

Memory Elements III

CS31

Pascal Van Hentenryck



Overview

Memory Elements (III)

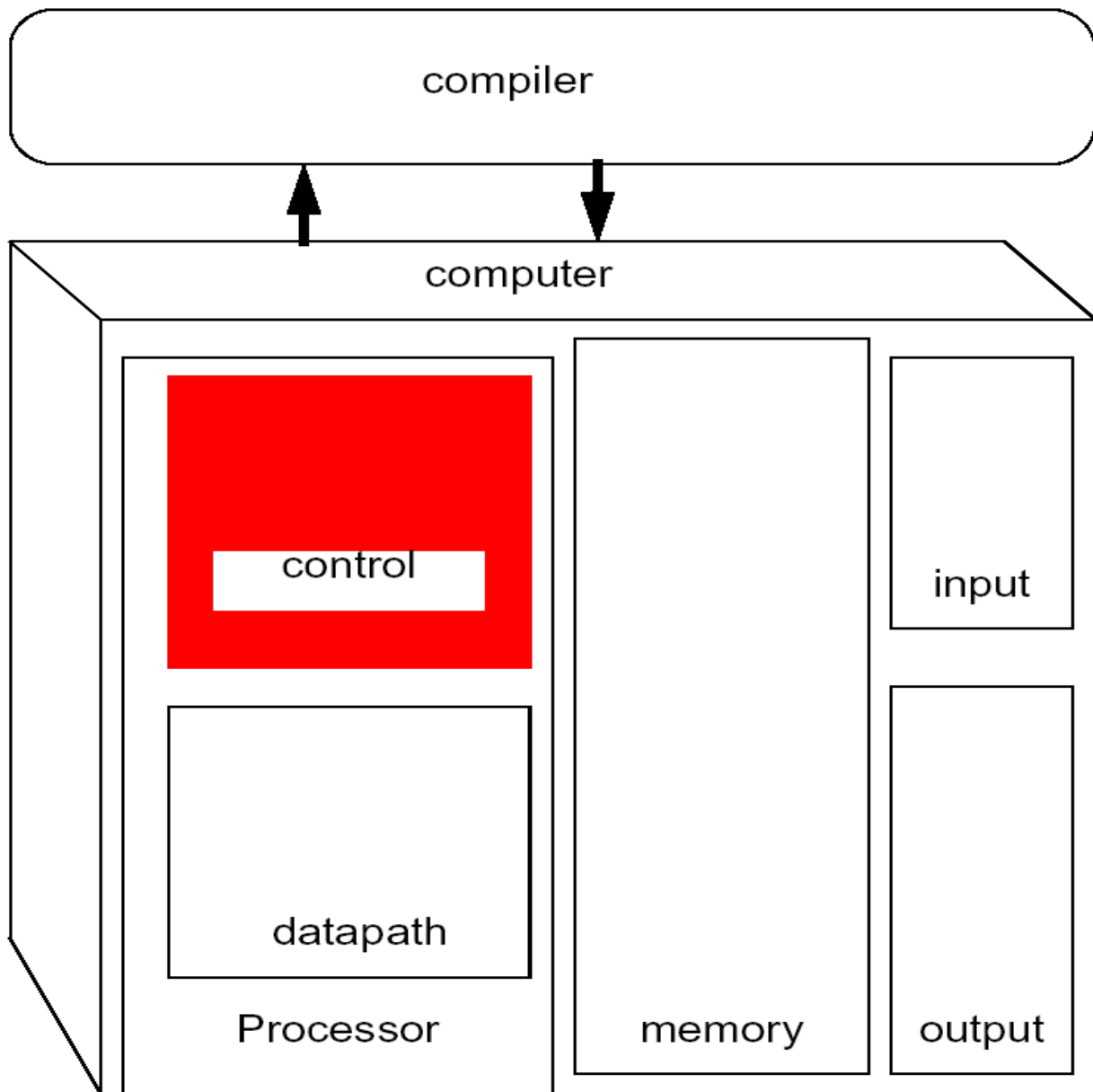
Designing Sequential Circuits

- How to make a sequential circuit for a given application?

