Digital Logic Design: a rigorous approach ©

Chapter 20: Synchronous Modules

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

December 29, 2014

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

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

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

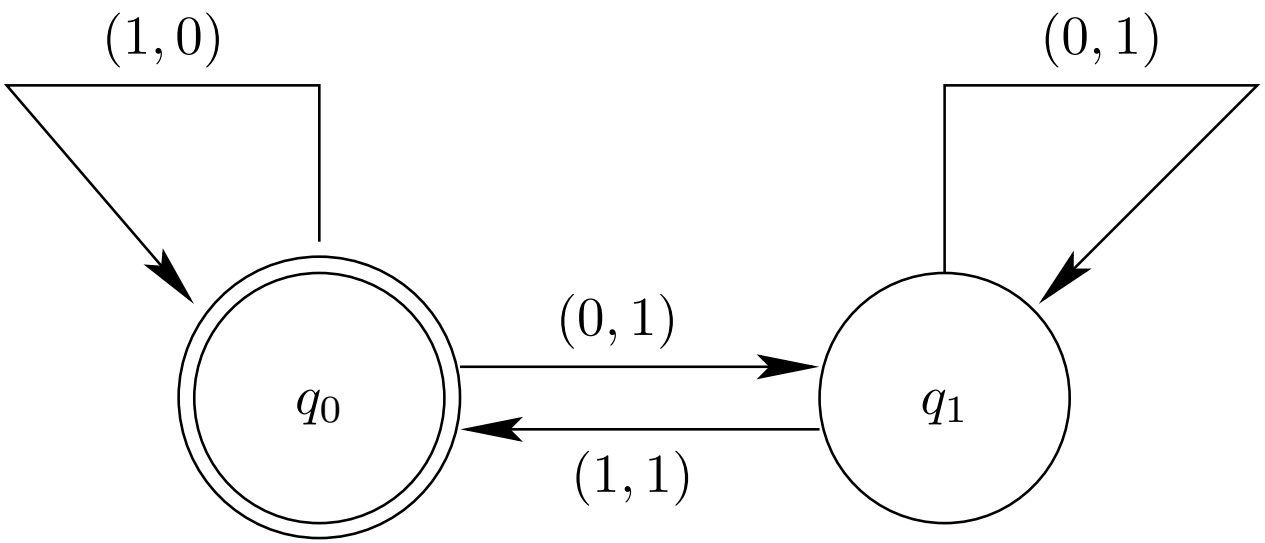

Figure: A two-state FSM.

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 **??**.

