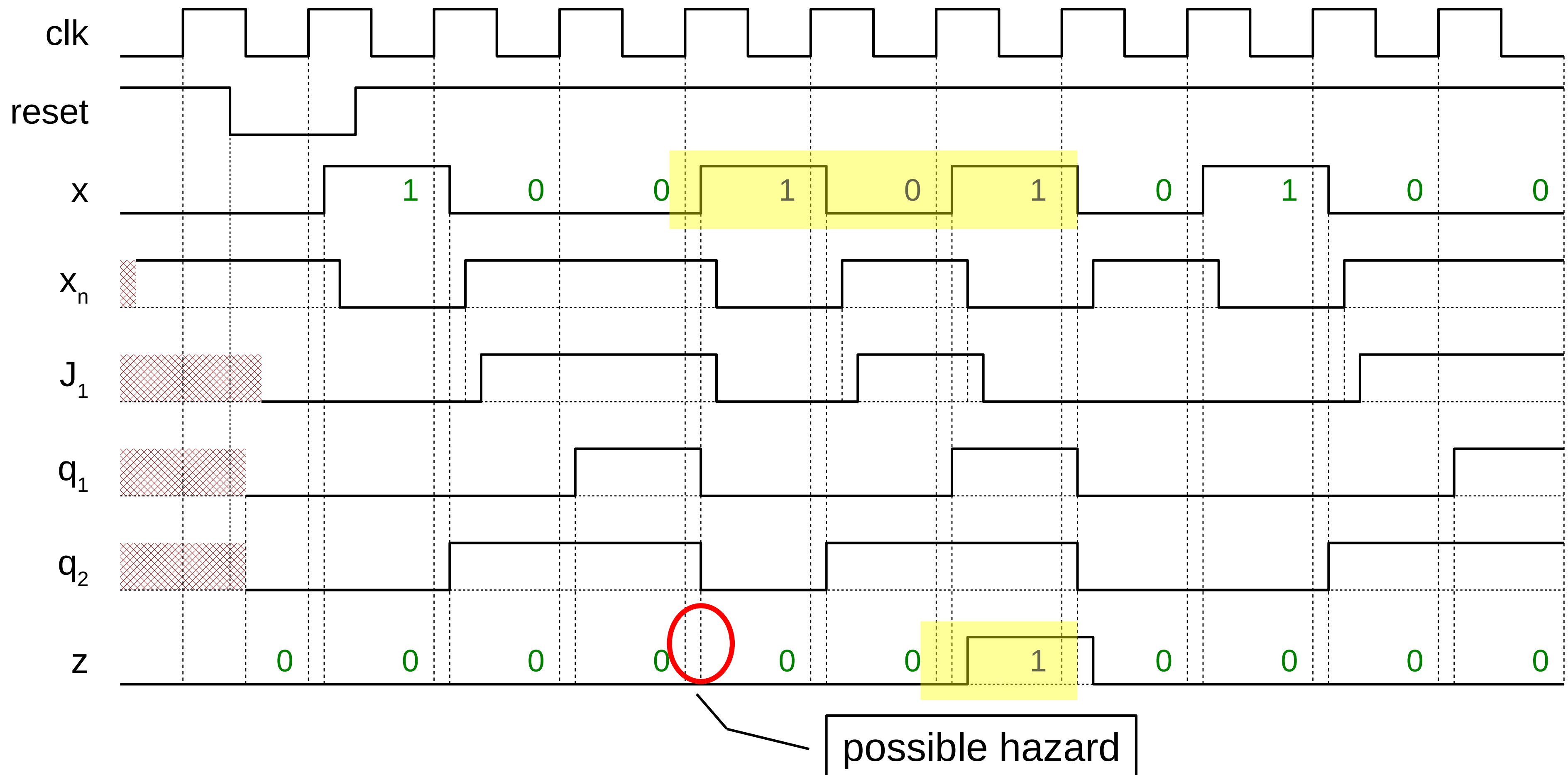
$K_1 = 1$

$J_2 = x$

$K_2 = q_1$

$z = x\, q_1$

Manual timing analysis. Example with delay

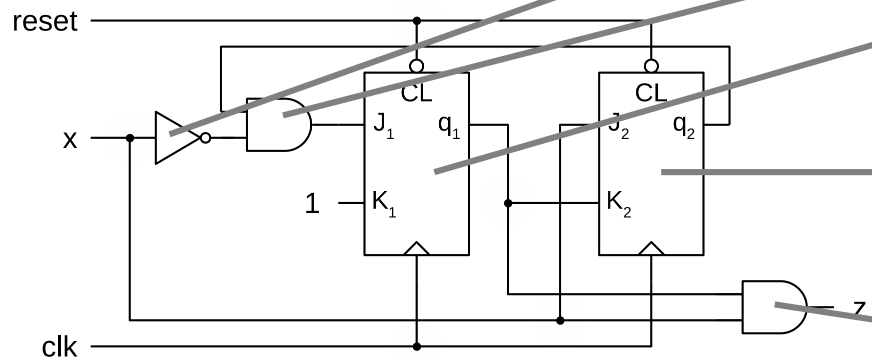


Manual timing analysis. Example with delay

- Notes:
  - Input 'x' is synchronized to the clock.
  - The initial value of the signals is unknown.
- Procedure
  - Draw $x_n = \overline{x}$ from x.
  - Apply the 'reset' and draw q_1 and q_2 as long as you can (up to the third active edge of the clock).
  - Do repeatedly:
    - Draw J_1 up to where we know the value of q_2 ($J_1 = x_n q_2$).
    - Draw q_1 up to where we know the value of J_1.
    - Draw q_2 up to where we know the value of q_1 ($K_2 = q_1$).
  - Draw 'z' from 'x' and q_1.

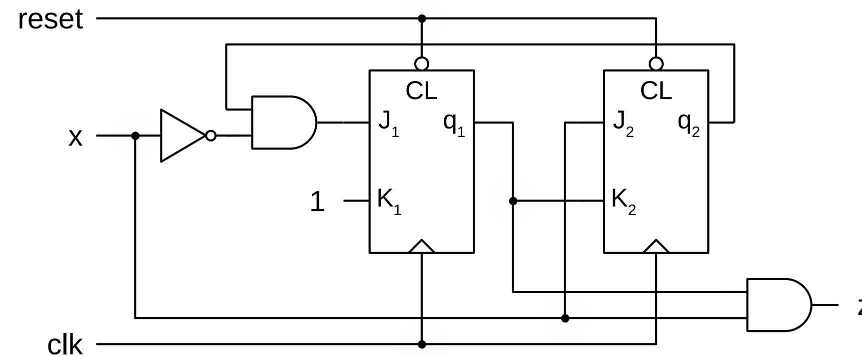