Tools

- Finite state machine
- ROM-based Controllers

The Big Picture



Abstraction Hierarchy

Programming Language

Assembly Language

Machine Language

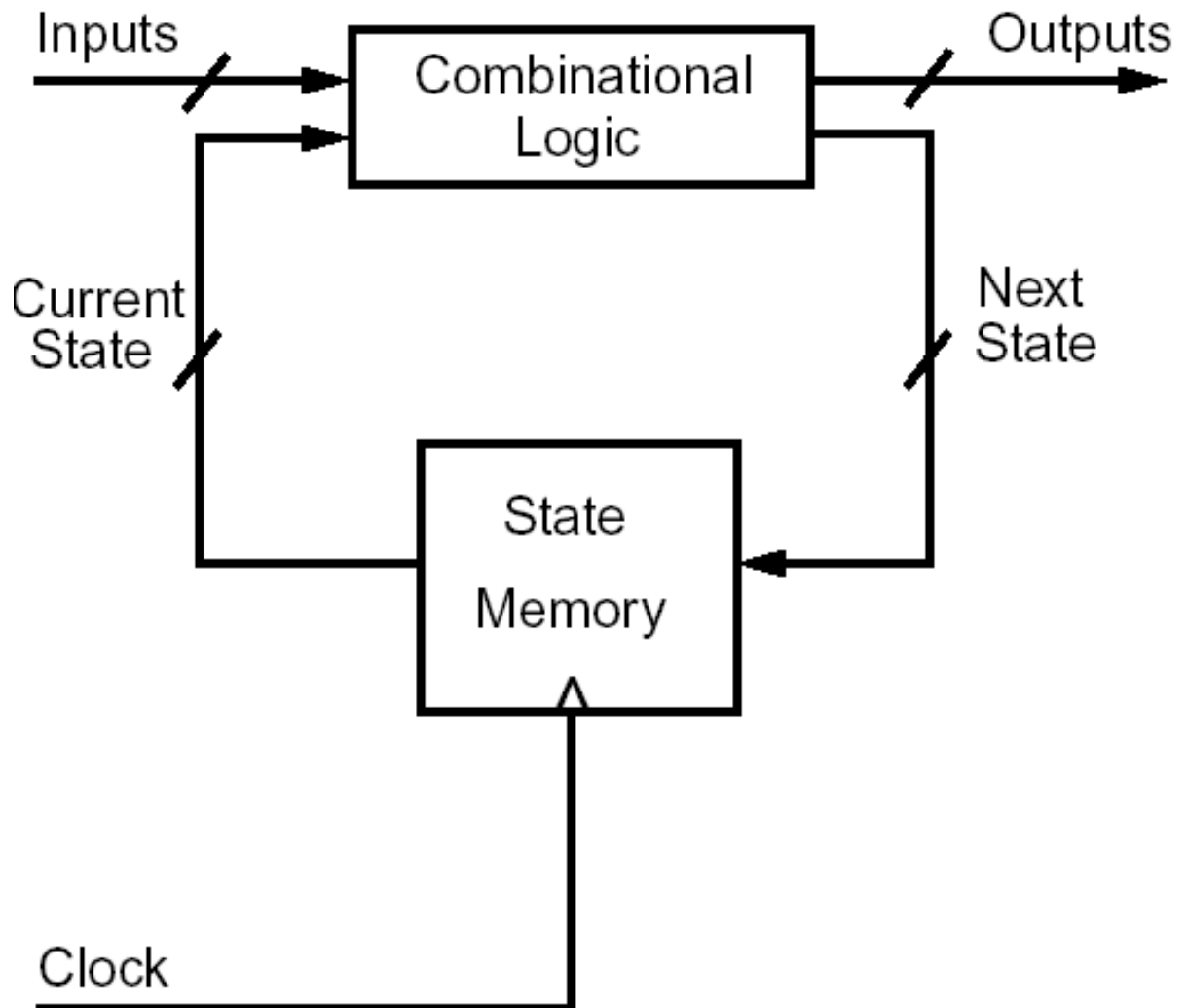
Sequential Circuit

Combinational Circuit

Binary Value

Voltage

Sequential Circuit

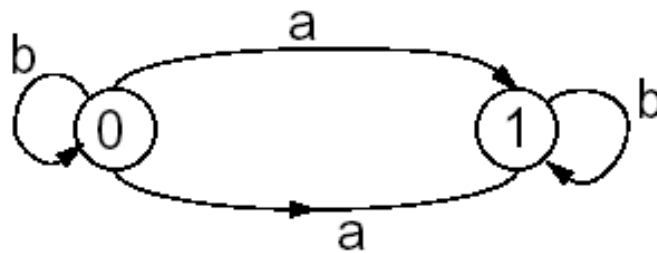


Finite State Machines

An Abstract Model of Sequential Circuits Specified by

- set of states (S)
- set of inputs (I)
- set of outputs (O)
- state-transition function ($S \times I \rightarrow S$)
- output function ($S \times I \rightarrow O$)