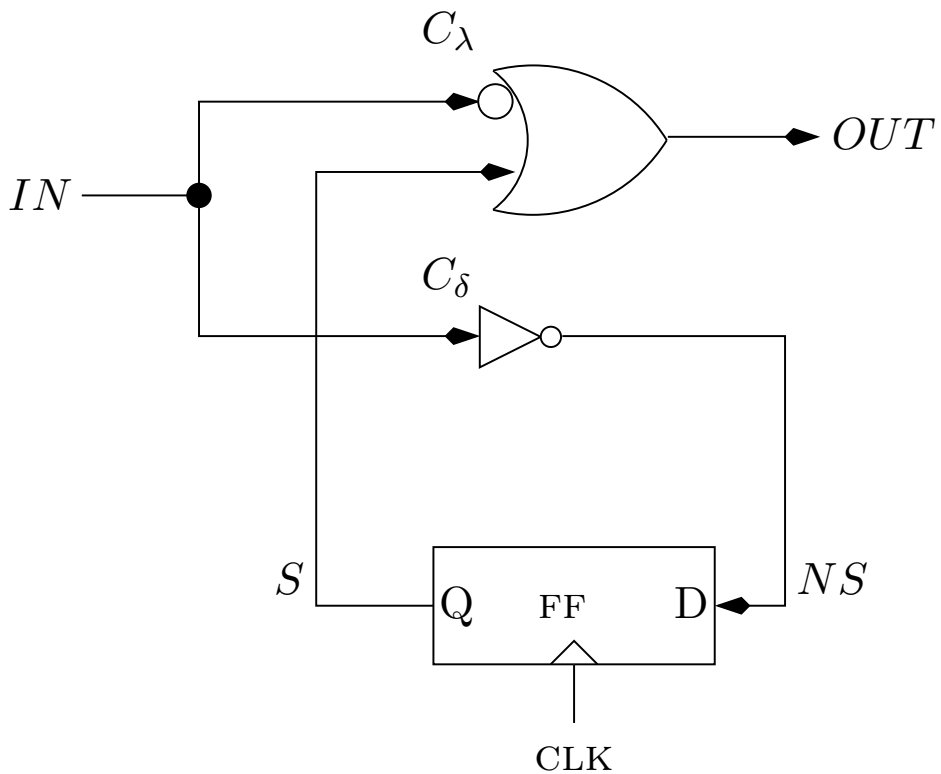

Figure: Synthesis of $\mathcal{A}$.

Sequential Adder

Definition

A *sequential adder* is defined as follows.

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

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

Functionality: The *reset* signal is an initialization signal that satisfies:

$$reset_i = \begin{cases} 1 & \text{if } i = 0, \\ 0 & \text{if } i > 0. \end{cases}$$

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 (cont.)

What happens if the value of the input *reset* equals 1 in more than once cycle? The above definition means that if $reset_i = 1$, then we forget about the past, we treat clock cycle (t_i, t_{i+1}) as the first clock cycle.
Formally, we define the last initialization $r(i)$ as follows:

$$r(i) \stackrel{\triangle}{=} \max\{j \leq i : reset_j = 1\}.$$

Namely, $r(i)$ specifies the last time $reset_j = 1$ 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 specification is that, for every $i \geq 0$,

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

Sequential Adder: Implementation

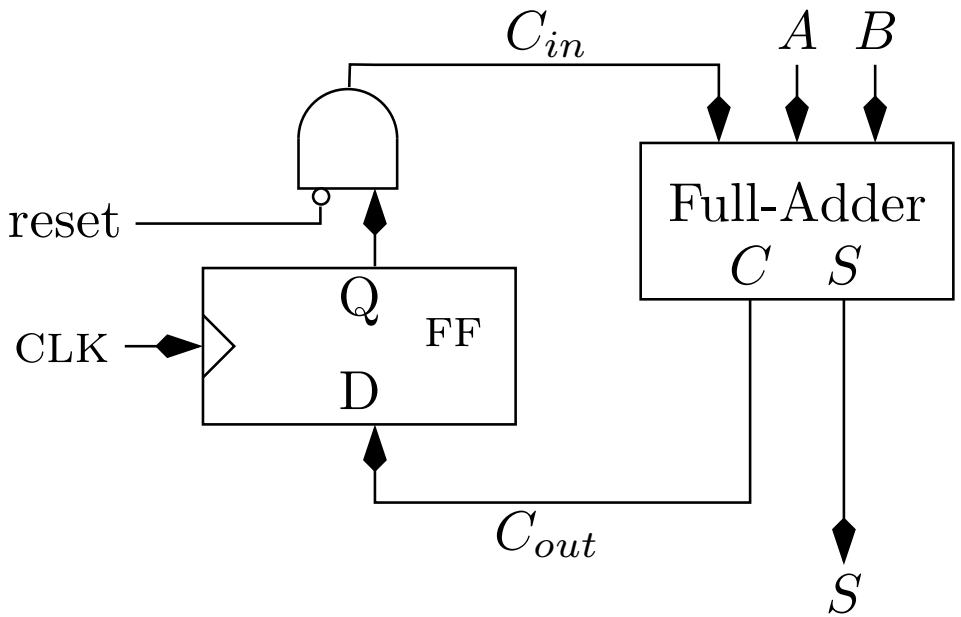

Figure: A synchronous circuit that implements a sequential adder.