Automatic timing analysis using Verilog

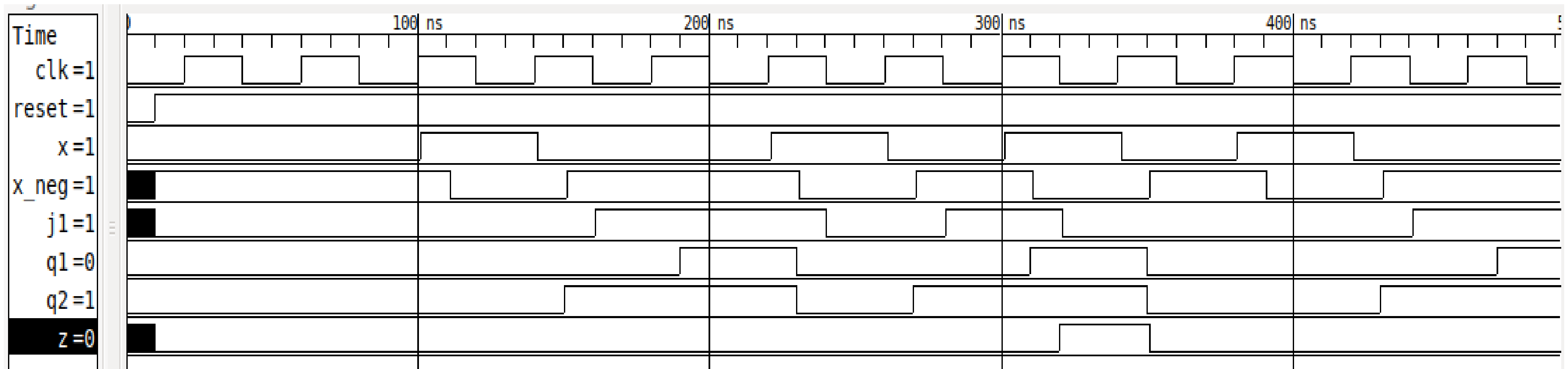
- Convert the circuit to analyze to a Verilog's structural description including delays.
- Write a test bench with the input waveforms you want.
- Let the simulator do its job :)




```
module sequence #(
    parameter delay = 10
    )(
    input wire clk,
    input wire reset,
    input wire x,
    output wire z
    );

    // internal wires
    wire x_neg, j1, q1, q2;

    not #delay not1 (x_neg, x);

    and #delay and1 (j1, x_neg, q2);

    jkff #(.delay(delay)) jkff1
        (.clk(clk), .cl(reset),
         .j(j1), .k(1'b1), .q(q1));

    jkff #(.delay(delay)) jkff2
        (.clk(clk), .cl(reset),
         .j(x), .k(q1), .q(q2));

    and #delay and2 (z, x, q1);
endmodule
```