Usually specified with a picture



States: $\{0, 1\}$

Inputs: $\{a, b\}$

Building a Sequential Circuit

1. Draw the transition table, i.e., a table, which, given a state and some values for the inputs, produces a new state
2. From the transition table, deduce the flip-flop input functions to obtain the new state
3. Simplify the flip-flop input functions
4. Draw the output table, i.e, a table, which, given a state and some values for the inputs, produces an output value
5. Simplify the output table
6. Draw the circuit

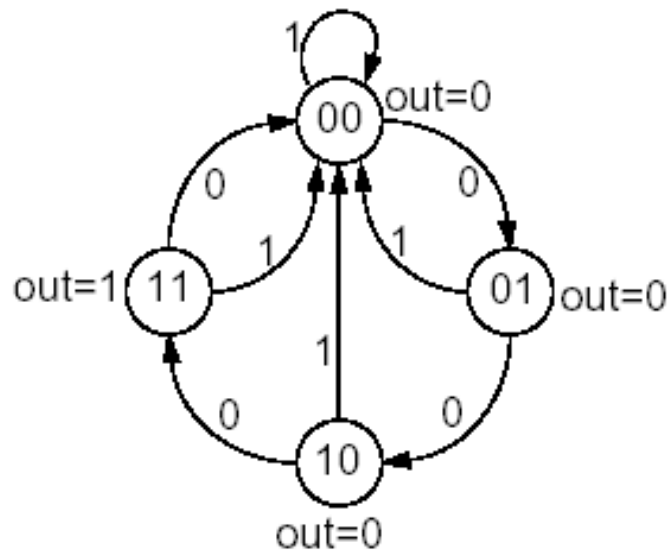
Four State Counter

Problem:

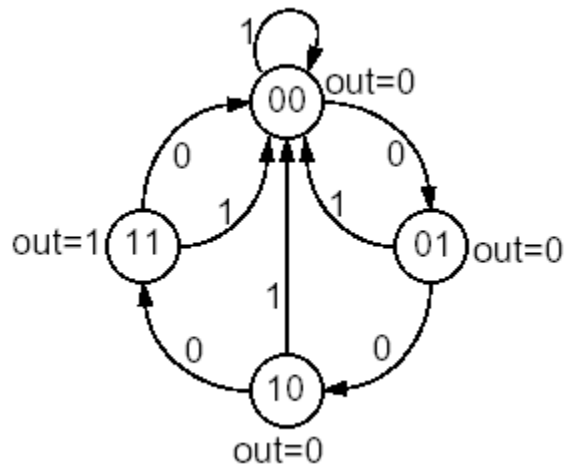
Make a device that outputs a 1 on every fourth clock-tick and 0 otherwise. Allow it to be reset at any time.

Step 1: Model the device as an FSM

- four states (2 bits)
- 1 output bit
- 1 input bit



State Table



Step 2: Make state table

| $S_1(t)$ | $S_0(t)$ | R | $S_1(t+1)$ | $S_0(t+1)$ |
|----------|----------|---|------------|------------|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

Flip-Flop Input Functions

Step 3: Calculate flip-flop input functions

JK Excitation Table

| Q(t) | Q(t+1) | J | K |
|------|--------|---|---|
| 0 | 0 | 0 | x |
| 0 | 1 | 1 | x |
| 1 | 0 | x | 1 |
| 1 | 1 | x | 0 |

| S ₁ | S ₀ | R | S ₁ (t+1) | S ₀ (t+1) | JS ₁ | KS ₁ | JS ₀ | KS ₀ |
|----------------|----------------|---|----------------------|----------------------|-----------------|-----------------|-----------------|-----------------|
| 0 | 0 | 0 | 0 | 1 | 0 | x | 1 | x |
| 0 | 0 | 1 | 0 | 0 | 0 | x | 0 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | x | x | 1 |
| 1 | 0 | 0 | 1 | 1 | x | 0 | 1 | x |
| 1 | 0 | 1 | 0 | 0 | x | 1 | 0 | x |
| 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 |
| 1 | 1 | 1 | 0 | 0 | x | 1 | x | 1 |

