Digital Logic Design: a rigorous approach ©

Chapters 17-20: Flip-Flops, Synchronous Circuits, and Finite State Machines

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

December 21, 2021

Book Homepage:
`http://www.eng.tau.ac.il/~guy/Even-Medina`

Preliminary questions

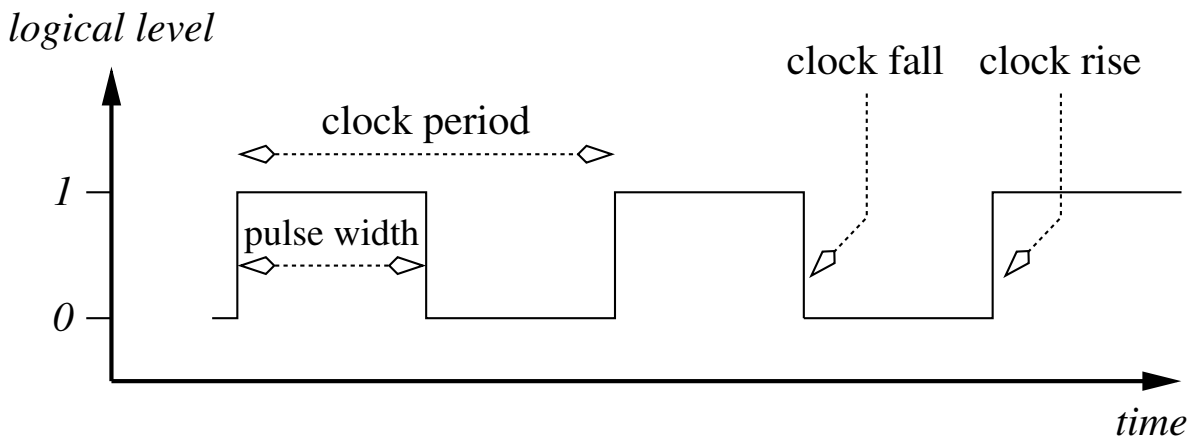
1. How is time measured in a synchronous circuit?
2. What is a "clock" in a microprocessor?
3. What is the frequency of a clock?
4. What is memory? How are bits stored?

The clock

the clock is generated by rectifying and amplifying a signal generated by special non-digital devices (e.g., crystal oscillators).

Definition

A clock is a periodic logical signal that oscillates instantaneously between logical one and logical zero. There are two instantaneous transitions in every clock period: (i) in the beginning of the clock period, the clock transitions instantaneously from zero to one; and (ii) at some time in the interior of the clock period, the clock transitions instantaneously from one to zero.



Clock cycles

- A clock partitions time into discrete intervals.
- t_i - the starting time of the ith clock cycle.
- $[t_i, t_{i+1})$ -clock cycle i.
- Clock period $= t_{i+1} - t_i$.

Assumption

We assume that the clock period equals 1.

$$t_{i+1} = t_i + 1 \ .$$

Flip-Flop

Definition

A flip-flop is defined as follows.

Inputs: Digital signals $D(t)$ and a clock CLK.

Output: A digital signal $Q(t)$.

Functionality:

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

t	$D[t]$	$Q[t]$
0	1	?
1	0	1
2	0	0
3	1	0
4	1	1

Clock enabled flip-flops

Definition

A clock enabled flip-flop is defined as follows.

Inputs: Digital signals $D(t), \text{CE}(t)$ and a clock CLK.

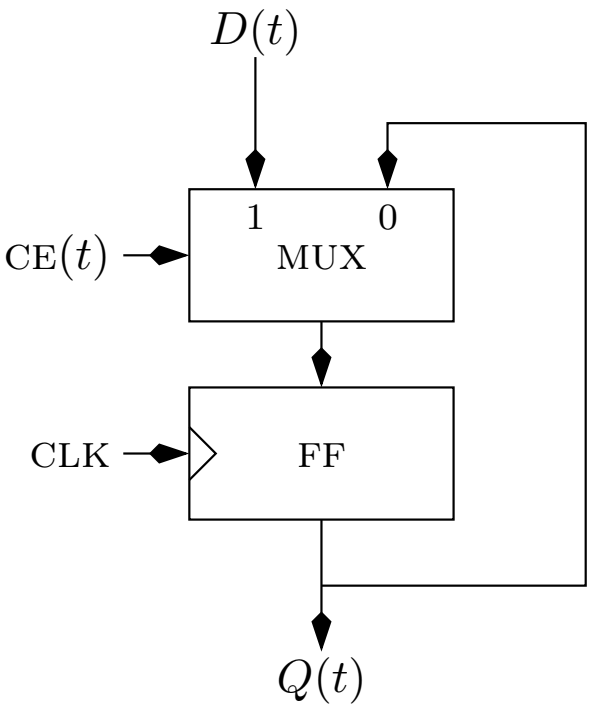
Output: A digital signal $Q(t)$.

Functionality:

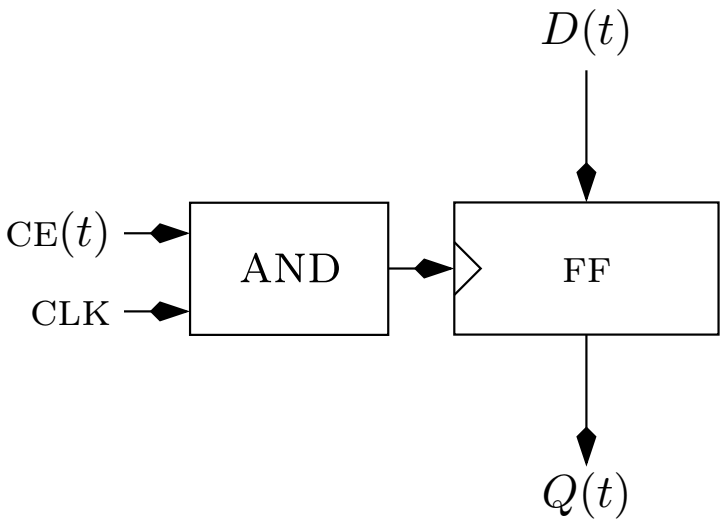
$$Q(t+1) = \begin{cases} D(t) & \text{if } \text{CE}(t) = 1 \\ Q(t) & \text{if } \text{CE}(t) = 0. \end{cases}$$

We refer to the input signal $\text{CE}(t)$ as the clock-enable signal. Note that the input $\text{CE}(t)$ indicates whether the flip-flop samples the input $D(t)$ or maintains its previous value.

Which design is a correct clock enabled FF?



(A)



(B)

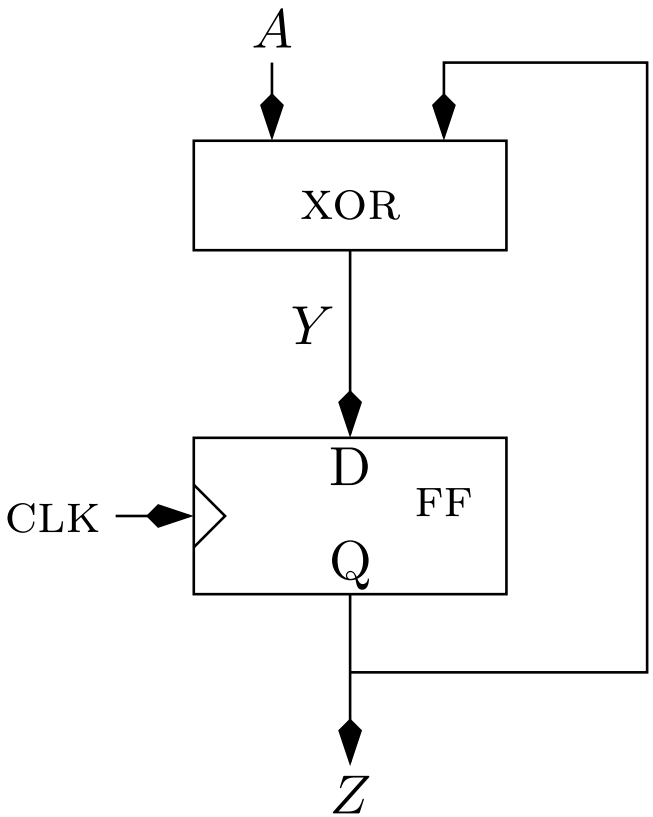
The Zero Delay Model

1. Transitions of all signals are instantaneous.
2. Combinational gates: $t_{pd} = t_{cont} = 0$.
3. Flip-flops satisfy:

$$Q(t+1) = D(t) \; .$$

4. Simplified model for specifying and simulating the functionality of circuits with flip-flops.
5. For a signal X, let X_i denote its value during the ith clock cycle.

Example: Sequential XOR



i	A_i	Y_i	Z_i
0	0	0	0
1	0	0	0
2	1	1	0
3	0	1	1
4	0	1	1
5	1	0	1
6	0	0	0
7	1	1	0
8	0	1	1

Sequential Adder

Definition

A *sequential adder* is defined as follows.

Inputs: A, B and a clock signal $\textsc{clk}$, where $A_i, B_i \in \{0, 1\}$.

Output: S, where $S_i \in \{0, 1\}$.

Functionality: Then, for every $i \geq 0$,

$$\langle A[i:0] \rangle + \langle B[i:0] \rangle = \langle S[i:0] \rangle \pmod{2^{i+1}}.$$

Sequential Adder Implementation

Sequential Adder: Correctness

Theorem

$$\sum_{j=0}^{i} A_j \cdot 2^j + \sum_{j=0}^{i} B_j \cdot 2^j = \sum_{j=0}^{i} S_j \cdot 2^j + c_{out}(i) \cdot 2^{i+1} .$$

Proof.

The proof is by induction on i.
The induction basis for $i = 0$ follows from the functionality of the full-adder:

$$A_0 + B_0 + C_{in}(0) = 2 \cdot C_{out}(0) + S_0 .$$

This requires that $C_{in}(0) = 0$! Namely, that the FF is initialized to zero. (We will discuss how to partly mitigate the issue of initialization later.) □

Sequential Adder: Implementation - correctness (cont.)

Proof.

We now prove the induction step for $i > 0$.

$$\sum_{j=0}^{i} A_j \cdot 2^j + \sum_{j=0}^{i} B_j \cdot 2^j = (A_i + B_i) \cdot 2^i + \sum_{j=0}^{i-1} A_j \cdot 2^j + \sum_{j=0}^{i-1} B_j \cdot 2^j$$

$$= (A_i + B_i) \cdot 2^i + \sum_{j=0}^{i-1} S_j \cdot 2^j + C_{out}(i-1) \cdot 2^i$$

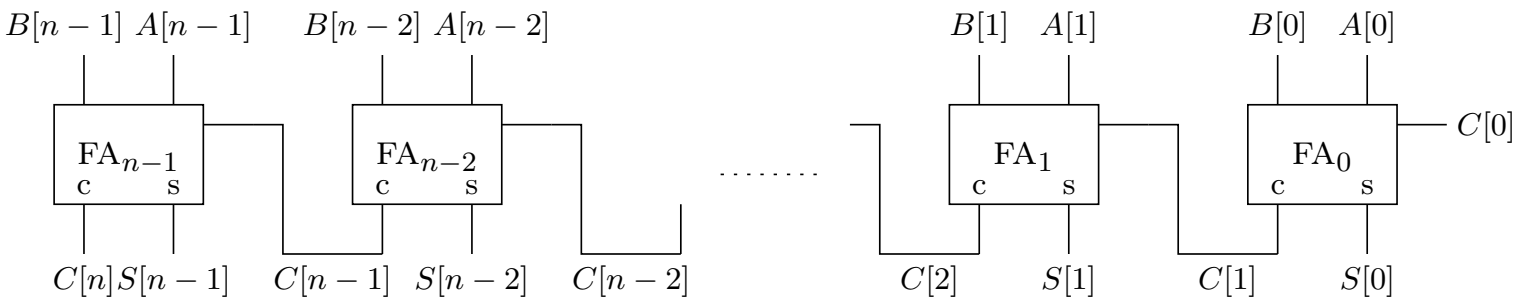
$$= (C_{in}(i) + A_i + B_i) \cdot 2^i + \sum_{j=0}^{i-1} S_j \cdot 2^j$$

$$= (S_i + 2 \cdot C_{out}(i)) \cdot 2^i + \sum_{j=0}^{i-1} S_j \cdot 2^j$$

$$= \sum_{j=0}^{i} S_j \cdot 2^j + C_{out}(i) \cdot 2^{i+1}.$$

Relation between $\mathrm{RCA}(n)$ and Sequential Adder

1. FA_i is "simulated" by the FA (in Seq. Adder) in the i'th clock cycle.
2. We can view $\mathrm{RCA}(n)$ as an "unrolling" of the Seq. Adder.