Automatic timing analysis using Verilog





Contents

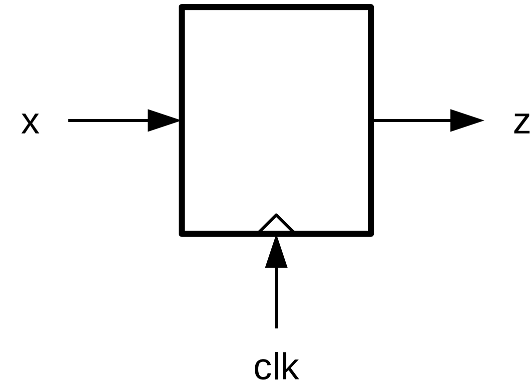
- Introduction
- Latches and flip-flops
- Synchronous Sequential Circuits (SSC) design
- SSC analysis
- SCC application examples
  - Sequence detectors
  - Control systems
  - Sequential arithmetic

Sequence detectors

- A FSM that detects one or more specific sequence of symbols in the output.
  - Symbols may be one or more bits. Often just one bit like in serial communication: data arrives one bit at a time.
- Upon sequence detection, some action is taken: activate something, etc.
- Many problems can be modeled as sequence detectors.
- Sequence detector types:
  - Using groups: symbols arrive in groups of fixed length. A correct sequence must fit one group.
  - Not using groups: a correct group may start with any input symbol.
    - Overlapping sequences: the ending symbols of a detected sequence my also be the first symbols of another detected sequence.
    - Non-overlapping sequences: detected sequence cannot share symbols.

Sequence detectors

- Detectors of sequence “1001”
  - x: input.
  - z1: overlapping (example 3).
  - z2: non-overlapping.
  - z3: using groups.




```
 x: 0 0 1 0 | 0 1 0 0 | 1 0 0 1 | 0 1 0 0 | 1 0 0 1 | ...
z1: 0 0 0 0 | 0 1 0 0 | 1 0 0 1 | 0 0 0 0 | 1 0 0 1 | ...
z2: 0 0 0 0 | 0 1 0 0 | 0 0 0 1 | 0 0 0 0 | 1 0 0 0 | ...
z3: 0 0 0 0 | 0 0 0 0 | 0 0 0 1 | 0 0 0 0 | 0 0 0 1 | ...
```

Sequence detectors

Example 10

Design the following sequence detectors (only states diagram) for the sequence "1001".

a) Non-overlapping as a Mealy's machine.

b) Non-overlapping as a Moore's machine.

c) Using groups as a Mealy's machine.

d) Using groups as a Moore's machine.

Optionally, implement the circuits using the type of flip-flop and combinational circuit that you prefer. Determine which is the reset state and use the asynchronous clear in the flip-flops to force it at initialization.

Sequence detectors

Example 11

When a signal is generated from a mechanical contact (like a button press), it typically “bounces” between '0' and '1' before it settles at its final value both when pressing and releasing the button. This is known as contact bounce.

Design a filter circuit with an input 'x' and an output 'z'. The input is supposed to be connected to a mechanical button. Each time the button is pressed (x=1), the output 'z' should be set to '1' during one clock cycle, but only if x=1 for at least 3 clock cycles in order to avoid possible contact bounces.

a) Design the filter as a Mealy's machine.