Simplify

Step 4: Simplify flip-flop input functions

| $S_1(t)$ | $S_0(t)$ | R | $S_1(t+1)$ | $S_0(t+1)$ | JS_1 | KS_1 | JS_0 | KS_0 |
|----------|----------|---|------------|------------|--------|--------|--------|--------|
| 0 | 0 | 0 | 0 | 1 | 0 | x | 1 | x |
| 0 | 0 | 1 | 0 | 0 | 0 | x | 0 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | x | x | 1 |
| 1 | 0 | 0 | 1 | 1 | x | 0 | 1 | x |
| 1 | 0 | 1 | 0 | 0 | x | 1 | 0 | x |
| 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 |
| 1 | 1 | 1 | 0 | 0 | x | 1 | x | 1 |

JS_0

| | S_0 | S_0 | S_0' | S_0' |
|----|--------|-------|--------|--------|
| R | x | x | 0 | 0 |
| R' | x | x | 1 | 1 |
| | S_1' | S_1 | S_1 | S_1' |

JS_1

| | S_0 | S_0 | S_0' | S_0' |
|----|--------|-------|--------|--------|
| R | 0 | x | x | 0 |
| R' | 1 | x | x | 0 |
| | S_1' | S_1 | S_1 | S_1' |

More Simplifying

Step 4: Simplify flip-flop input functions

| S_1 | S_0 | R | $S_1(t+1)$ | $S_0(t+1)$ | JS_1 | KS_1 | JS_0 | KS_0 |
|-------|-------|---|------------|------------|--------|--------|--------|--------|
| 0 | 0 | 0 | 0 | 1 | 0 | x | 1 | x |
| 0 | 0 | 1 | 0 | 0 | 0 | x | 0 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | x | x | 1 |
| 1 | 0 | 0 | 1 | 1 | x | 0 | 1 | x |
| 1 | 0 | 1 | 0 | 0 | x | 1 | 0 | x |
| 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 |
| 1 | 1 | 1 | 0 | 0 | x | 1 | x | 1 |

KS_0

| | S_0 | S_0 | S_0' | S_0' |
|----|--------|-------|--------|--------|
| R | 1 | 1 | x | x |
| R' | 1 | 1 | x | x |
| | S_1' | S_1 | S_1 | S_1' |

KS_1

| | S_0 | S_0 | S_0' | S_0' |
|----|--------|-------|--------|--------|
| R | x | 1 | 1 | x |
| R' | x | 1 | 0 | x |
| | S_1' | S_1 | S_1 | S_1' |

Output Function

Step 5: Determine the output function

| S_1 | S_0 | R | O |
|-------|-------|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

We can do this one by inspection!