Comparison with Combinational Lower Bounds

1. Addition and XOR_n have functional cone of size n.
2. Every combinational circuit has cost $\Omega(n)$ and delay $\Omega(\log n)$.
3. But sequential versions have cost $O(1)$! How can that be?

Registers

A term register is used to define a memory device that stores a bit or more. There are two main types of register depending on how their contents are loaded.

1. Parallel Load Register
2. Shift Register (also called a serial load register)

Parallel Load Register - specification

Definition

An n-bit *parallel load register* is specified as follows.

Inputs:
- $D[n-1:0](t)$,
- $\mathrm{CE}(t)$, and
- a clock CLK.

Output: $Q[n-1:0](t)$.

Functionality:

$$Q[n-1:0](t+1) = \begin{cases} D[n-1:0](t) & \text{if } \mathrm{CE}(t) = 1 \\ Q[n-1:0](t) & \text{if } \mathrm{CE}(t) = 0. \end{cases}$$

An n-bit parallel load register is simply built from n clock enabled flip-flops.

Parallel Load Register - design

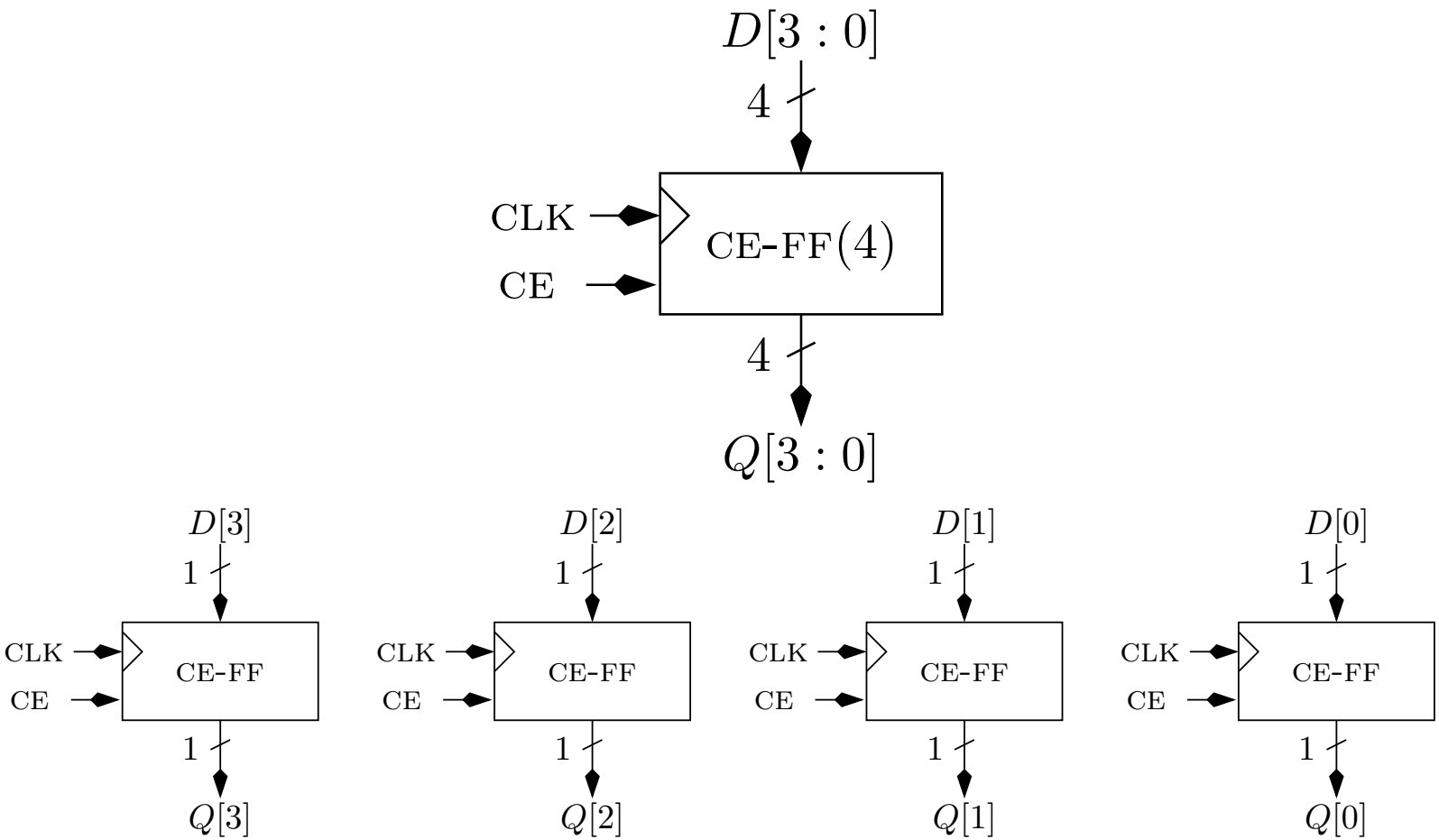
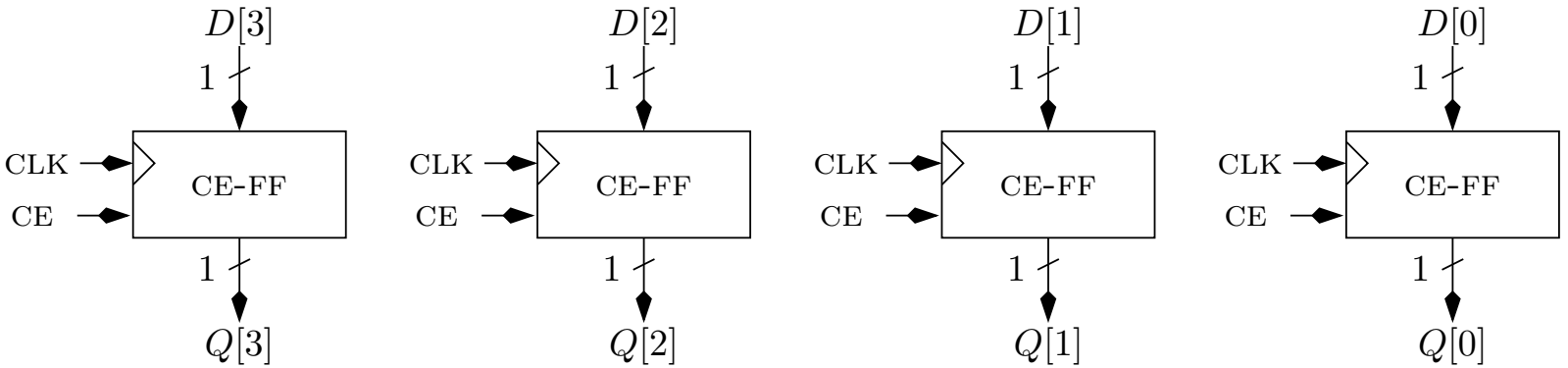


Figure: A 4-bit parallel load register

Parallel Load Register - simulation



i	$D[3:0]$	CE	$Q[3:0]$
0	1010	1	0000
1	0101	1	1010
2	1100	0	0101
3	1100	1	0101
4	0011	1	1100

Shift Register - definition

Definition

A *shift register* of n bits is defined as follows.

Inputs: $D[0](t)$ and a clock CLK.

Output: $Q[n-1](t)$.

Functionality: $Q[n-1](t+n) = D[0](t)$.

Shift Register - design

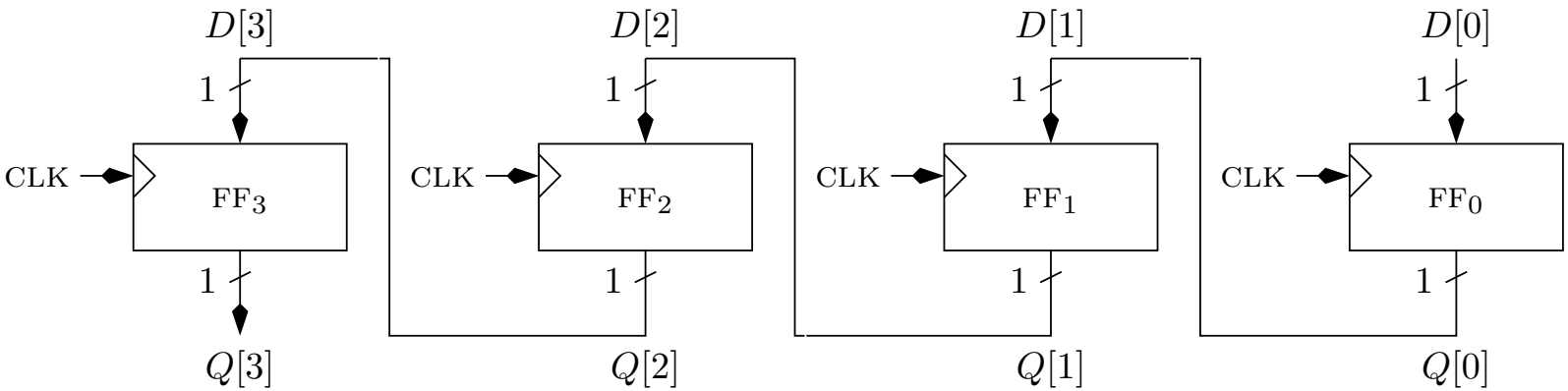
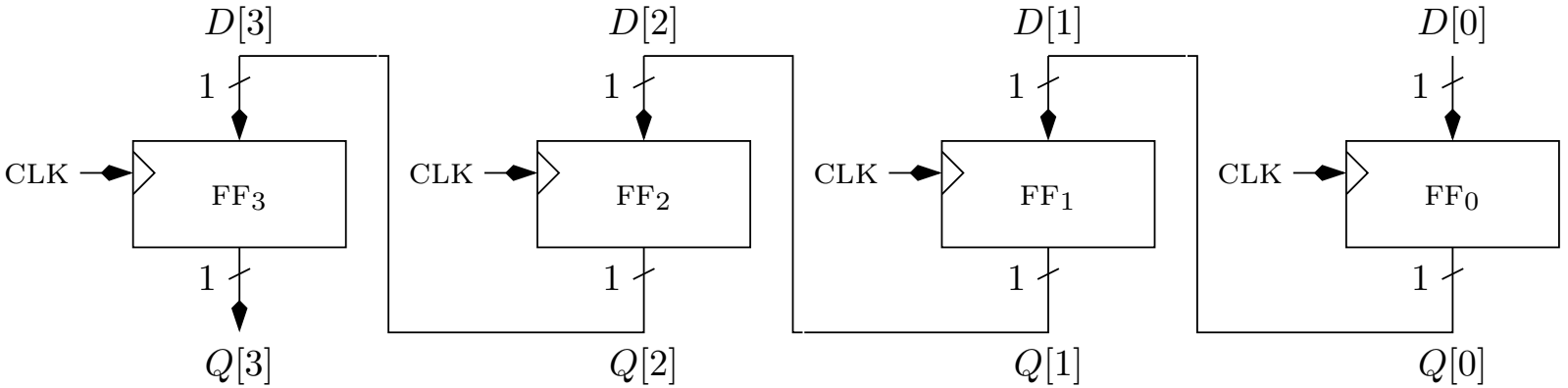