Sequential Adder: Implementation - 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:

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

Sequential Adder: Implementation - correctness (cont.)

Proof.

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

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

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

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

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

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

Sequential Adder: Analysis

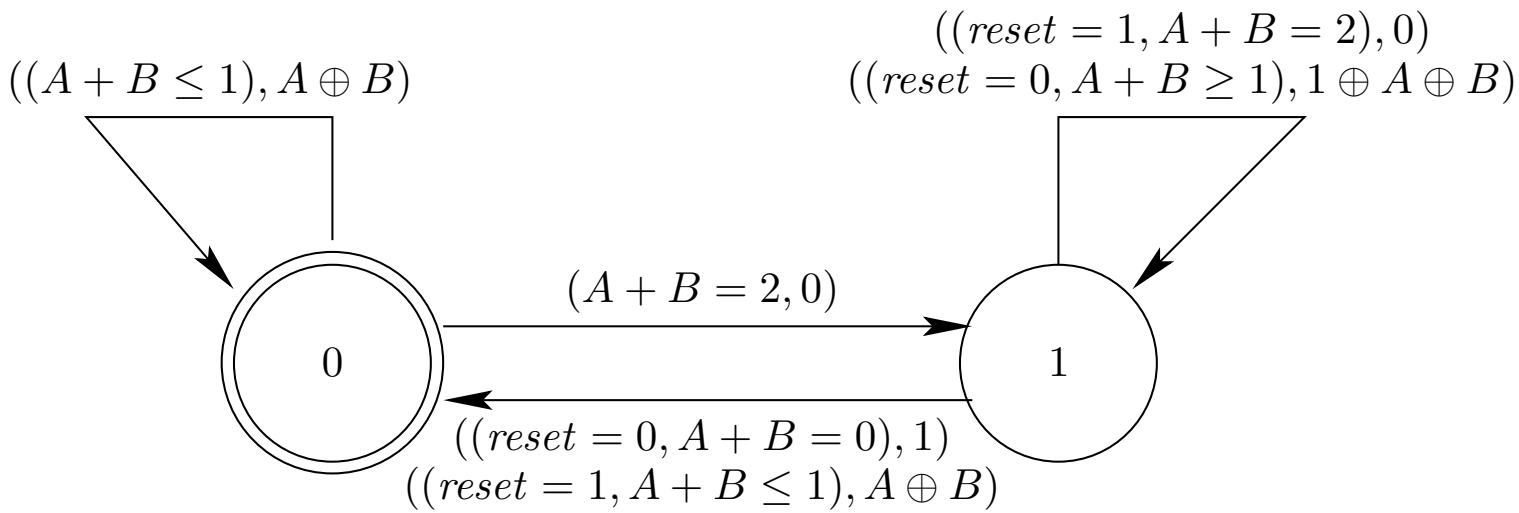

Figure: an FSM of a sequential adder (each transition is labeled by a pair: the condition that the input satisfies and the value of the output).

Sequential Adder: Executing Min-$\Phi(C)$.

Let C denote the Sequential Adder that we have just implemented.

Assume that all the parameters equal to '1'. Assume that a full adder is implemented by a single gate.

Execute the Min-$\Phi(C)$ algorithm. Note that the *reset* signal is also an input signal, hence we should assign a weight to the input gate that feeds it as well.

The "heaviest" path is of weight 5 (*reset* input gate $\rightarrow$ NOT gate $\rightarrow$ AND gate $\rightarrow$ Full adder gate $\rightarrow$ the output gate the corresponds to the D port), hence $\varphi^* = 5$.

Adding the initialization signal to an FSM

- 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? Replace each edge triggered D-flop-flop by an edge triggered D-flip-flop with a reset input. The *reset* signal is fed to the reset input port of each flip-flop.
- 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,

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

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

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

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

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

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

A Counter

Definition

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

Inputs: a clock CLK.

Output: $\{N_i\}_i$, where $N_i \in \{0,1\}^n$.

Functionality: For every i, the number of clock cycles (mod 2^n) since the last *reset* equals N_i.