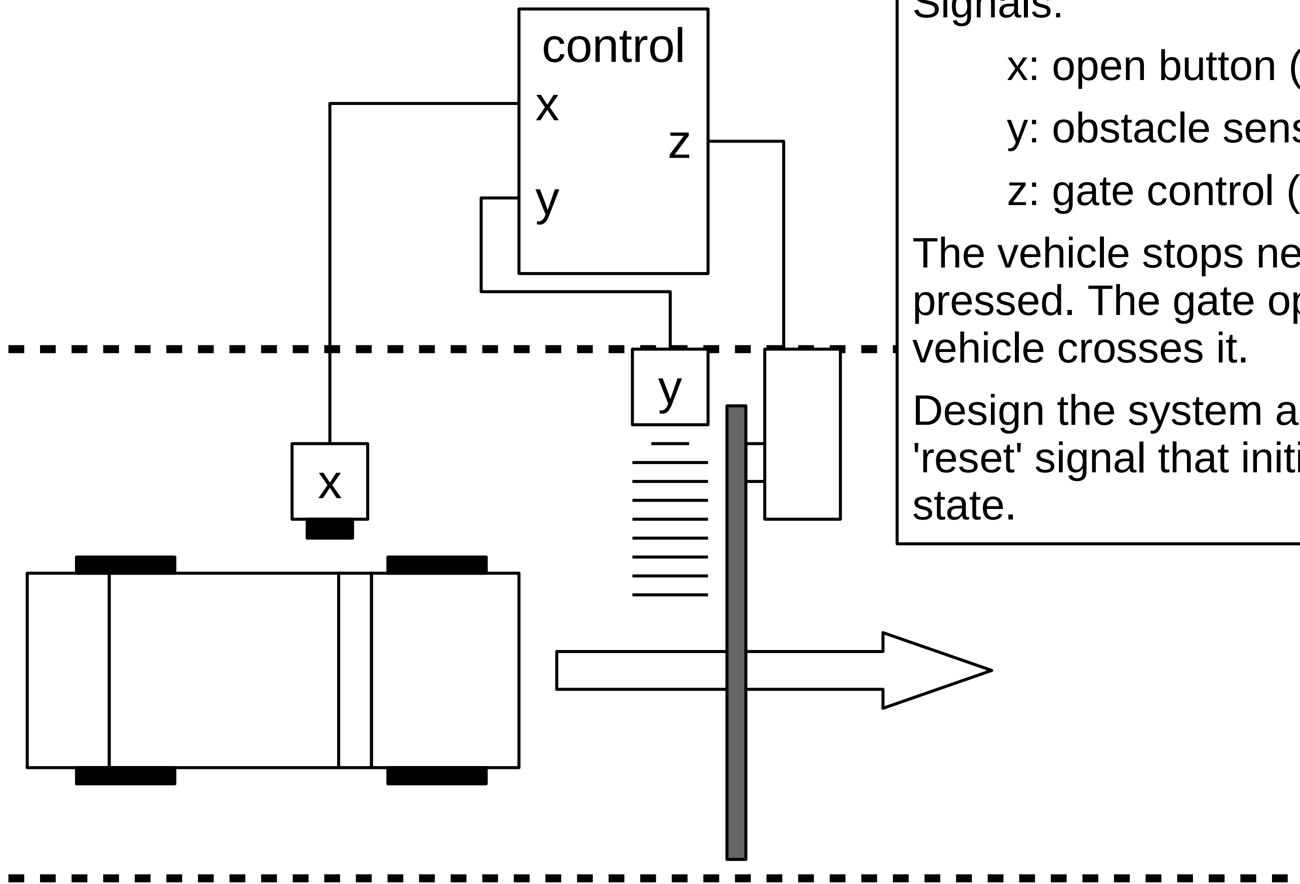
b) Design the filter as a Moore's machine.

Since the input is asynchronous and noisy, use a D flip-flop as an input synchronizer.

Sample sequence and expected output (output timing may change)

```
x: 0000010110111111111101011000000...
z: 0000000000000100000000000000000...
```

Gate control with obstacle sensor



Example 12

Design a gate control circuit as follows:

Signals:

- x: open button (0: not pressed, 1: pressed)
- y: obstacle sensor (0: obstacle, 1: no obstacle)
- z: gate control (0: close, 1: open)

The vehicle stops next to the button and the button is pressed. The gate opens and remains open until the vehicle crosses it.

Design the system as a Moore's machine. Add a 'reset' signal that initializes the system in a safe state.

Sequential adder

Example 13

A sequential adder circuit has two inputs 'a' and 'b' and an output 'z'. The digits of two number A and B arrive at input 'a' and 'b' respectively, bit by bit, and synchronized to the clock signal. The circuit must generate at output 'z', bit by bit, the result of the operation A+B, synchronized to the clock signal.

a) Design the circuit as a Mealy's machine using using JK flip-flops and logic gates.

b) Design the circuit as a Moore's machine using D flip-flops and multiplexers.

c) Design the circuit as a Moore's machine using Verilog.

In all cases, use the input sequence shown below to test the circuits.

Inputs:

- clk: positive edge active clock.
- reset: asynchronous initialization (active low).
- a, b: data inputs.

Output:

- z: result.

```
reset: 0 0 1 1 1 1 1 1 1 1 ...
    a: 0 0 0 1 0 1 0 0 0 0 ...
    b: 0 0 1 1 1 0 0 0 0 0 ...
```

Sequential multiplier

Example 14

A sequential multiplier circuit has one input 'x' and one output 'z'. The digits a number X arrive at input 'x', bit by bit, and synchronized to the clock signal. The circuit must generate at output 'z', bit by bit, the result of the operation 3*X, synchronized to the clock signal.

a) Design the circuit as a Mealy's machine using using JK flip-flops and logic gates.

b) Design the circuit as a Moore's machine using D flip-flops and multiplexers.

c) Design the circuit as a Moore's machine using Verilog.

In all cases, use the input sequence shown below to test the circuits.

Inputs:

- clk: positive edge active clock.
- reset: asynchronous initialization (active low).
- x: data input.

Output:

- z: result.

```
reset: 0 0 1 1 1 1 1 1 1 1 1 ...
    x: 0 0 0 1 0 1 1 0 0 0 0 ...
```