Figure: A 4-bit shift register. Functionality: $Q[3](t+4) = D[0](t)$

Shift Registers - simulation



i	$D[0]$	$Q[3:0]$
0	1	0000
1	1	0001
2	1	0011
3	0	0111
4	1	1110

ROM - definition/design

Definition

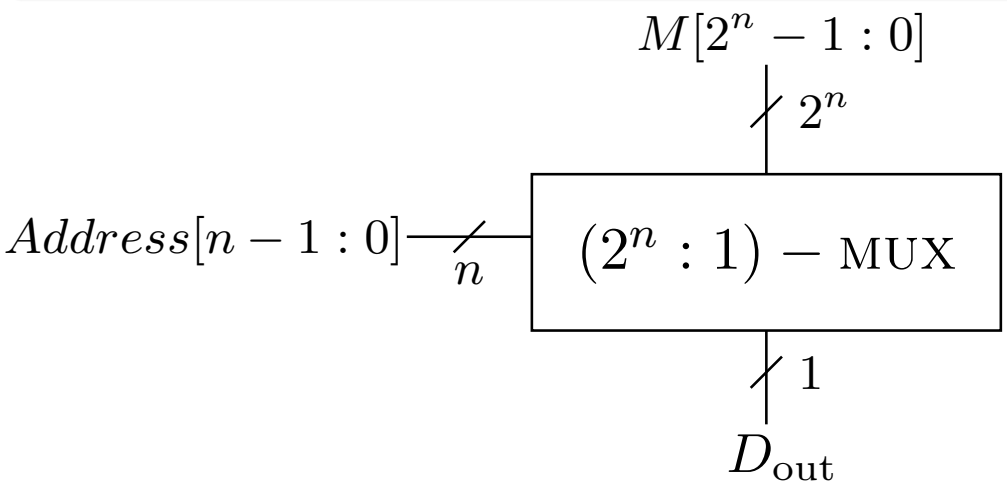
A $\text{ROM}(2^n)$ that implements a Boolean function $M : [0..2^n - 1] \to \{0, 1\}$ is defined as follows.

Inputs: $Address[n-1:0](t)$.

Output: $D_{\text{out}}(t)$.

Functionality :

$$D_{\text{out}} = M[\langle Address \rangle] .$$



Read-Only Memory (ROM)

- The contents stored in each memory cell are preset and fixed.
- ROMs are used to store information that should not be changed.
- For example, the ROM stores the program that is executed when the computer is turned on.
- Modern computers use non-volatile memory as ROM (such memory does allow write operations - and writing is often limited by "permissions")

Random Access Memory (RAM)

1. Hardware module that implements an array of memory cells, where each memory cell stores a single bit.
2. In each cycle, a single memory cell is accessed.
3. Two operations are supported: read and write.
   - read operation: the contents of the accessed memory is output.
   - write operation: a new value is stored in the accessed memory cell.
4. The number of memory cells is denoted by 2^n.
5. Each cell has a distinct address between 0 and $2^n - 1$.
6. The cell to be accessed is specified by an n-bit string called *Address*.
7. The array of memory cells is denoted by $M[2^n - 1 : 0]$. Let $M[i](t)$ denote the value stored in the ith entry of the array M during clock cycle t.

RAM - definition

Definition

A $\text{RAM}(2^n)$ is specified as follows.

Inputs: $Address[n-1:0](t) \in \{0,1\}^n, D_{\text{in}}(t) \in \{0,1\}$, $R/\overline{W}(t) \in \{0,1\}$ and a clock CLK.

Output: $D_{\text{out}}(t) \in \{0,1\}$.

Functionality :

1. data: array $M[2^n - 1 : 0]$ of bits.
2. initialize: $\forall i : M[i] \leftarrow 0$.
3. For $t = 0$ to ∞ do
   1. $D_{\text{out}}(t) = M[\langle Address \rangle](t)$.
   2. For all $i \neq \langle Address \rangle$: $M[i](t+1) \leftarrow M[i](t)$.
   3. $$M[\langle Address \rangle](t+1) \leftarrow \begin{cases} D_{\text{in}}(t) & \text{if } R/\overline{W}(t) = 0 \\ M[\langle Address \rangle](t) & \text{else.} \end{cases}$$

RAM - schematic

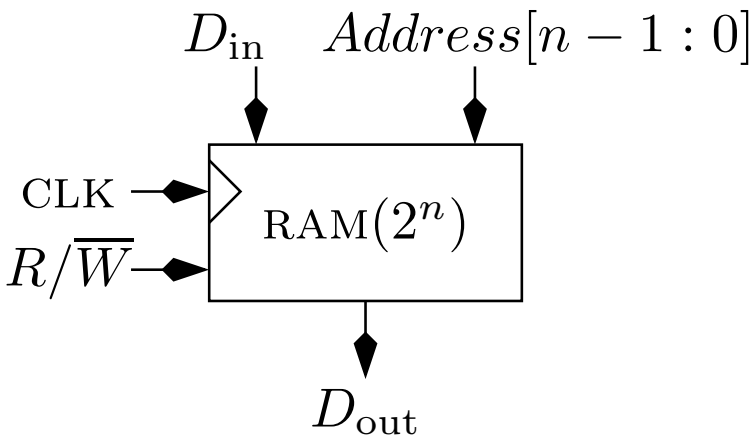
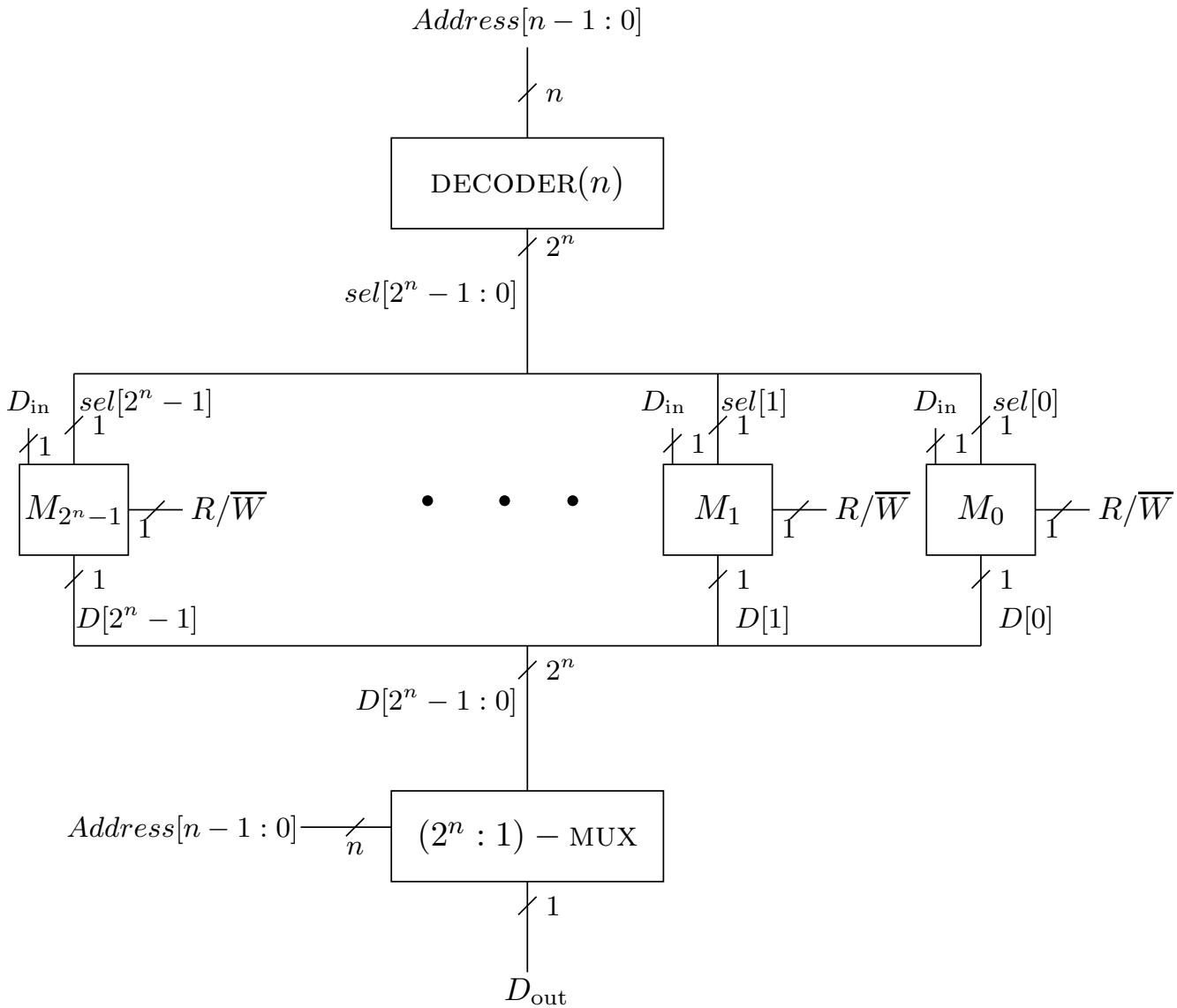


Figure: A schematic of a $\text{RAM}(2^n)$.

RAM -design


$Address[n-1:0]$
n
DECODER(n)
2^n
$sel[2^n-1:0]$
D_{in}
$sel[2^n-1]$
M_{2^n-1}
$R/\overline{W}$
$D[2^n-1]$
D_{in}
$sel[1]$
M_1
$R/\overline{W}$
$D[1]$
D_{in}
$sel[0]$
M_0
$R/\overline{W}$
$D[0]$
2^n
$D[2^n-1:0]$
$Address[n-1:0]$
n
$(2^n:1)-$MUX
1
D_{out}

Memory Cell - specification

Definition

A single bit *memory cell* is defined as follows.

Inputs: $D_{\text{in}}(t)$, $R/\overline{W}(t)$, $sel(t)$, and a clock CLK.

Output: $D_{\text{out}}(t)$.

Functionality:

Assume that D_{out} is initialized zero, i.e., $D_{\text{out}}(0) = 0$. The functionality is defined according to the following cases.

$$D_{\text{out}}(t+1) \leftarrow \begin{cases} D_{\text{in}}(t) & \text{if } sel(t) = 1 \text{ and } R/\overline{W}(t) = 0 \\ D_{\text{out}}(t) & \text{otherwise.} \end{cases}$$

Memory Cell - design

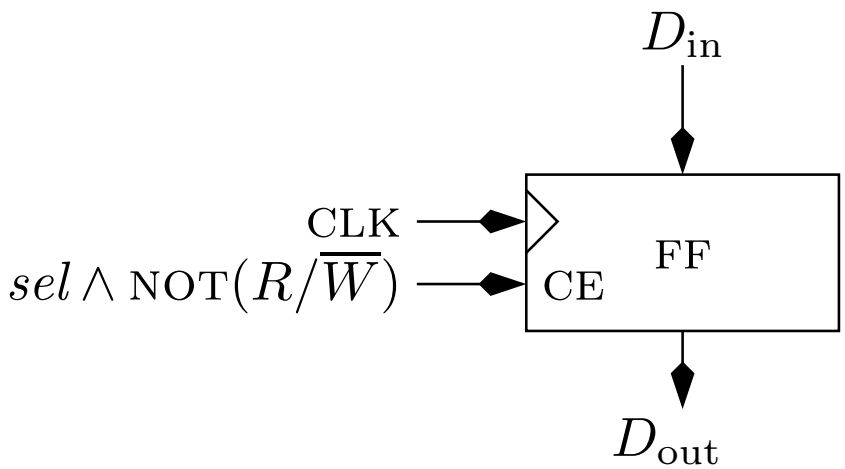


Figure: An implementation of a memory cell.