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

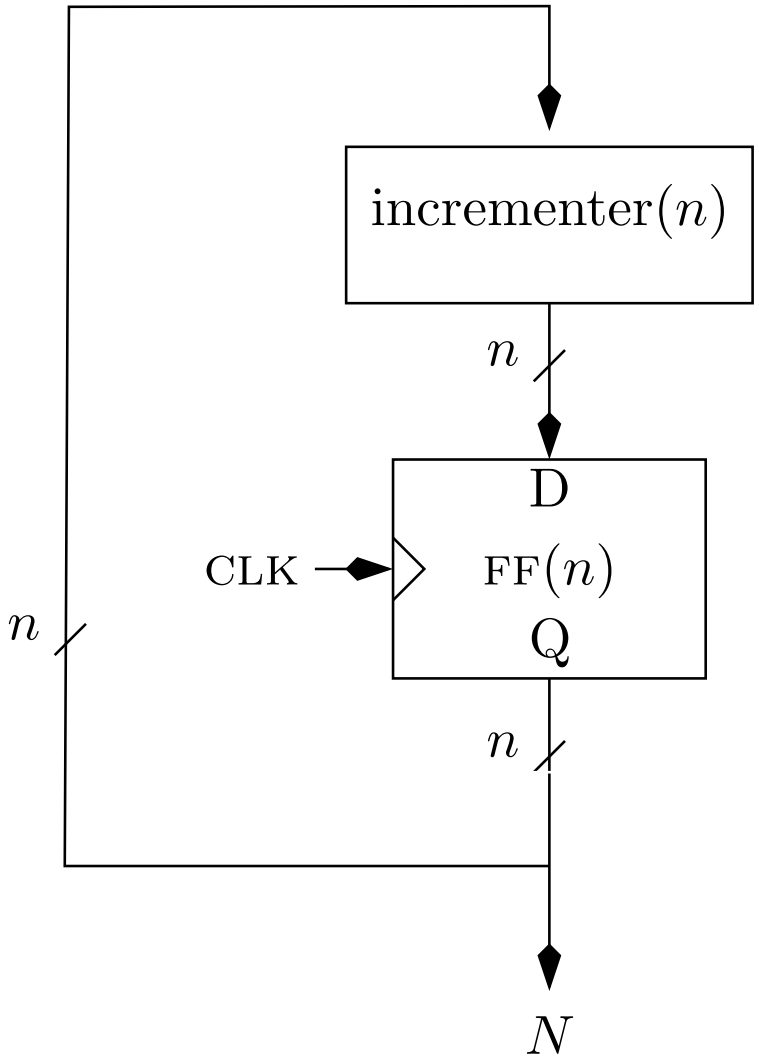

Figure: A synchronous circuit that implements a counter.

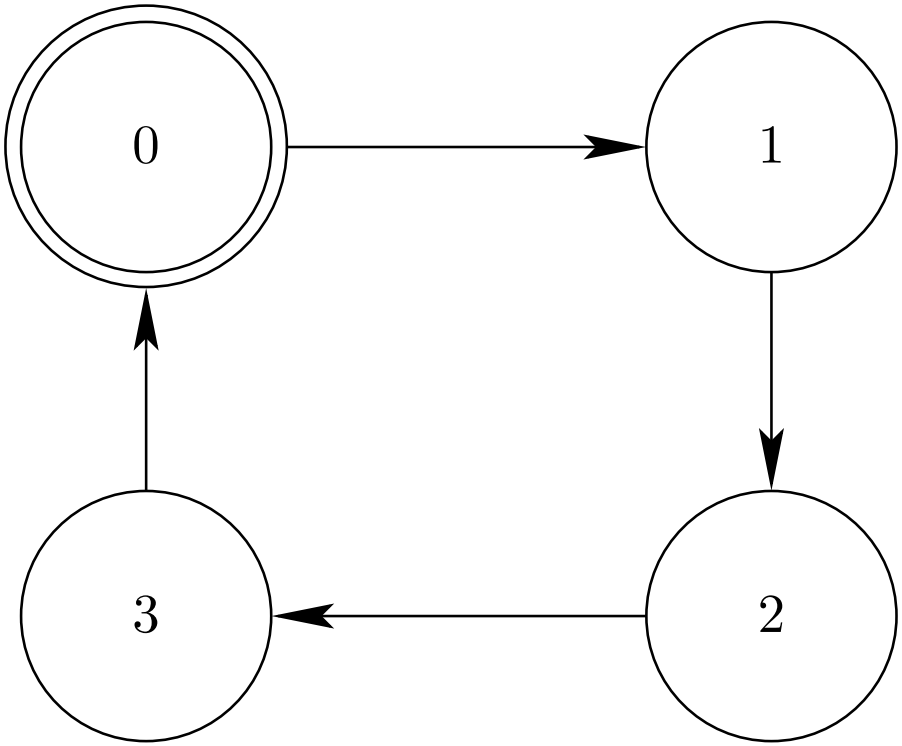

Figure: An FSM of a counter(2). The output always equals the state from which the edge emanates.

Revisiting Shift Registerers

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

Inputs: $D[0](t)$ and a clock $\textsc{clk}$.

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

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

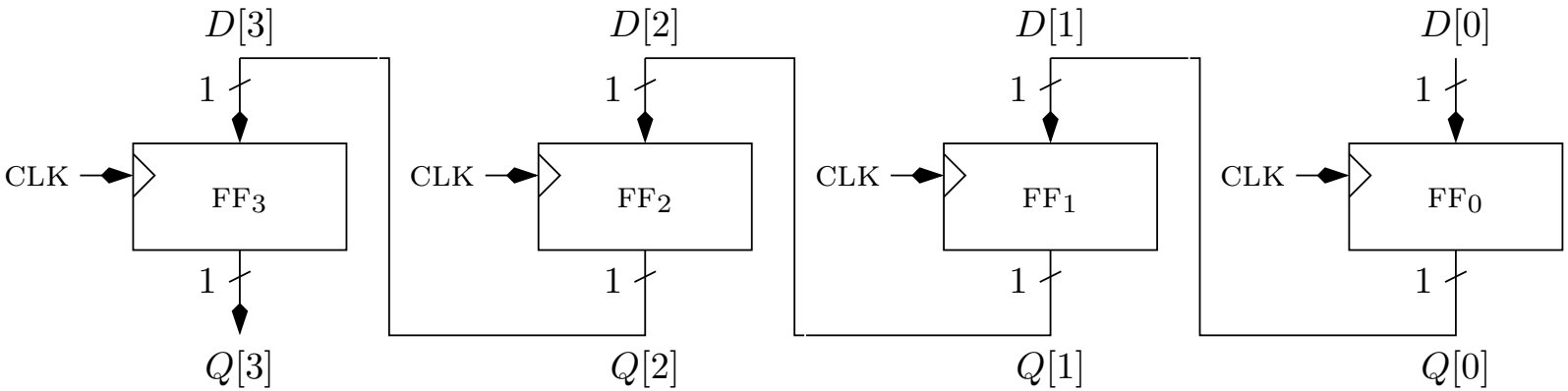

Figure: A 4-bit shift register.

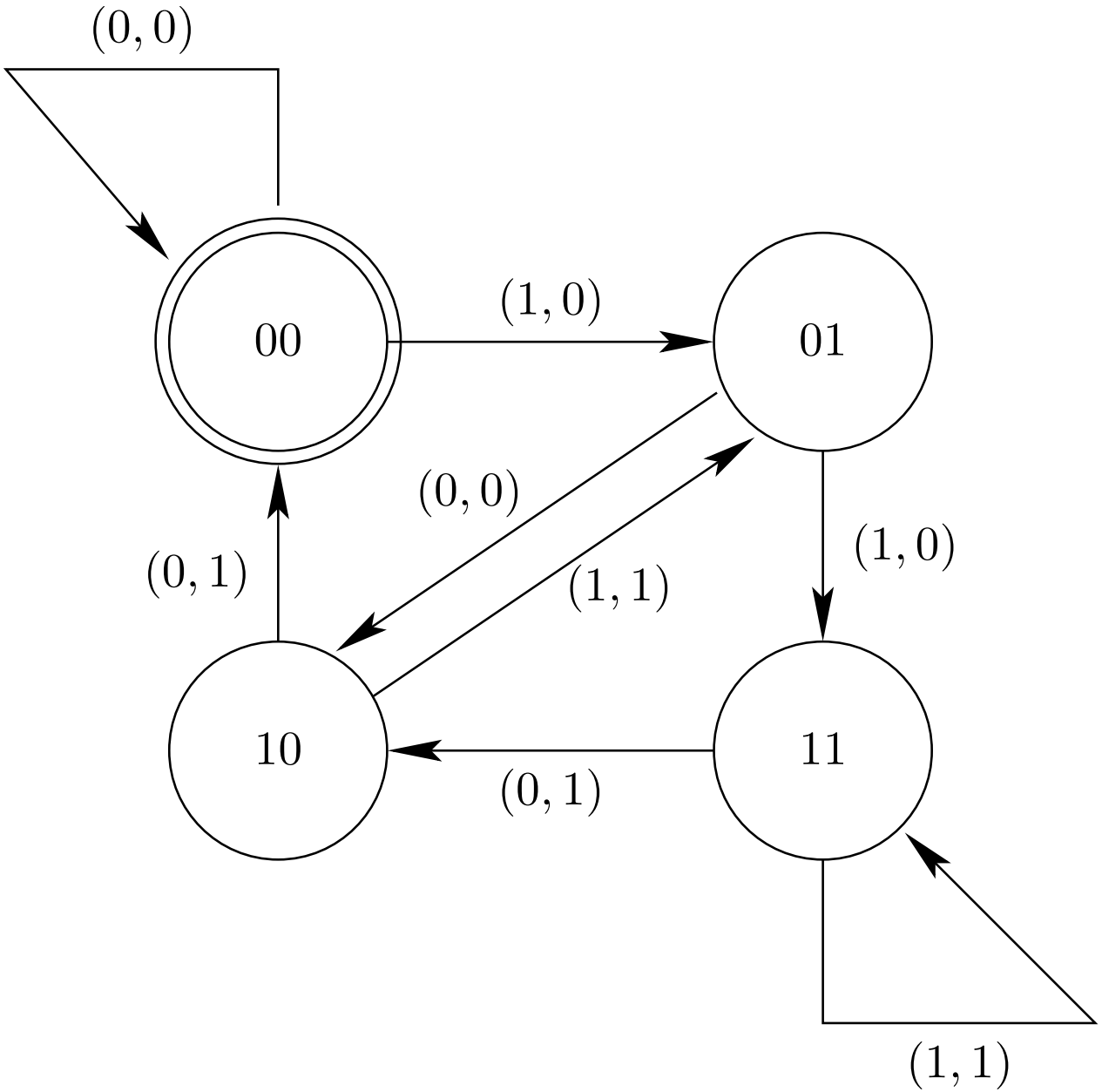

Figure: FSM of a 2-bit shift register, also called *De Bruijn Graph*.

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.

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

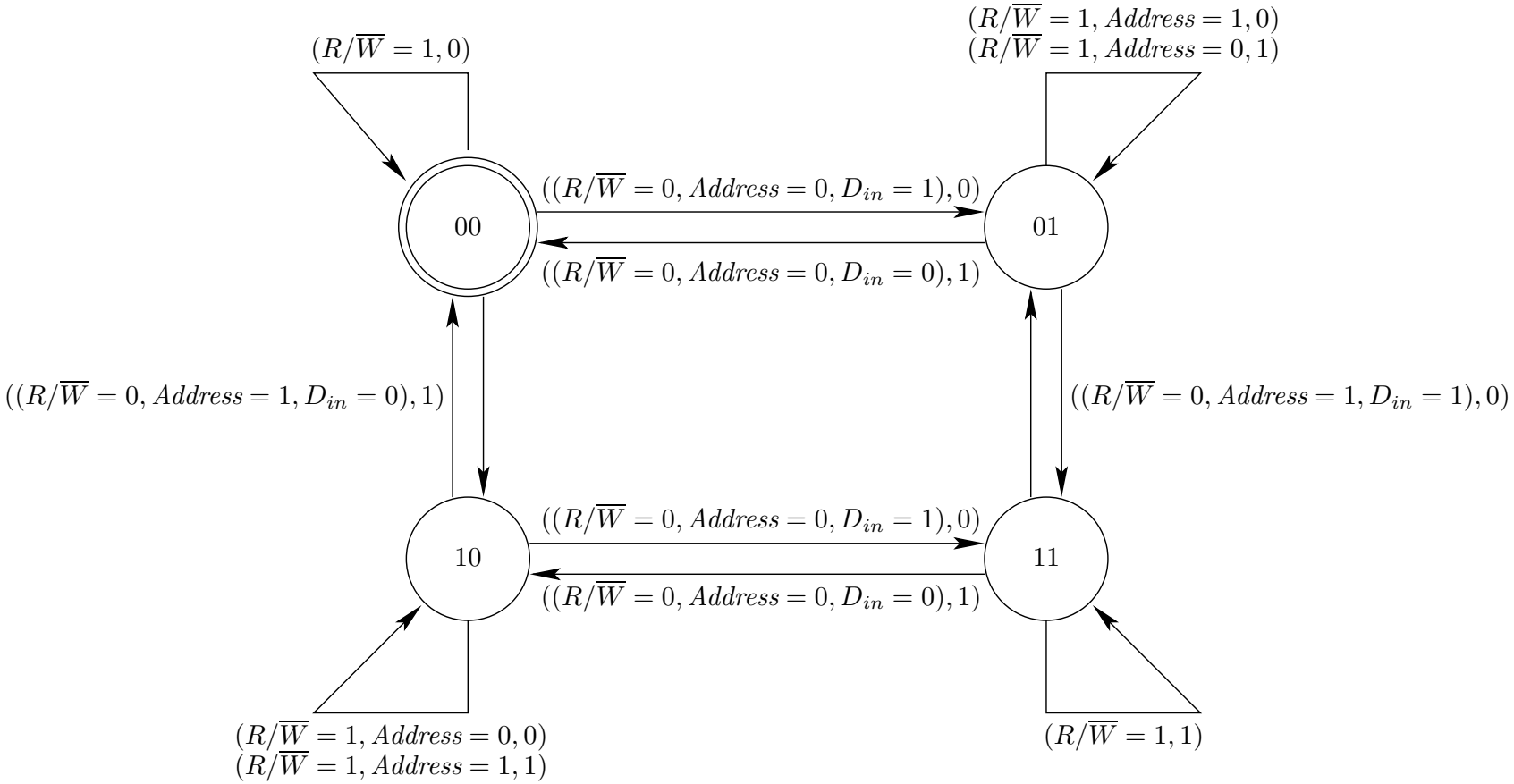

Figure: A (partial) FSM of a $\mathrm{RAM}(2^1)$ (the "legend" of the edge labels:$((D_{\mathsf{in}}, \mathit{address}, R/\overline{W}), D_{\mathsf{out}})$).

Summary

- We presented a few synchronous circuits and their corresponding FSMs. We started by synthesizing a two-state FSM. We then specified, implemented, and analyzed a few synchronous circuits such as: a sequential adder, a counter, a shift register, and a RAM.
- We presented a general method for introducing initialization to a synchronous circuit and to its corresponding FSM.
- When the number flip-flops in a synchronous circuit is large, such as $\text{RAM}(2^n)$, it is not very useful to model its functionality by an FSM.