The Circuit

Step 6: Draw the Circuit

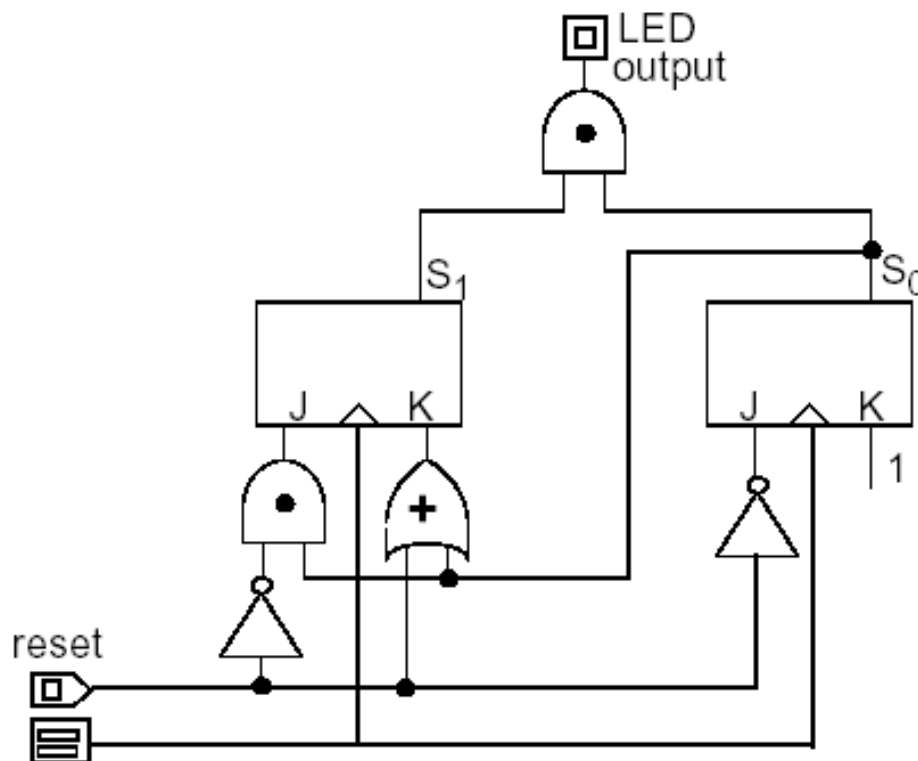
$$\text{Out} = S_0 S_1$$

$$J S_1 = R' S_0$$

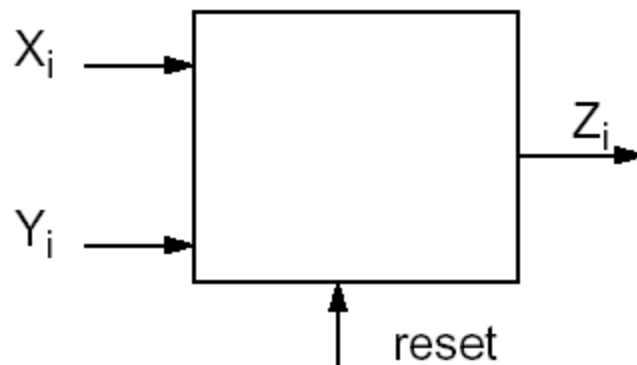
$$K S_1 = S_0 + R$$

$$J S_0 = R'$$

$$K S_0 = 1$$



A Serial Adder



$$Z_0 = (X_0 + Y_0) \bmod 2$$

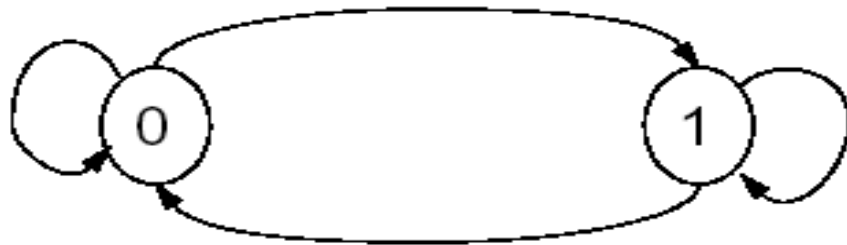
$$Z_1 = X_1 + Y_1 + (X_0 + Y_0) \text{ div } 2$$

etc.