Summary of Part 1

- Clock signal & clock cycles.
- Flip-Flops and clock-enabled FF's
- Examples:
  1. Sequential XOR
  2. Sequential Adder
  3. Comparison with combinational lower bounds.
- Registers: parallel load and shift registers.
- ROM and RAM.

Preliminary questions

1. What is a synchronous circuit?
2. How can we initialize a synchronous circuit?

Synchronous Circuits

- Building blocks: combinational gates, wires, and flip-flops.
- The graph G of a synchronous circuit is directed but may contain cycles (e.g., sequential adder).
- A flip-flop has two inputs D and $\textsc{clk}$ that play quite different roles. We must make sure that we know the input port of each incoming edge.
- Definition based on a reduction to a combinational circuit...

Synchronous Circuits

Definition

A synchronous circuit is a circuit C composed of combinational gates, wires, and flip-flops that satisfies the following conditions:

1. There is an input gate that feeds the clock signal CLK.
2. The set of ports that are fed by the clock CLK equals the set of clock-inputs of the flip-flops.
3. Let C' denote a circuit obtained from C by stripping the flip-flops away. Then, the circuit C' is a combinational circuit.

Stripping Flip-Flops Away

Definition

1. Delete the input gate that feeds the clock CLK and all the wires carrying the clock signal.
2. Remove all the flip-flops.
3. Add an output gate for each D port.
4. Add an input gate for each Q port.

Example - stripping FFs away

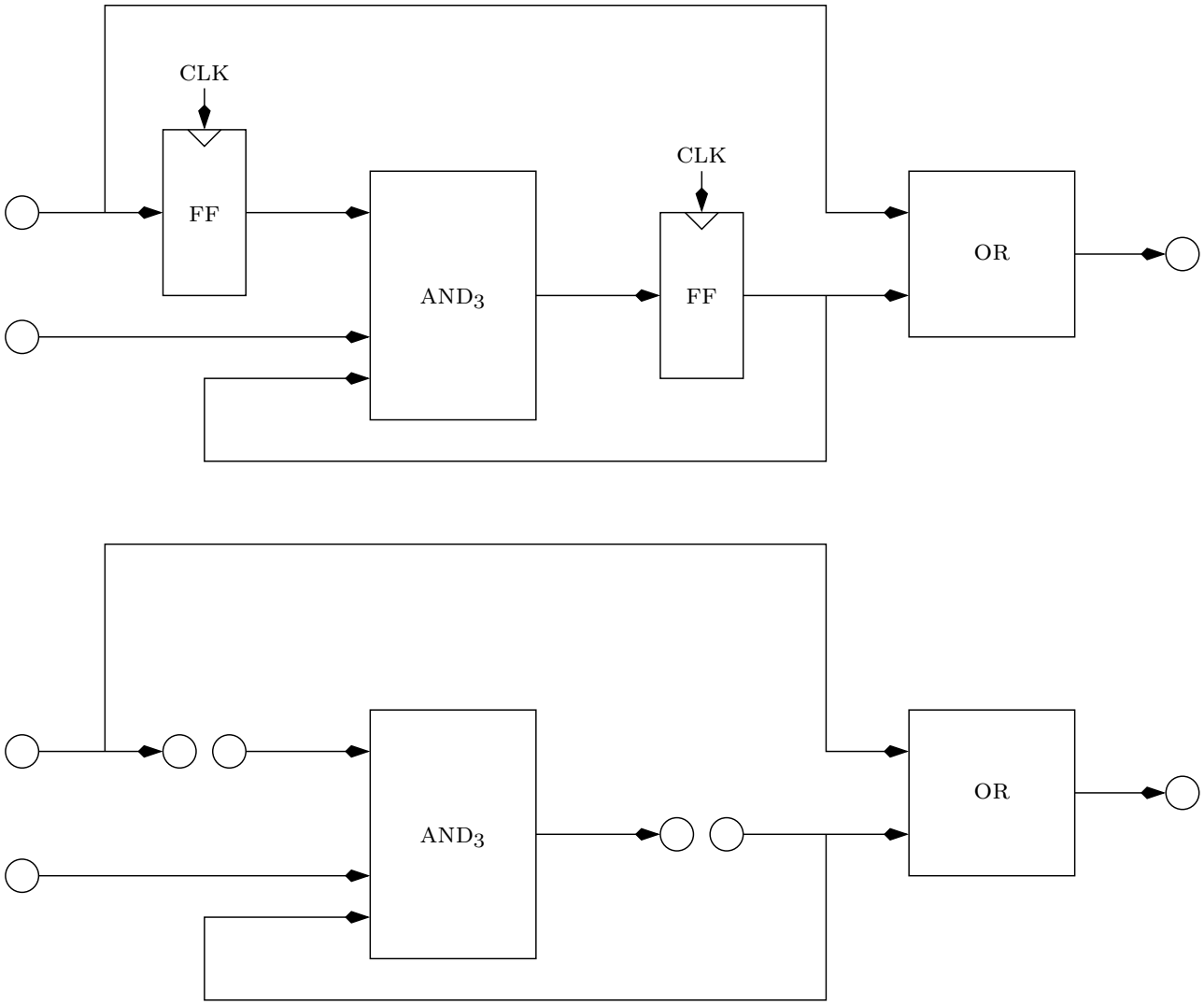


Figure: A synchronous circuit C and the combinational circuit C' obtained from C by stripping away the flip-flops.

Remarks:

It is easy to check if a given circuit C is a synchronous circuit.

- Check if there is a clock signal that is connected to all the clock terminals of the flip-flops and only to them.
- Strip the flip-flops away to obtain the circuit C'. Check if C' is a combinational circuit.

Cycles (closed paths) in a synchronous circuit

Claim

Every closed path in a synchronous circuit traverses at least one flip-flop.

Logical Simulation of Synchronous Circuits

Assumptions:

- Initialization (magical?): For every flip-flop FF_i, let $S_0(FF_i) \in \{0, 1\}$ denote the value output by FF_i in clock cycle $t = 0$.
- Input sequence: For every input gate X let $IN_t(X) \in \{0, 1\}$ the input fed by X in clock cycle t.

Initialization serves a crucial role in the induction basis!

Simulation Algorithm

Algorithm 1 SIM$(C, S_0, \{IN_t\}_{t=0}^{T-1})$ - An algorithm for simulating a synchronous circuit C with respect to an initialization S_0 and a sequence of inputs $\{IN_t\}_{t=0}^{T-1}$.

1. Construct the combinational circuit C' obtained from C by stripping away the flip-flops.
2. For $t = 0$ to $T - 1$ do:
   1. Simulate the combinational circuit C' with input values corresponding to S_t and IN_t. Namely, every input gate in C feeds a value according to IN_t, and every Q-port of a flip-flop feeds a value according to S_t. For every sink z in C', let z_t denote the value fed to z according to this simulation.
   2. For every Q-port S of a flip-flop, define $S_{t+1} \leftarrow NS_t$, where NS denotes the D-port of the flip-flop.

The Canonic Form of a Synchronous Circuit

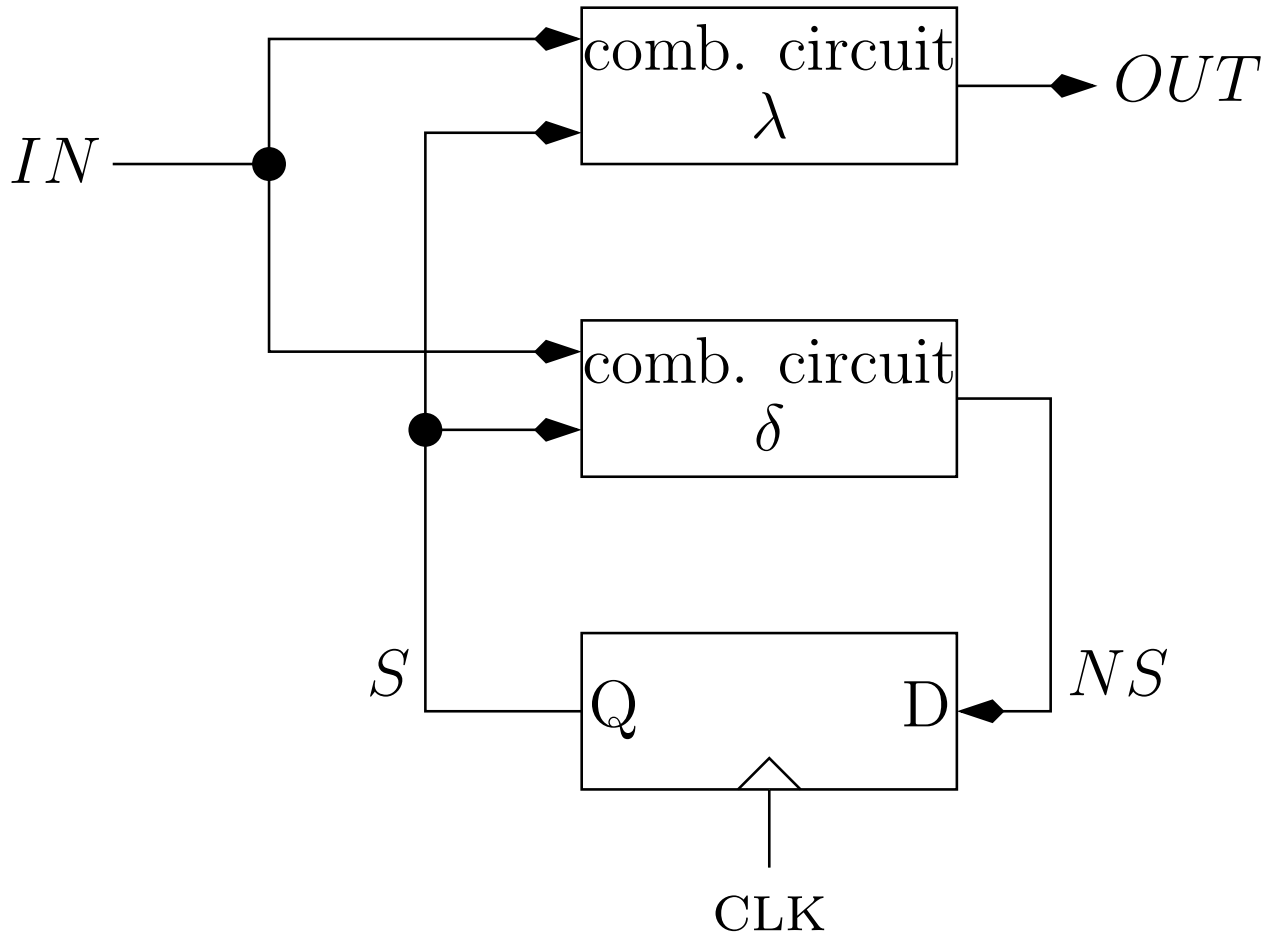


Figure: A synchronous circuit in canonic form.