When $r=1$, the carry is forgotten and we go back to X_0+Y_0

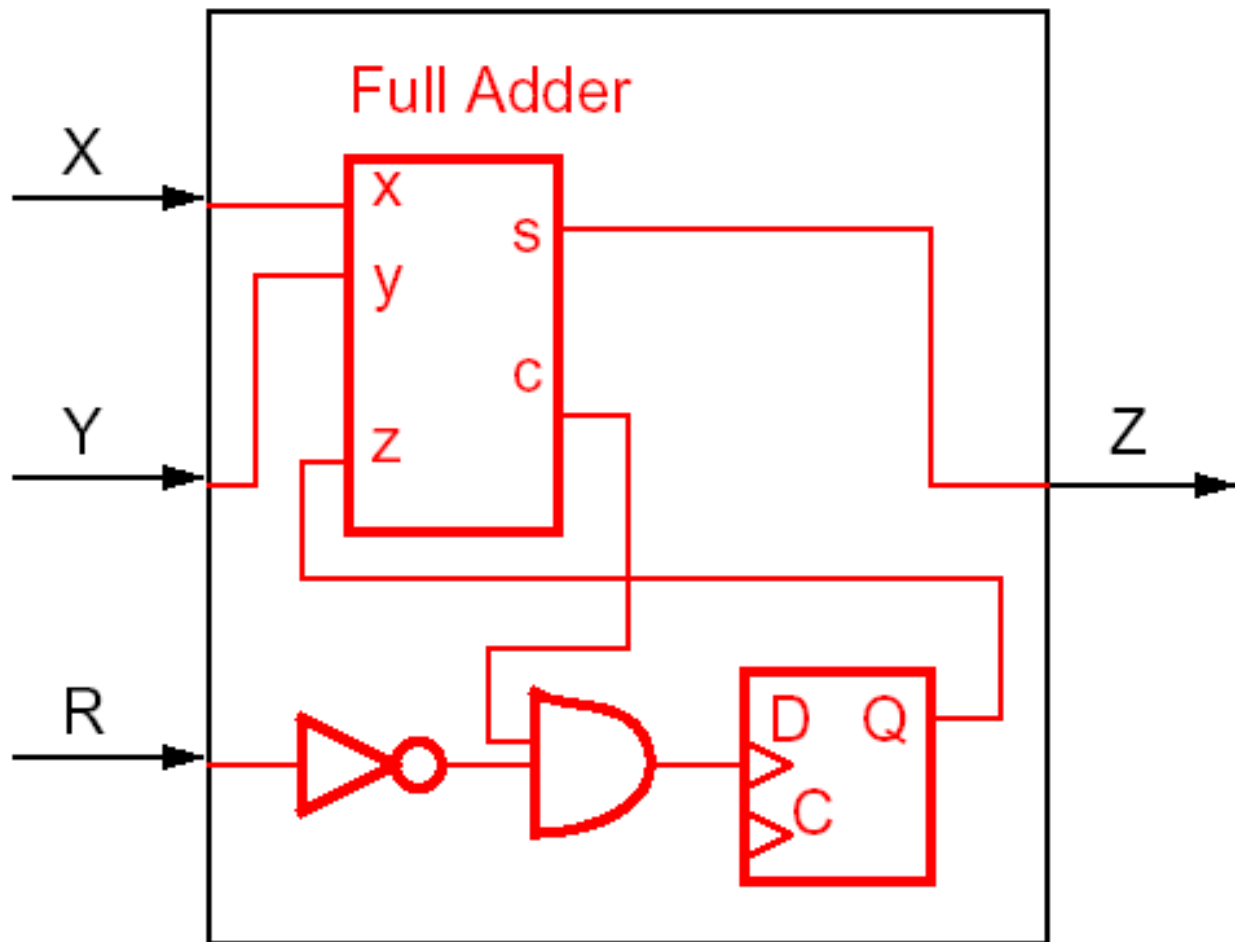
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|----|
| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| x | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| y | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| r | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| z | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

FSM for Serial Adder



| R | X | Y | S(t) | S(t+1) | Z |
|---|---|---|------|--------|---|
| 1 | x | x | x | 0 | x |
| 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 1 | 0 | 1 |
| | 0 | 1 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 1 | 0 |
| | 1 | 0 | 0 | 0 | 1 |
| | 1 | 0 | 1 | 1 | 0 |
| | 1 | 1 | 0 | 1 | 0 |
| | 1 | 1 | 1 | 1 | 1 |

Circuit for Serial Adder



Use a D flip-flop and a full adder

Big FSM

What if we have a finite-state machine with

- 10 possible inputs
- 100 possible states
- 20 possible outputs

How many

- input bits?
- flip-flops?
- output bits?

Who wants to do the Karnaugh maps?

Big FSM

What if we have a finite-state machine with

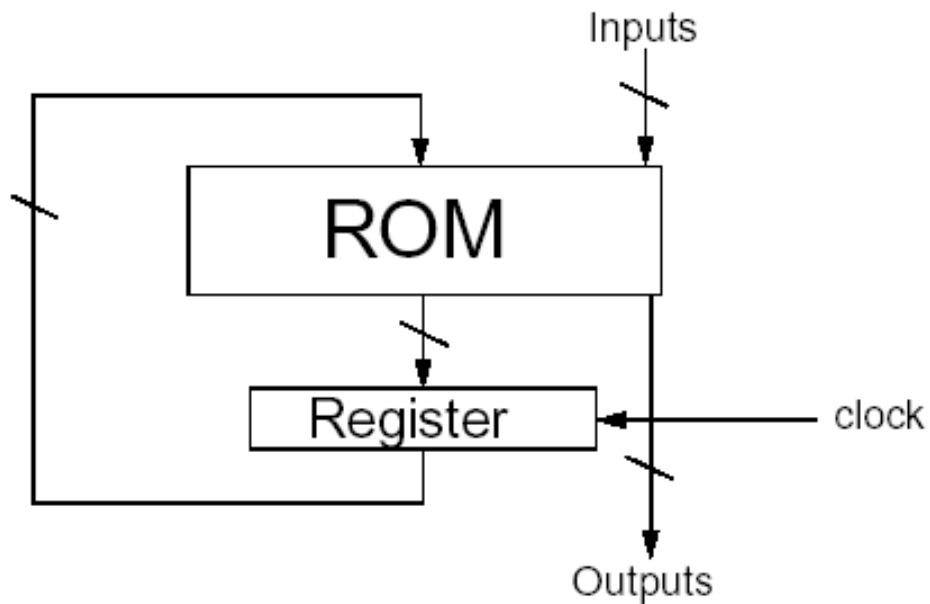
- 10 possible inputs
- 100 possible states
- 20 possible outputs