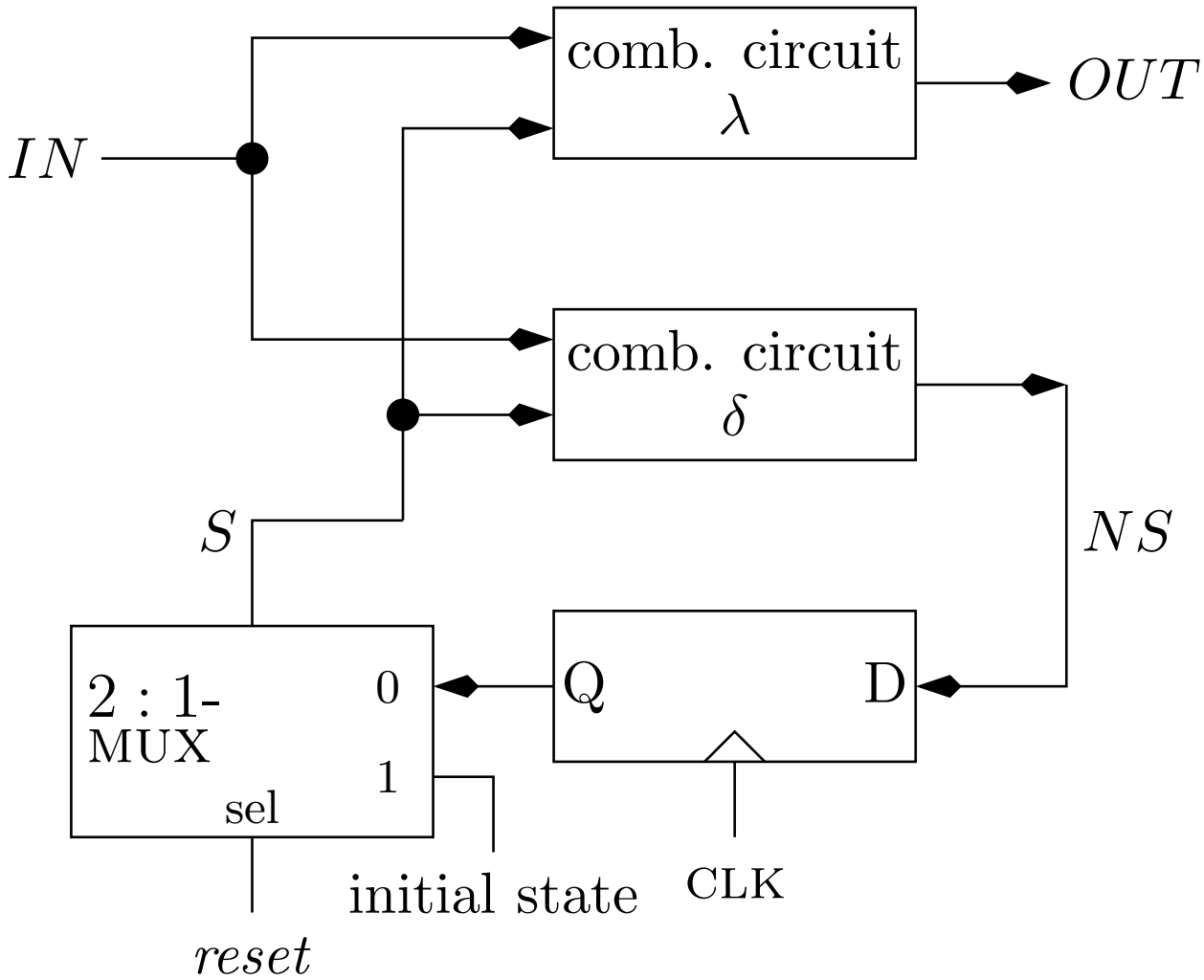
Initialization

- We require that the output of every flip-flop be defined during the first clock cycle. Impossible?
  1. How can we even define the "first" clock cycle?
  2. What is the state of a flip-flop after power on?
  3. How can anything be set or determined after power on?
- Deus ex machina: introduce a reset signal:

$$reset(t) \triangleq \begin{cases} 1 & \text{if } t = 0, \\ 0 & \text{otherwise.} \end{cases}$$

- How is a reset signal generated? How could a reset signal differ from the the output of a flip-flop?
- No solution to this problem within the digital abstraction. All we can try to do is reduce the probability of such an event.
- In practice, a special circuit, called a reset controller, generates a proper reset signal with very high probability. Oddly enough, a reset controller is usually constructed by cascading flip-flops!

Using the reset



Restart "time": If $reset(t) = 1$, then set $t \leftarrow 0$.

Functionality: the canonic form

We denote the logical value of a signal X during the i'th clock cycle by X_i.

Claim

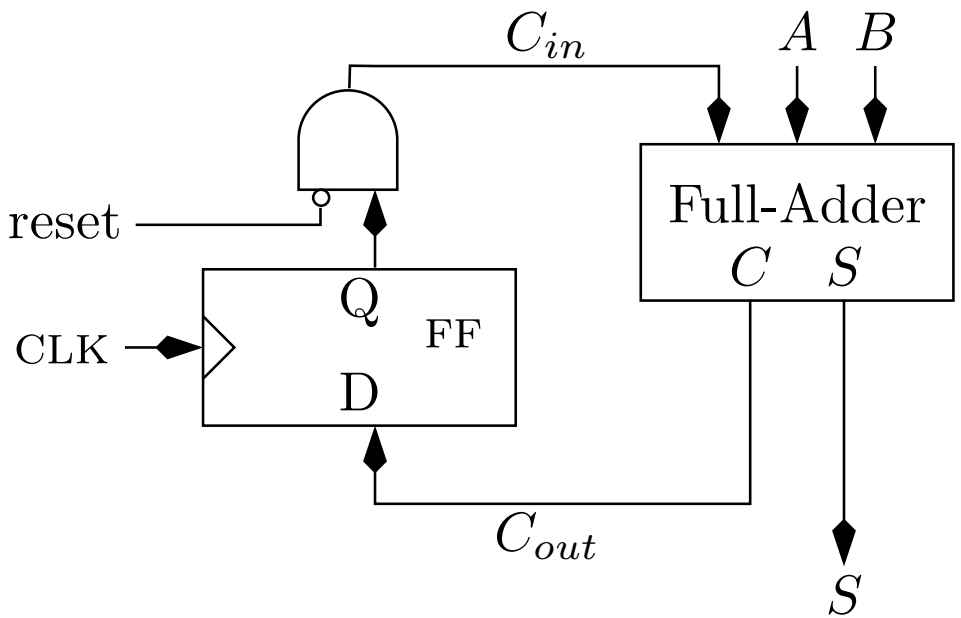
For every $i \geq 0$:

$$S_i = \begin{cases} \textit{initial state} & \textit{if } i = 0 \\ NS_{i-1} & \textit{if } i \geq 1 \end{cases}$$

$$NS_i = \delta(IN_i, S_i)$$

$$OUT_i = \lambda(IN_i, S_i)$$

Sequential Adder with Reset



Note: Mux controlled by reset implemented by an AND-gate.

Sequential Adder with Reset

What happens if $|\{t \mid reset(t) = 1\}| > 1$? If $reset(t) = 1$, then we forget about the past, we treat clock cycle t as the first clock cycle. Formally, we define the last initialization $r(i)$ as follows:

$$r(i) \triangleq \max\{t \leq i : reset(t) = 1\}.$$

Namely, $r(i)$ specifies the last time $reset(t) = 1$ not after cycle i. If $reset_j = 0$, for every $j \leq i$, then $r(i)$ is not defined, and functionality is unspecified. If $r(i)$ is well defined, then the functionality is that, for every $i \geq 0$,

$$\langle A[i : r(i)]\rangle + \langle B[i : r(i)]\rangle = \langle S[i : r(i)]\rangle \pmod{2^{i-r(i)+1}}.$$

Finite State Machines

The functionality of a synchronous circuit in the canonic form is so important that it justifies a term called finite state machines.

Definition

A finite state machine (FSM) is a 6-tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$, where

- Q is a set of states.
- Σ is the alphabet of the input.
- Δ is the alphabet of the output.
- $\delta : Q \times \Sigma \to Q$ is a transition function.
- $\lambda : Q \times \Sigma \to \Delta$ is an output function.
- $q_0 \in Q$ is an initial state.

What does an FSM do?

An FSM is an abstract machine that operates as follows. The input is a sequence $\{x_i\}_{i=0}^{n-1}$ of symbols over the alphabet Σ. The output is a sequence $\{y_i\}_{i=0}^{n-1}$ of symbols over the alphabet Δ. An FSM transitions through the sequence of states $\{q_i\}_{i=0}^{n}$. The state q_i is defined recursively as follows:

$$q_{i+1} \stackrel{\triangle}{=} \delta(q_i, x_i)$$

The output y_i is defined as follows:

$$y_i \stackrel{\triangle}{=} \lambda(q_i, x_i).$$

FSM - terminology

Other terms for a finite state machine are a finite automaton with outputs and transducer. In the literature, an FSM according to our definition is often called a Mealy Machine. Another type of machine, called Moore Machine, is an FSM in which the domain of output function λ is Q (namely, the output is only a function of the state and does not depend on the input).

State Diagrams

FSMs are often depicted using state diagrams.

Definition

The state diagram corresponding to an FSM $\mathcal{A}$ is a directed graph $G = (Q, E)$ with edge labels $(x, y) \in \Sigma \times \Delta$. The edge set E is defined by

$$E \triangleq \{(q, \delta(q, x)) : q \in Q \text{ and } x \in \Sigma\}.$$

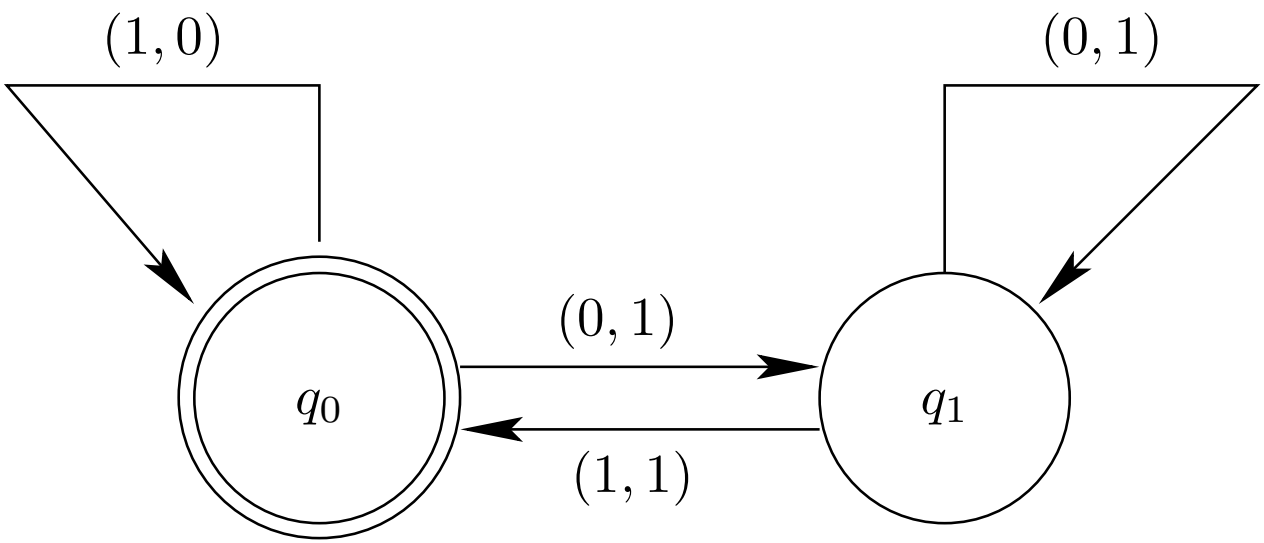
Each edge $(q, \delta(q, x))$ is labeled $(x, \lambda(q, x))$.

The vertex q_0 corresponding to the initial state of an FSM is usually marked in an FSM by a double circle.
We remark that a state diagram is in fact a multi-graph, namely, one allows more than one directed edge between two vertices. Such edges are often called parallel edges. Note that the out-degree of every vertex in a state diagram equals $|\Sigma|$.

Example: A two-state FSM

Consider the FSM $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ depicted in the next figure, where

$$Q = \{q_0, q_1\},$$
$$\Sigma = \Delta = \{0, 1\}.$$



Synthesis and Analysis

Two tasks are often associated with synchronous circuits. These tasks are defined as follows.

1. Analysis: given a synchronous circuit C, describe its functionality by an FSM.
2. Synthesis: given an FSM $\mathcal{A}$, design a synchronous circuit C that implements $\mathcal{A}$.

Analysis: Sync Circuit $\mapsto$ FSM

The task of analyzing a synchronous circuit C is carried out as follows.

1. Define the FSM $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ as follows.
    1. The set of states is $Q \triangleq \{0,1\}^k$, where k denotes the number of flip-flops in C.
    2. Define the initial state q_0 to be the initial outputs of the flip-flops.
    3. $\Sigma = \{0,1\}^\ell$, where ℓ denotes the number of input gates in C.
    4. $\Delta = \{0,1\}^r$, where r denotes the number of output gates in C.
    5. Define the transition function $\delta : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^k$ to be the function implemented by the combinational "part" of C for the inputs of the flip-flops.
    6. Define the output function $\lambda : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^r$ to be the function implemented by the combinational "part" of C for the output gates.

A Counter

Definition

A *counter*(n) is defined as follows.

Inputs: a clock CLK.

Output: $N \in \{0,1\}^n$.

Functionality:

$$\forall t \ : \ \langle N_t \rangle = t (\text{mod } 2^n)$$

No input?! Input is "implied": it is the (missing) reset signal!

Counter Implementation

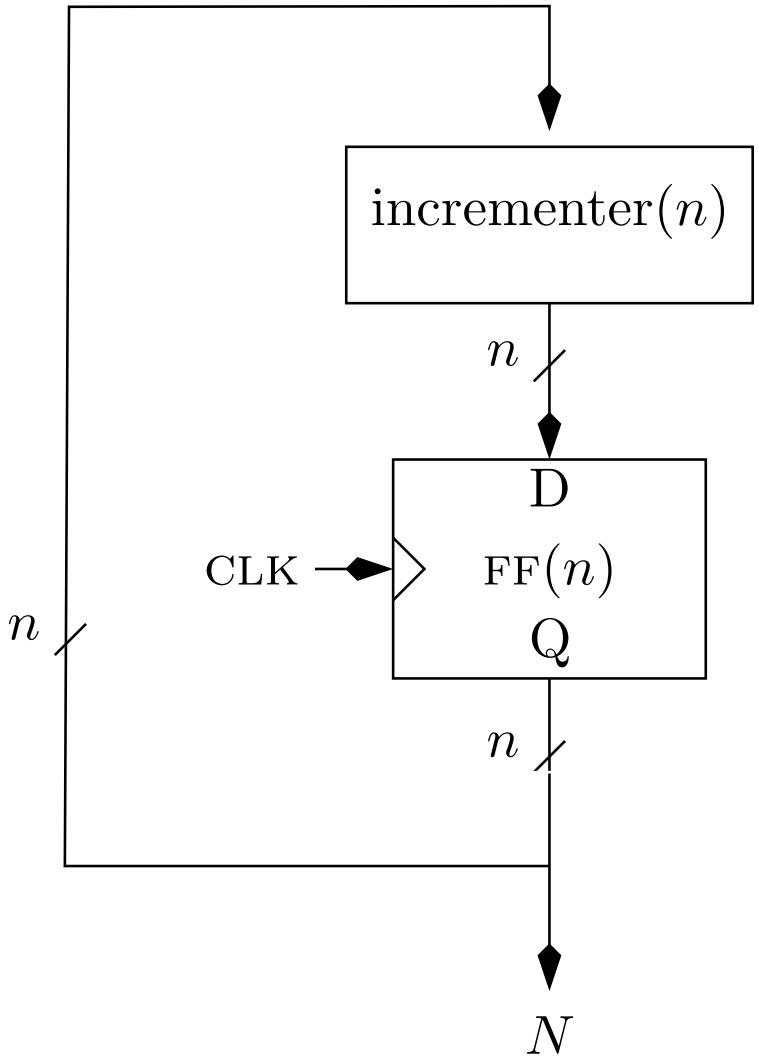


Figure: A synchronous circuit that implements a counter.

Counter Analysis

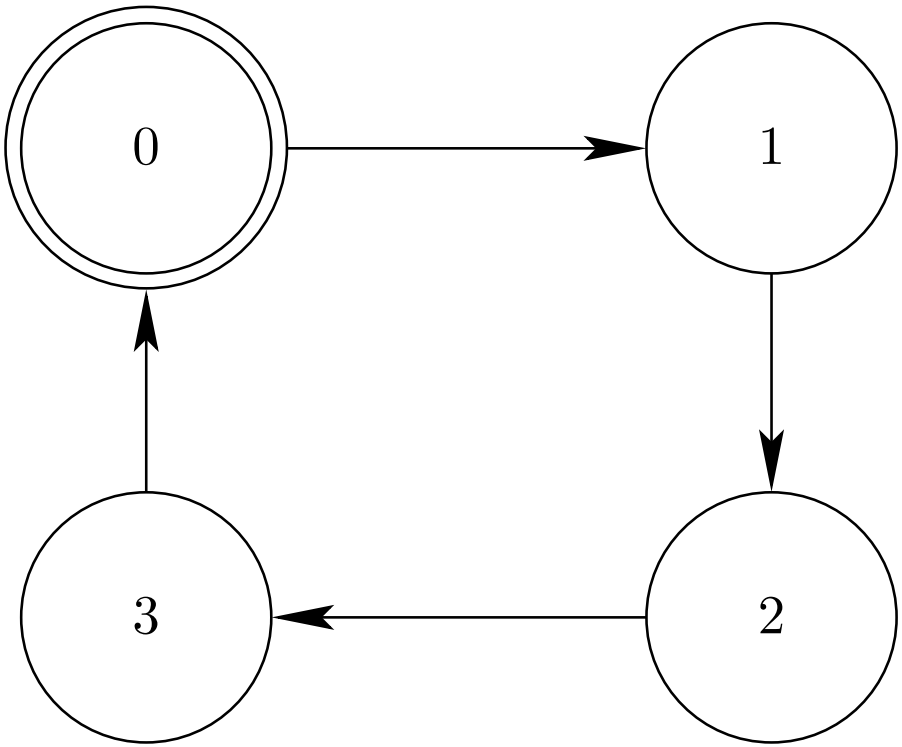


Figure: An FSM of a counter(2). The output always equals binary representation of the state from which the edge emanates.

A Counter with input

Definition

A *counter*(n) is defined as follows.

Inputs: $X \in \{0,1\}$ and a clock CLK.

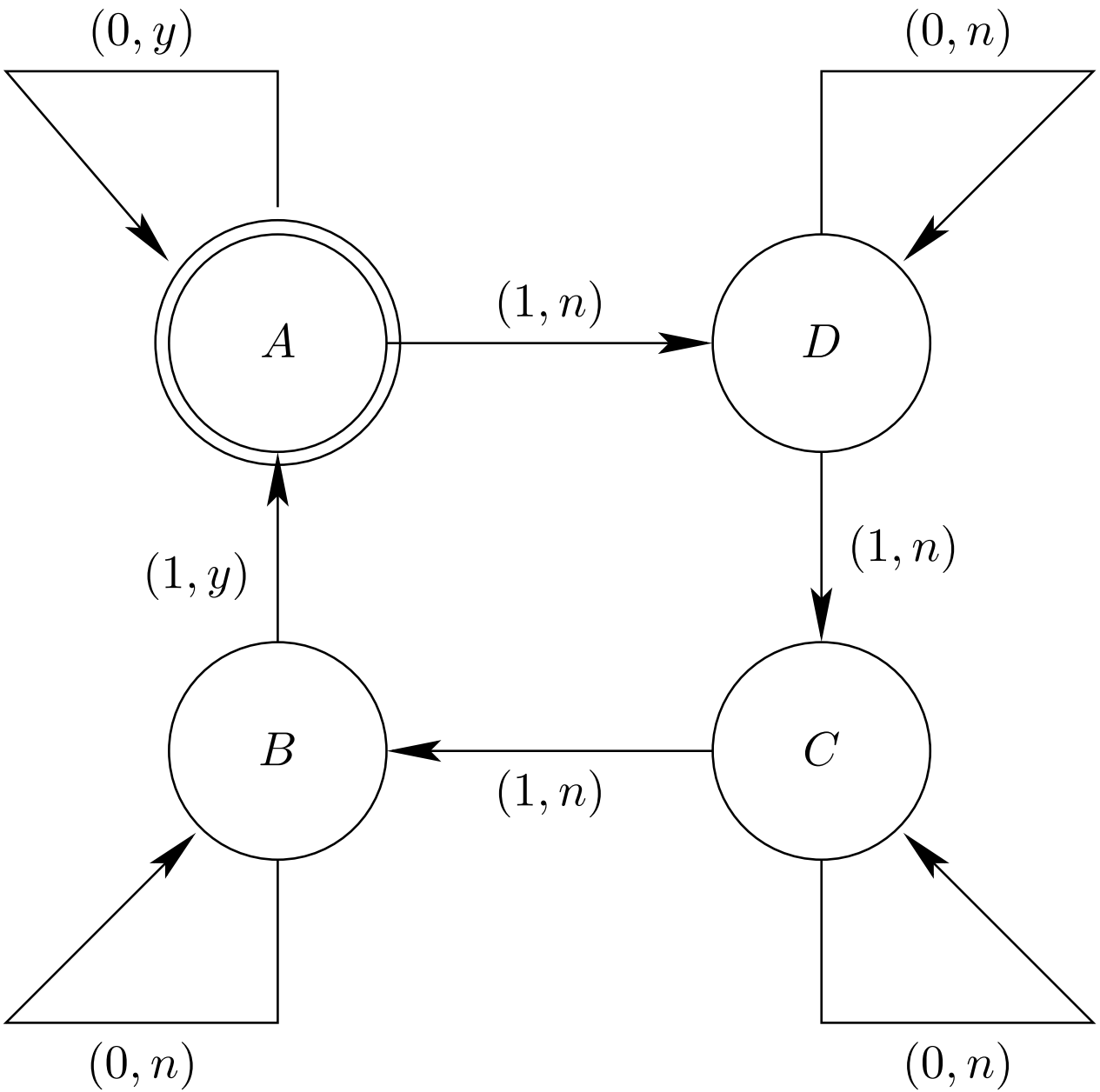
Output: $N \in \{0,1\}^n$.

Functionality:

$$\forall t \ : \ \langle N_t \rangle = \sum_{i=0}^{t} X_i \pmod{2^n}$$

Implementation of Counter with Input

Analysis of Counter with Input for $n = 2$



Sequential Adder: Analysis

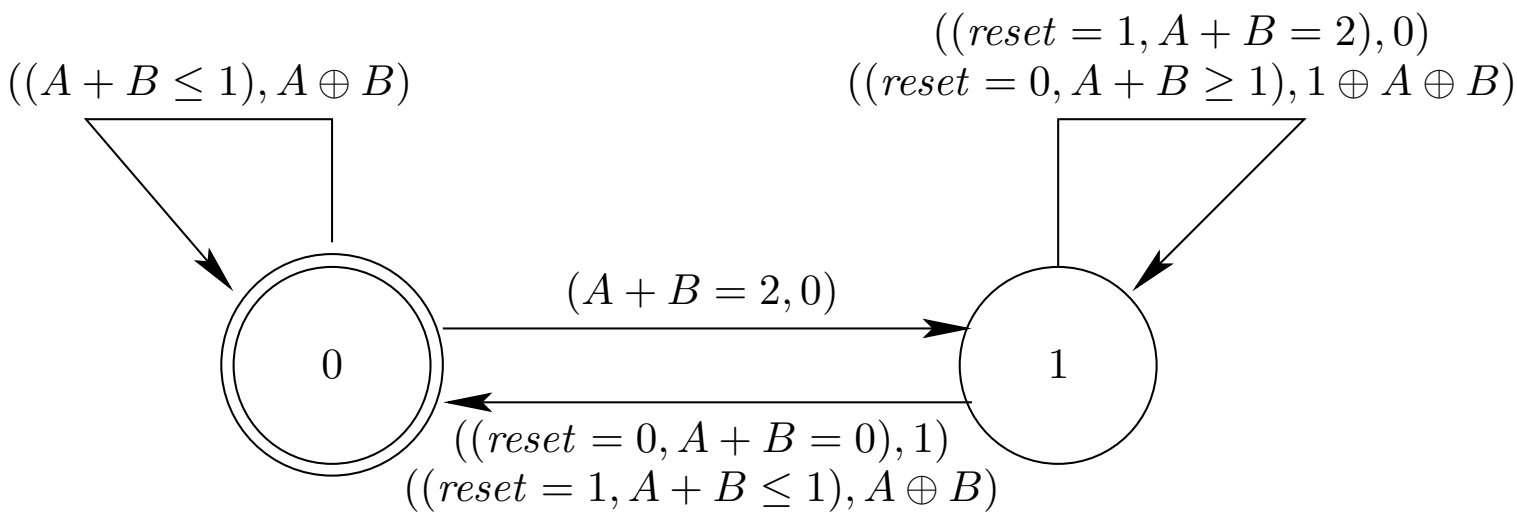


Figure: an FSM of a sequential adder (each transition is labeled by a pair: the condition that the input satisfies and the value of the output).

Revisiting Shift Registerers

Recall the definition of a shift register of n bits, that is:

Inputs: $D[0](t)$ and a clock CLK.

Output: $Q[n-1](t)$.

Functionality: $Q[n-1](t+n) = D[0](t)$.

Implementation of Shift Register

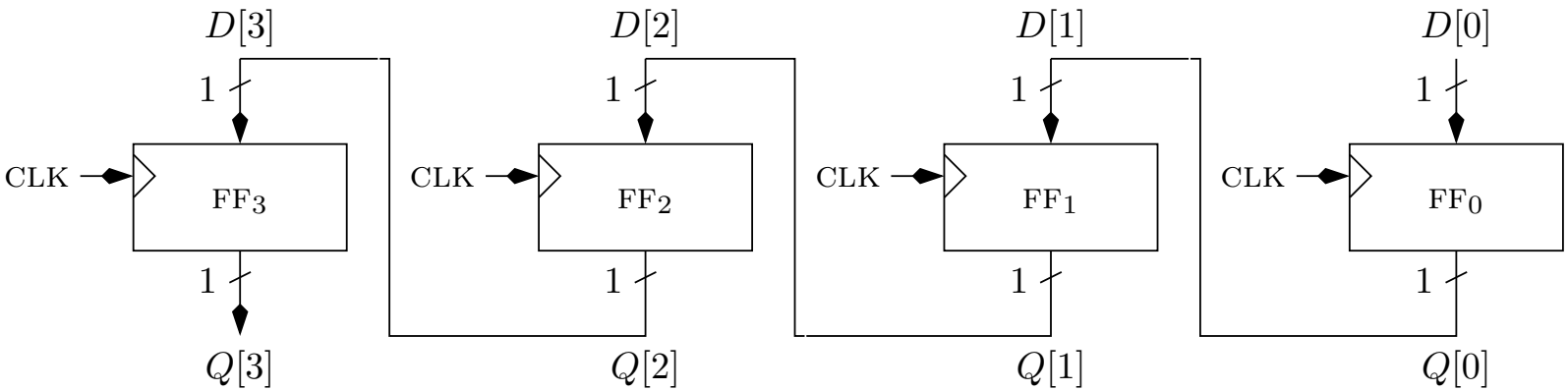
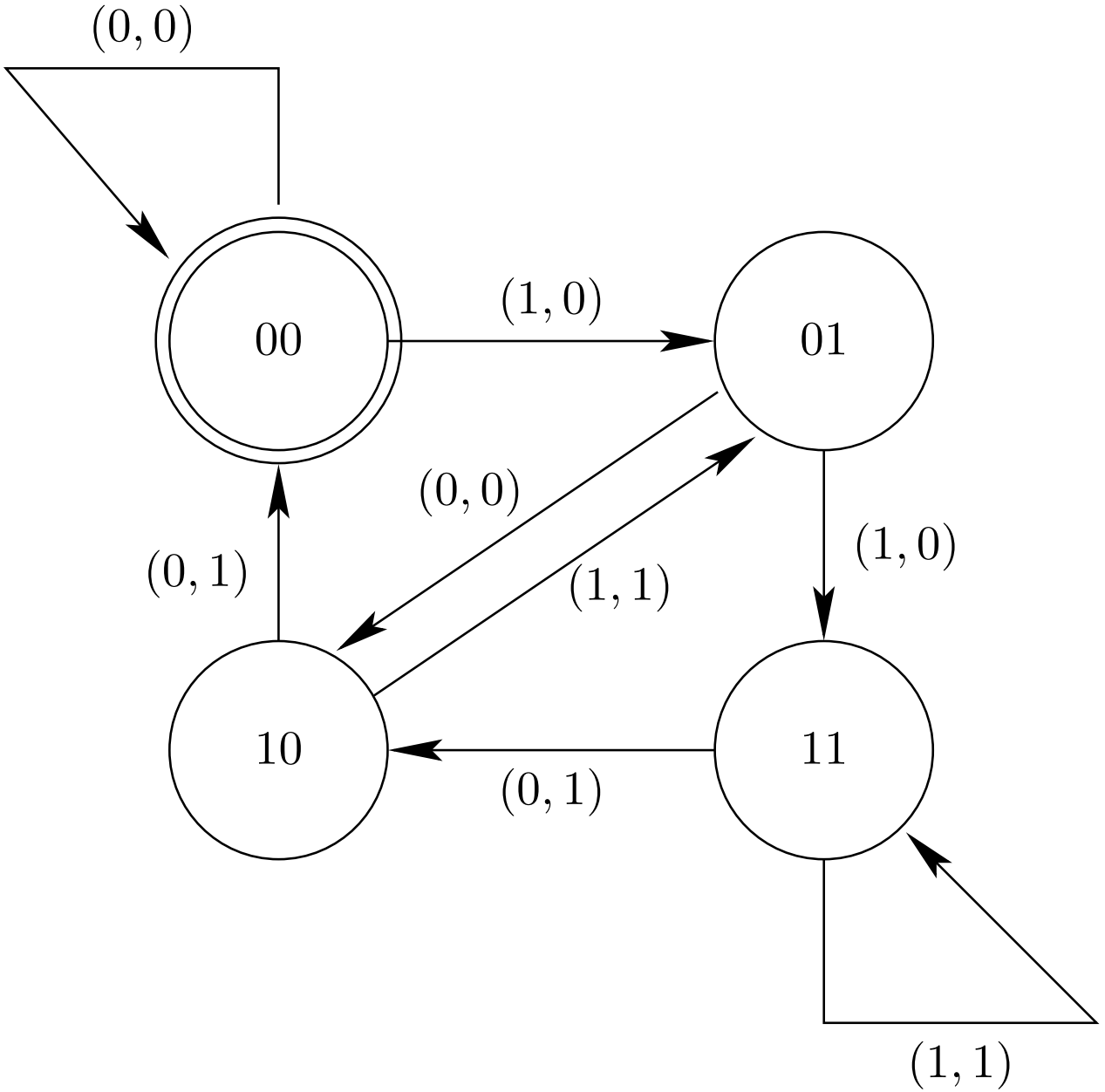


Figure: A 4-bit shift register.

Analysis of Shift Register for $n = 2$



Revisiting RAM

Definition

A $\text{RAM}(2^n)$ is specified as follows.

Inputs: $Address[n-1:0](t) \in \{0,1\}^n, D_{\text{in}}(t) \in \{0,1\}$, $R/\overline{W}(t) \in \{0,1\}$ and a clock CLK.

Output: $D_{\text{out}}(t) \in \{0,1\}$.

Functionality : The functionality of a RAM is specified by the following program:

1. data: array $M[2^n-1:0]$ of bits.
2. initialize: $\forall i : M[i] \leftarrow 0$.
3. For $t = 0$ to ∞ do
   1. $D_{\text{out}}(t) = M[\langle Address \rangle](t)$.
   2. For all $i \neq \langle Address \rangle$: $M[i](t+1) \leftarrow M[i](t)$.
   3.

$$M[\langle Address \rangle](t+1) \leftarrow \begin{cases} D_{\text{in}}(t) & \text{if } R/\overline{W}(t) = 0 \\ M[\langle Address \rangle](t) & \text{else.} \end{cases}$$

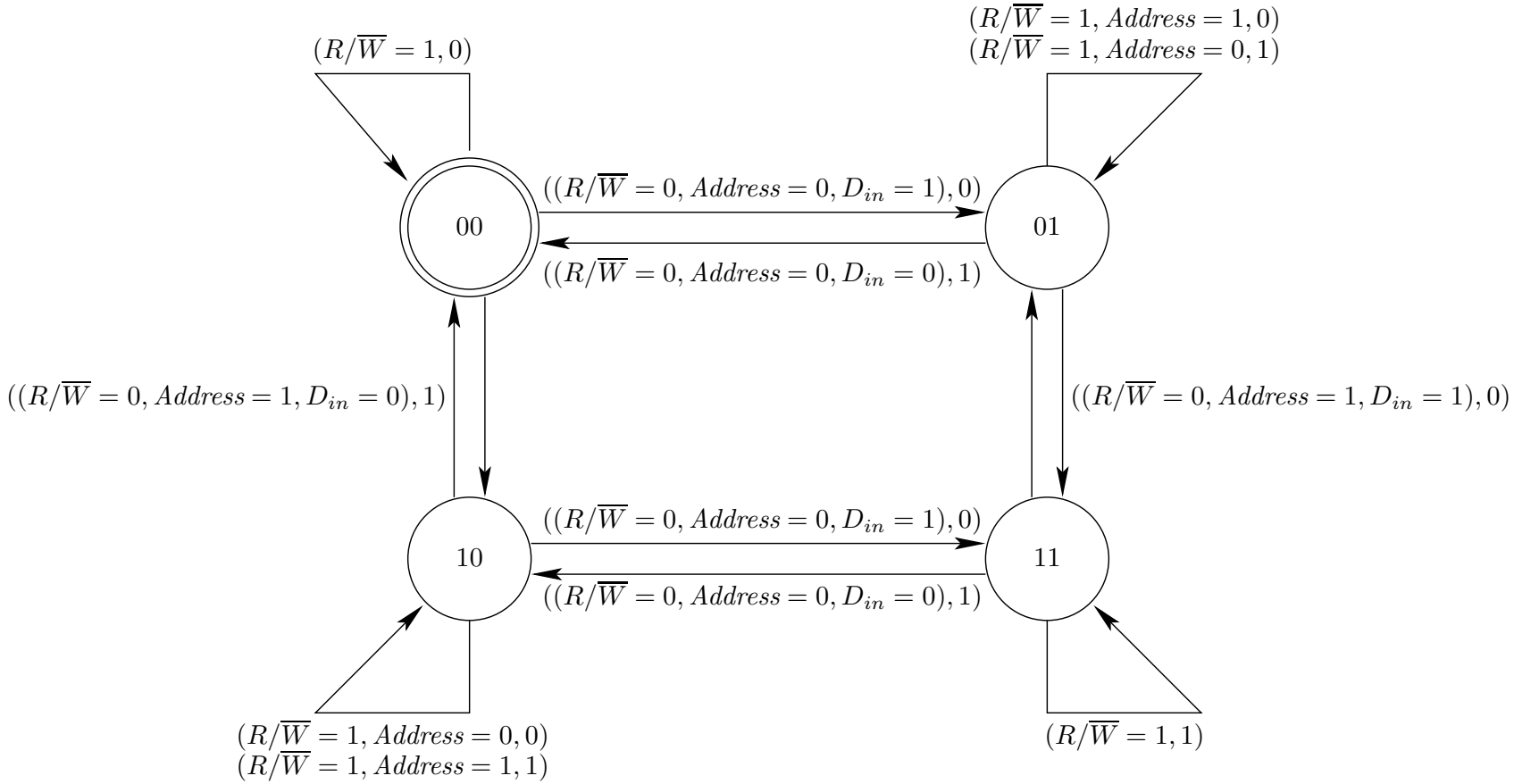


Figure: A (partial) FSM of a $\text{RAM}(2^1)$ (the "legend" of the edge labels:$((D_{\text{in}}, address, R/\overline{W}), D_{\text{out}})$).

Effect of Adding Initialization to a Synchronous Circuit

- C is a synchronous circuit without an initialization signal (but we assume FFs output a specific value in $t = 0$).
- Introduce an initialization signal *reset* that initializes the outputs of all flip-flops (namely, it cause the outputs of the flip-flops to equal a value that encodes the initial state).
- How? add a MUX after every FF that selects Q or initial-state based on *reset*.
- Denote the new synchronous circuit by $\hat{C}$.
- Let $\mathcal{A}$ and $\hat{\mathcal{A}}$ denote the FSMs that model the functionality of C and $\hat{C}$, respectively.
- What is the relation between $\mathcal{A}$ and $\hat{\mathcal{A}}$?

Adding the initialization signal to an FSM - cont

Theorem

Let $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ denote the FSM that models the functionality of the synchronous circuit C. Let $\hat{\mathcal{A}} = \langle Q', \Sigma', \Delta', \delta', \lambda', q_0' \rangle$ denote the FSM that models the synchronous circuit $\hat{C}$. Then,

$$Q' \stackrel{\triangle}{=} Q,$$

$$q_0' \stackrel{\triangle}{=} q_0,$$

$$\Sigma' \stackrel{\triangle}{=} \Sigma \times \{0, 1\},$$

$$\Delta' \stackrel{\triangle}{=} \Delta,$$

$$\delta'(q, (\sigma, \mathit{reset})) \stackrel{\triangle}{=} \begin{cases} \delta(q, \sigma), & \mathit{if\ reset} = 0, \\ \delta(q_0, \sigma), & \mathit{if\ reset} = 1, \end{cases}$$

$$\lambda'(q, (\sigma, \mathit{reset})) \stackrel{\triangle}{=} \begin{cases} \lambda(q, \sigma), & \mathit{if\ reset} = 0, \\ \lambda(q_0, \sigma), & \mathit{if\ reset} = 1. \end{cases}$$

Synthesis: FSM $\mapsto$ Sync Circuit

Given an FSM $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$, the task of designing a synchronous circuit C that implements $\mathcal{A}$ is carried out as follows.

1. Encode Q, Σ and Δ by binary strings. Formally, let f, g, h denote one-to-one functions, where
$$f : Q \to \{0,1\}^k$$
$$g : \Sigma \to \{0,1\}^\ell$$
$$h : \Delta \to \{0,1\}^r.$$
2. Design a combinational circuit C_δ that implements the (partial) Boolean function $B_\delta : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^k$ defined by
$$B_\delta(f(x), g(y)) \stackrel{\triangle}{=} f(\delta(x,y)), \text{ for every } (x,y) \in Q \times \Sigma.$$
3. Design a combinational circuit C_λ that implements the (partial) Boolean function $B_\lambda : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^r$
$$B_\lambda(f(x), g(y)) \stackrel{\triangle}{=} h(\lambda(x,y)), \text{ for every } (x,y) \in Q \times \Sigma.$$

Synthesis - cont

- How many flip-flops are required? $f : Q \rightarrow \{0,1\}^k$ is one-to-one. So

$$k \geq \log_2 |Q|$$

- It is not clear that minimizing k is a always a good idea. Certain encodings lead to more complicated Boolean functions B_δ and B_λ.
- The question of selecting a "good" encoding is a very complicated task, and there is no simple solution to this problem.

Example: A two-state FSM

Consider the FSM $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ depicted in the next figure, where

$$Q = \{q_0, q_1\},$$
$$\Sigma = \Delta = \{0, 1\}.$$

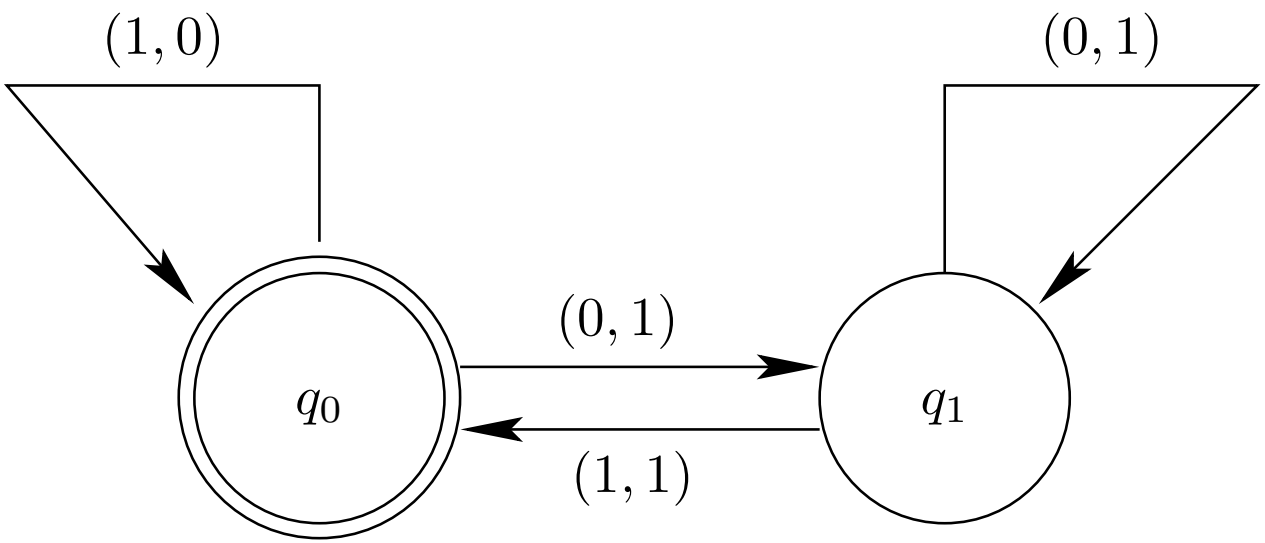


Figure: A two-state FSM.

Two-State FSMs: Synthesis

Given an FSM $\mathcal{A} = \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$, the synchronous circuit C that is obtained by executing the synthesis procedure is as follows. We encode Q, Σ and Δ by binary strings Formally, let f, g, h denote one-to-one functions, where

$$\begin{aligned} f &: Q \to \{0,1\} \\ g &: \Sigma \to \Sigma \\ h &: \Delta \to \Delta, \end{aligned}$$

where

$$f(q_0) = 0, f(q_1) = 1,$$

and

$$\forall x \in \{0,1\} : g(x) = h(x) = x.$$

Two-State FSMs: Synthesis - C_δ

We design a combinational circuit C_δ that implements the Boolean function $B_\delta : \{0,1\}^2 \to \{0,1\}$ defined by

$$B_\delta(f(x), g(y)) \stackrel{\triangle}{=} f(\delta(x,y)), \text{ for every } (x,y) \in Q \times \Sigma.$$

$f(x)$	$g(y)$	$f(\delta(x,y))$
0	0	1
1	0	1
0	1	0
1	1	0

Table: The truth table of B_δ.

It follows that $B_\delta(f(x), g(y)) = \text{NOT}(g(y))$.

Two-State FSMs: Synthesis - C_λ

We design a combinational circuit C_λ that implements the Boolean function $B_\lambda : \{0,1\}^2 \to \{0,1\}$ defined by

$$B_\lambda(f(x), g(y)) \stackrel{\triangle}{=} h(\lambda(x,y)), \text{ for every } (x,y) \in Q \times \Sigma.$$

$f(x)$	$g(y)$	$h(\lambda(x,y))$
0	0	1
1	0	1
0	1	0
1	1	1

Table: The truth table of B_λ.

It follows that $B_\lambda(f(x), g(y)) = f(x) \vee \overline{g(y)}$.

Two-State FSMs: Synthesis - the Synch. circuit C

The synchronous circuit in canonic form constructed from a flip-flops and two combinational circuits is depicted in Figure **??**.

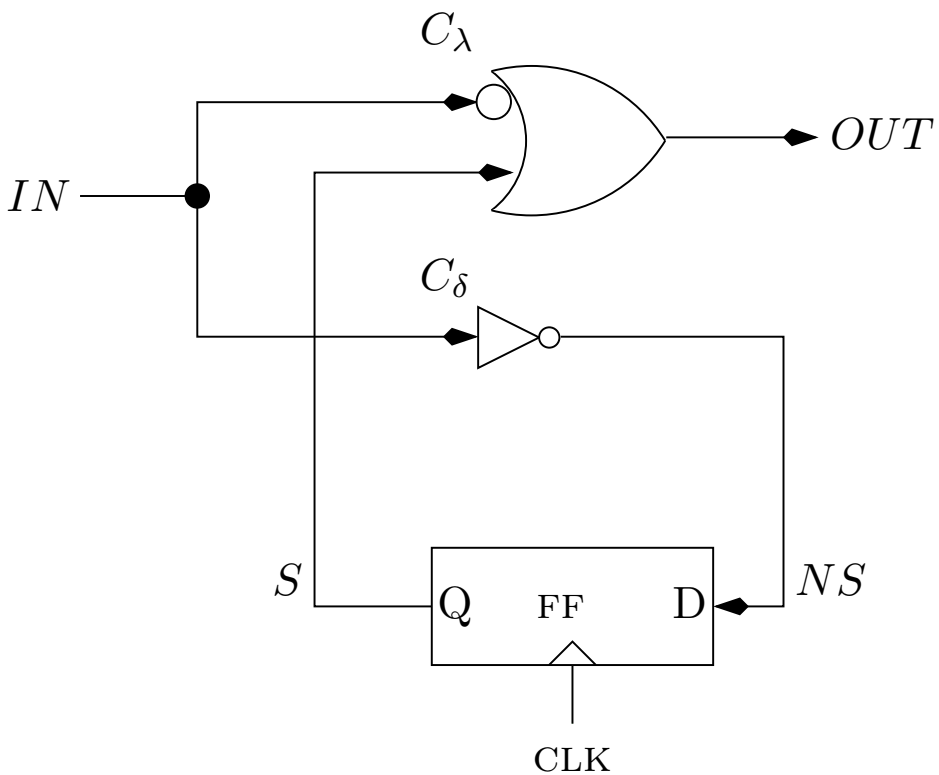


Figure: Synthesis of $\mathcal{A}$.

Summary of Part 2

- Definition of synchronous circuits.
- Simulation algorithm.
- Synchronous circuits in canonic form.
- Initialization & *reset* signal.
- Functionality: finite-state machines & state diagrams.
- Analysis and synthesis of synchronous circuits.
- FSM's are not a useful model for synchronous circuit with many FF's because $|States| = 2^{|FF's|}$.