How many

- input bits? 4
- flip-flops? 7
- output bits? 5

Who wants to do the Karnaugh maps?

ROM-based Controller



To address the ROM, give

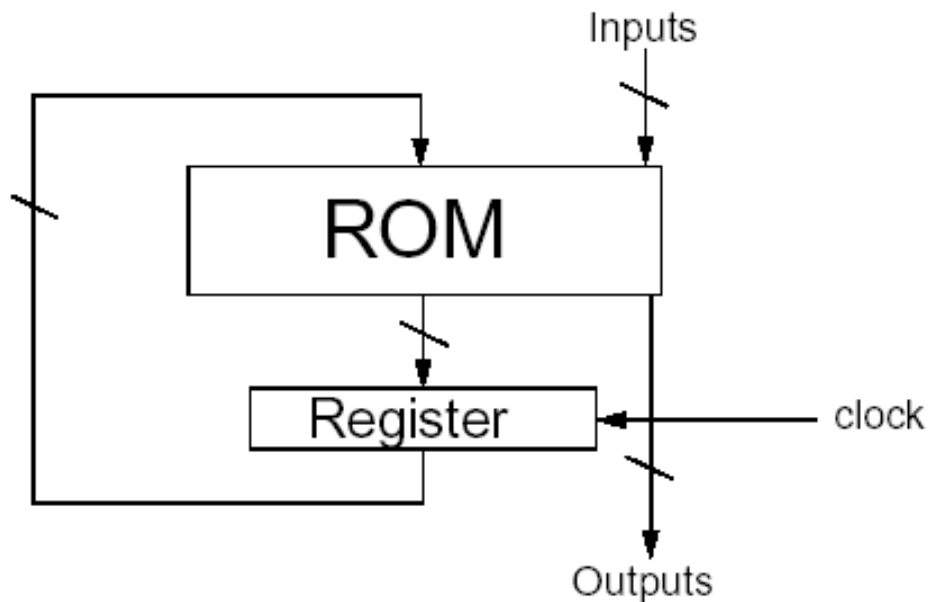
- old state
- current inputs

Result is

- new state
- new outputs

What size of ROM do we need?

ROM-based Controller



To address the ROM, give

- old state
- current inputs

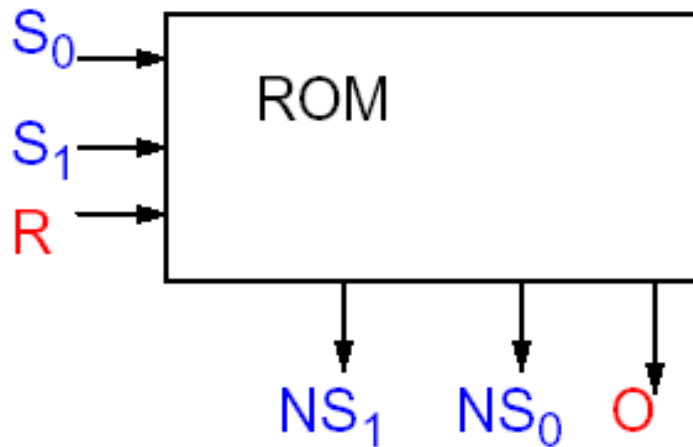
Result is

- new state
- new outputs

What size of ROM do we need?

$$2^{(4+7)} = 2^{11}$$

ROM-based Controller



Contents of the ROM

| S_1 | S_0 | R_0 | N_1 | N_0 | O |
|-------|-------|-------|-------|-------|-----|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |