Slide Set 8

for ENEL 353 Fall 2019

Steve Norman, PhD, PEng

Electrical & Computer Engineering
Schulich School of Engineering
University of Calgary

Fall Term, 2019

Contents

Outline of Slide Set 8

Synchronous sequential circuits

A *synchronous* sequential circuit is a sequential logic system that

- has one or more bits of **state**; and
- has its state updates controlled by a **clock signal**, so that the state updates are **synchronized** by the clock.

Making digital systems synchronous is a very powerful design technique—the vast majority of digital circuits, including just about all practical computer processors, are synchronous sequential systems.

Synchronous sequential circuit composition rules

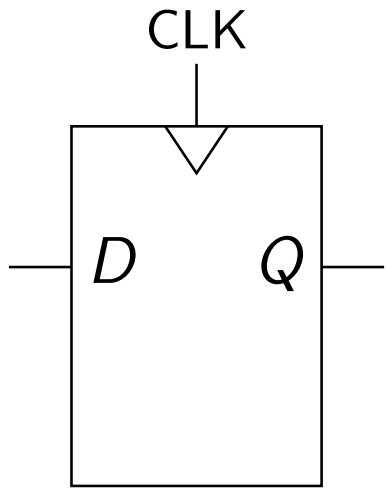
If a circuit satisfies these four rules, it is guaranteed to be a synchronous sequential circuit . . .

1. Every element in the circuit either is a register or is combinational.
2. At least one element is a register.
3. All registers receive the same clock signal.
4. Every cyclic path in the circuit passes through at least one register.

Remark: The above conditions are **sufficient** but **not necessary**. There are designs that break some of these rules but still result in synchronous sequential circuits.

Composition rules: Simplest possible example

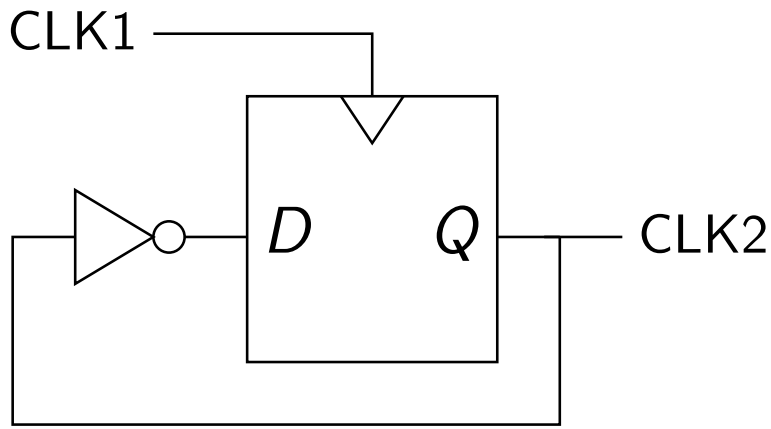
Does a single D flip-flop, by itself, satisfy the synchronous sequential circuit composition rules?



1. Every element in the circuit either is a register or is combinational.
2. At least one element is a register.
3. All registers receive the same clock signal.
4. Every cyclic path in the circuit passes through at least one register.

Composition rules: Another simple example

Does the clock divider circuit satisfy the synchronous sequential circuit composition rules?

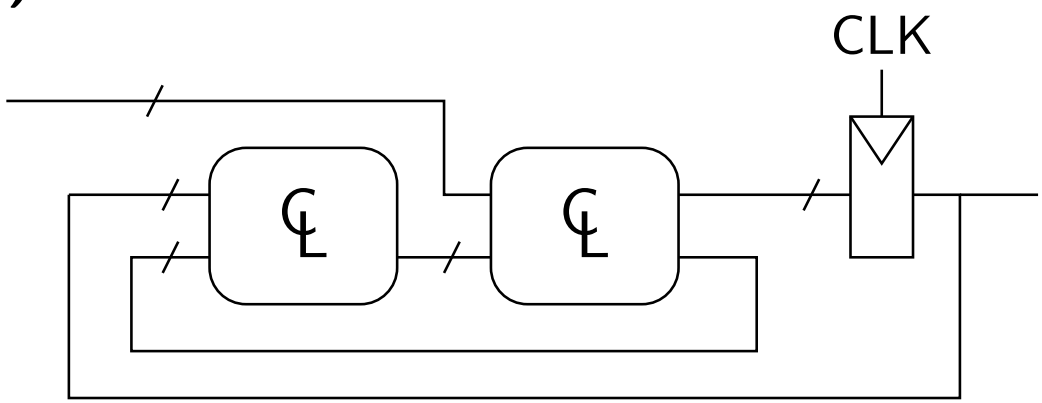


1. Every element in the circuit either is a register or is combinational.
2. At least one element is a register.
3. All registers receive the same clock signal.
4. Every cyclic path in the circuit passes through at least one register.

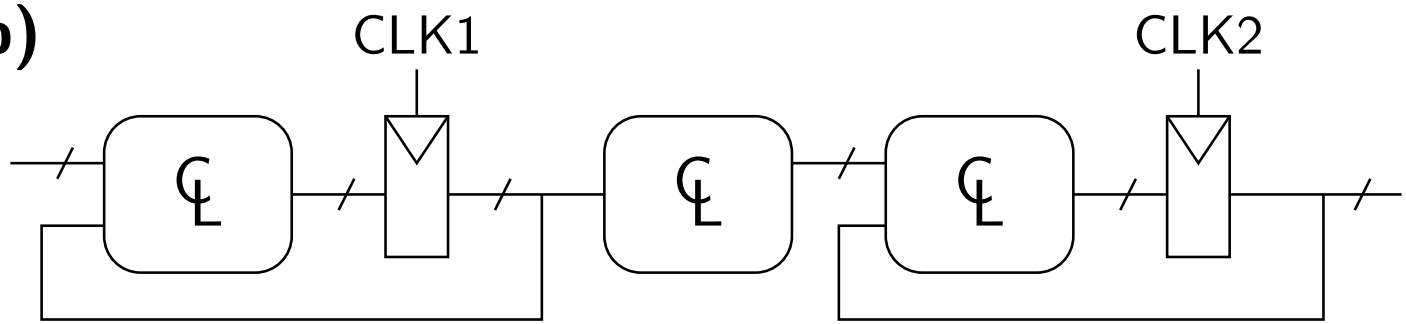
More general examples, part 1

Which of these circuits satisfies all four synchronous sequential circuit composition rules?

(a)



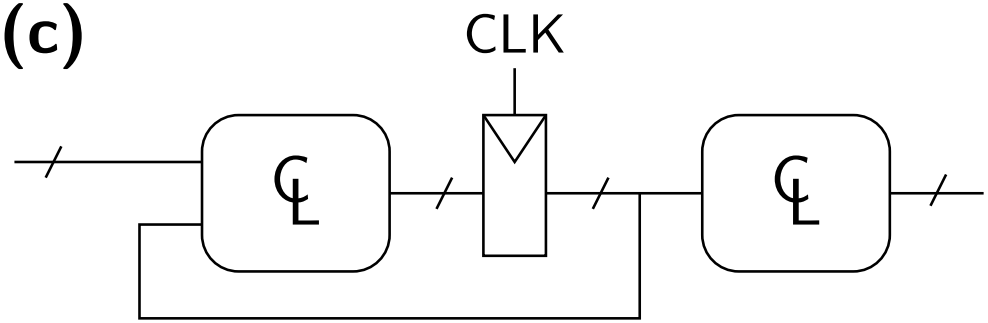
(b)



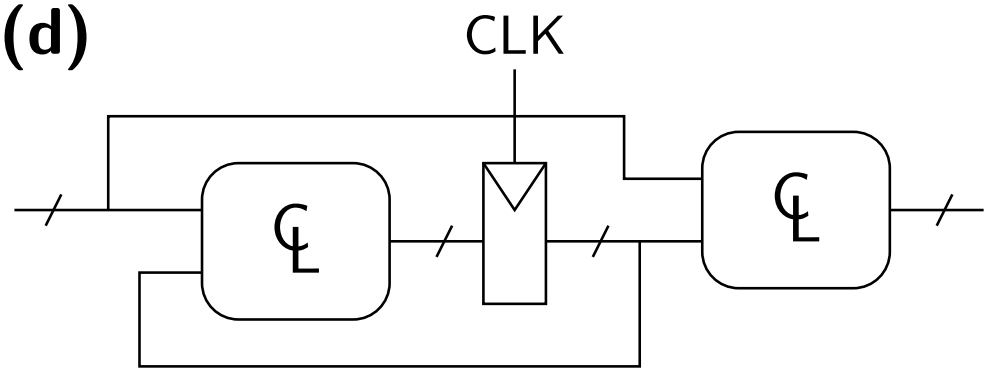
More general examples, part 2

Which of these circuits satisfies all four synchronous sequential circuit composition rules?

(c)



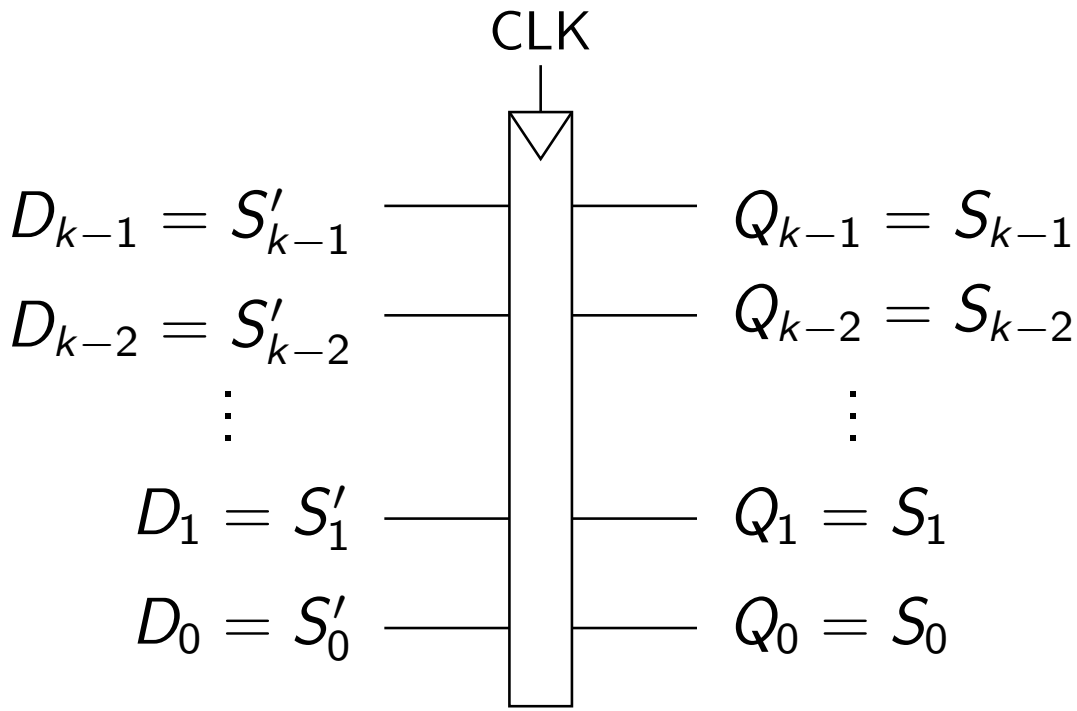
(d)



Outline of Slide Set 8

Finite state machines

A k-bit *finite state machine* **(FSM)** is a synchronous sequential circuit that has a k-bit register at its centre ...



How many different states can a k-bit FSM have?

Important words on notation for FSM algebra

Harris and Harris use S_i' to mean "next state i", the value DFF i within the register of an FSM will get just after a rising clock edge.

That is unfortunate, because A' means NOT(A) in a **large** body of literature on digital systems.

Many texts use something like Q_i^* for the value DFF i within the register of an FSM will get just after an active clock edge.

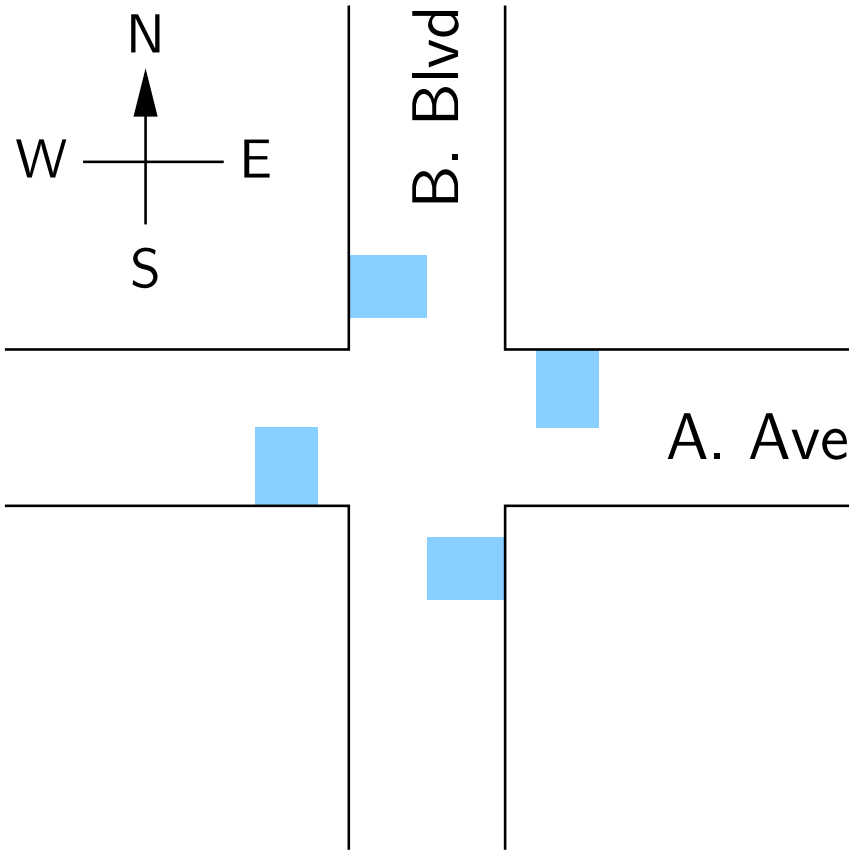
For ENEL 353 in Fall 2019, it's **crucial** to know that S_i' means "the value state bit i will get just after a rising clock edge" and **does not** mean "the complement of S_i".

Moore machines and Mealy machines: Two kinds of FSM

Let's draw diagrams and make notes to distinguish these two types of finite state machine.

Outline of Slide Set 8

Traffic light problem: system inputs T_A and T_B



There are traffic sensors built in to the road.

Each one indicates whether a vehicle has been near the sensor within the last several seconds.

T_A = (eastbound traffic on A) OR (westbound traffic on A)
T_B = (northbound traffic on B) OR (southbound traffic on B)

Traffic light problem: system outputs $L_{A1:0}$ and $L_{B1:0}$

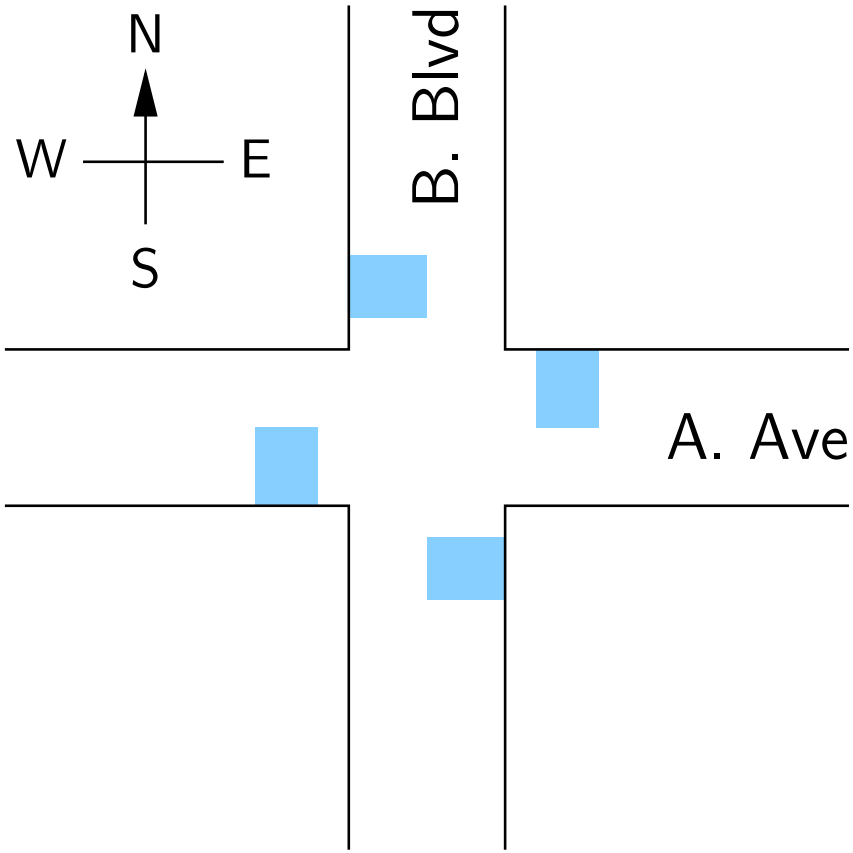
The table shows how commands will be sent from the traffic light controller to each of the four sets of red-yellow-green lights. Each set of lights has two input wires.

Output	Encoding $L_{1:0}$
green	00
yellow	01
red	10

How many output bits in total will the controller need to have?

Let's make a sketch showing all the inputs and outputs of the controller.

Traffic light problem: system specification



Let's write down a specification of the traffic light behaviour.

We'll start by specifying what should happen on a reset.

(The "spec" will match what is in the textbook, pages 124–125.)

Attention: We are circuit designers, not traffic engineers, so we're going to implement the spec, not critique it!

The clock for the traffic light controller

In digital design we usually think of clocks with kHz or MHz or GHz speed.

But in this application the potential state updates occur once every 5 seconds.

So here $T_C = 5\,\text{s}$, and $f_C = 1/5\,\text{s} = 0.2\,\text{Hz}$.

The state transition diagram

A key step in designing an FSM is making a *state transition diagram*, which shows graphically

- all the possible states of the system;
- all the possible **transitions** from one state to another, with information about how the inputs affect these transitions;
- descriptions of how outputs are related to the state (in Moore FSMs) or to the state and current inputs (in Mealy FSMs).

Let's make a state transition diagram for our traffic light controller (which will be a Moore FSM).

Then, let's make a few general notes about conventions for state transition diagrams for Moore FSMs.

From state transition diagram to state transition table

To continue moving from specification to circuit design, a table listing the next states for all possible combinations of current state and input signals is helpful.

This kind of table is called a *state transition table*.

Use of X for don't-care inputs can help keep the size of the table manageable.

Let's write out a state transition table for the traffic light controller.

If we didn't use X for don't-care inputs, how many rows would the state transition table have?

State encoding

In our FSM circuit, the state will be stored in DFFs in a register, so to make further progress, we will need to decide on **bit patterns** to represent each of the states that we have named S0, S1, S2, and S3.

Let's write out the most obvious choice of state encoding for the traffic light controller example.

Important remark about notation:

- S0, S1, S2, S3 (with digits the same size as letters) are **names of states**.
- S_0 and S_1 (with digits appearing in subscripts) identify **bits within the state register**.

A truth table for next state logic

By substituting our state encodings into the state transition table, we can get a version of the state transition table that completely specifies the next state as a **combinational logic function** of the current state and input variables.

Let's make that table for the traffic light controller.

Let's use the table and some K-maps to find equations for S_1' *and* S_0'. (Harris and Harris go straight from the table to equations, but your instructor thinks K-maps help to explain where the equations come from.)

Output logic for the traffic light controller

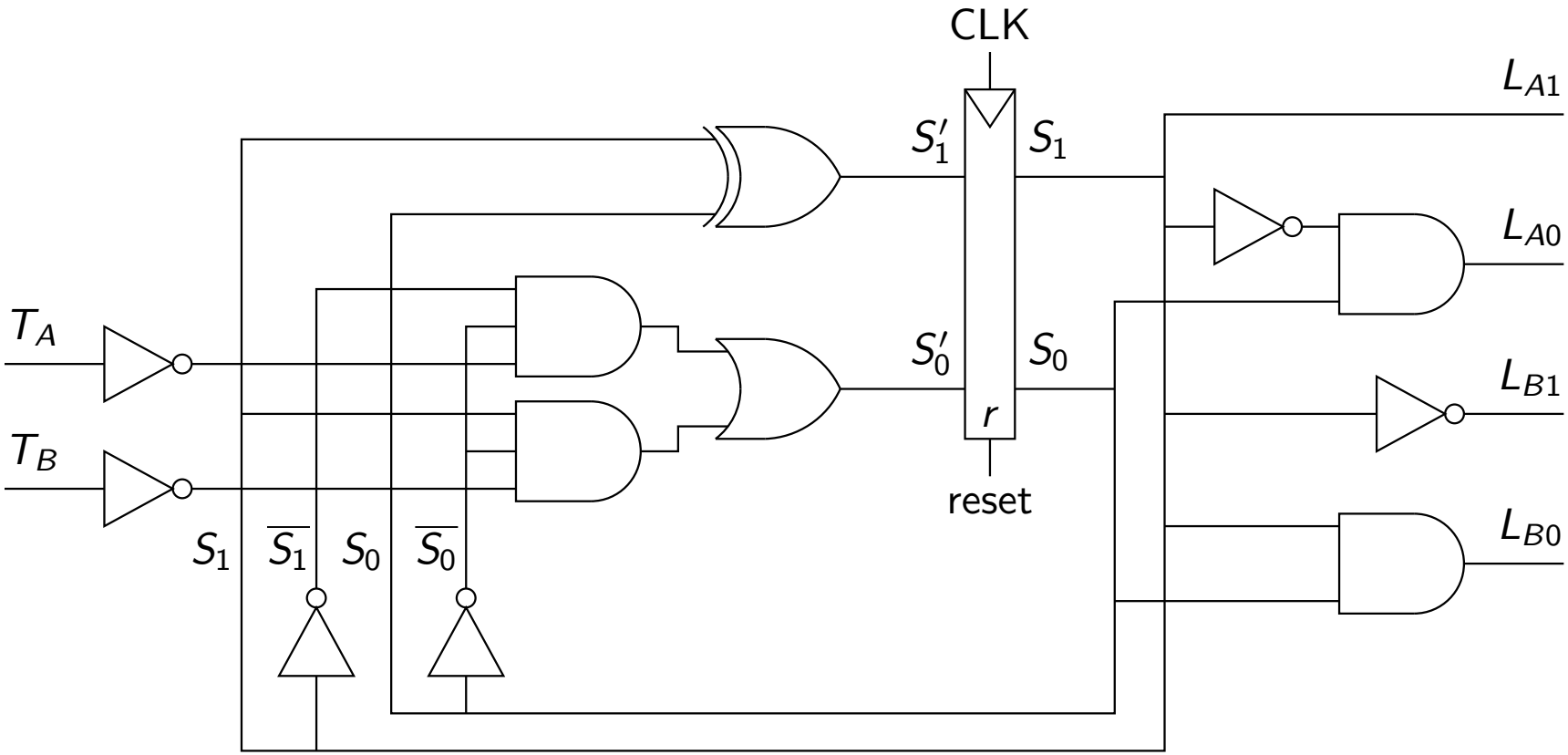
To design output logic, we need to combine the information below into a truth table.

Output	$L_{1:0}$
green	00
yellow	01
red	10

State, $S_{1:0}$	Desired lights
S0, 00	green on A, red on B
S1, 01	yellow on A, red on B
S2, 10	red on A, green on B
S3, 11	red on A, yellow on B

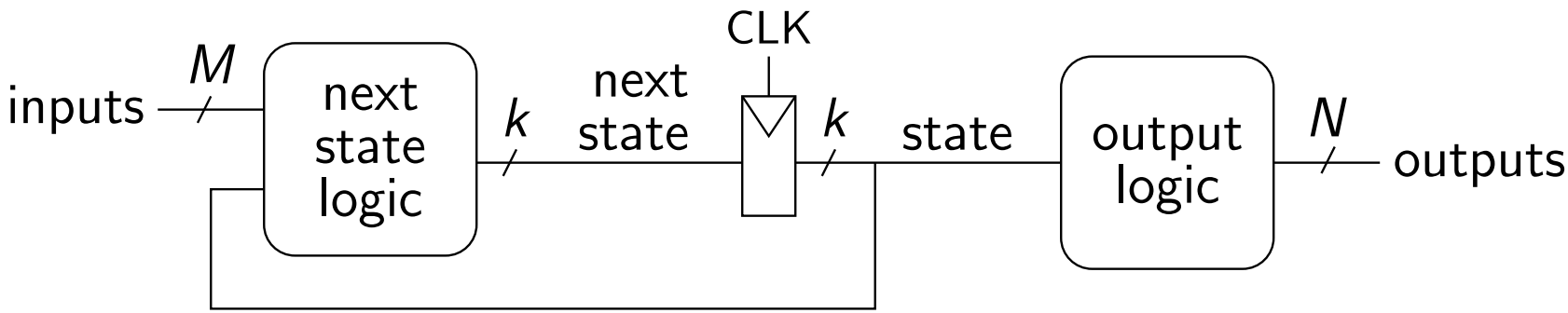
Let's make the truth table, then use it to get equations for L_{A1}, L_{A0}, L_{B1}, *and* L_{B0}.

From equations to a circuit ...



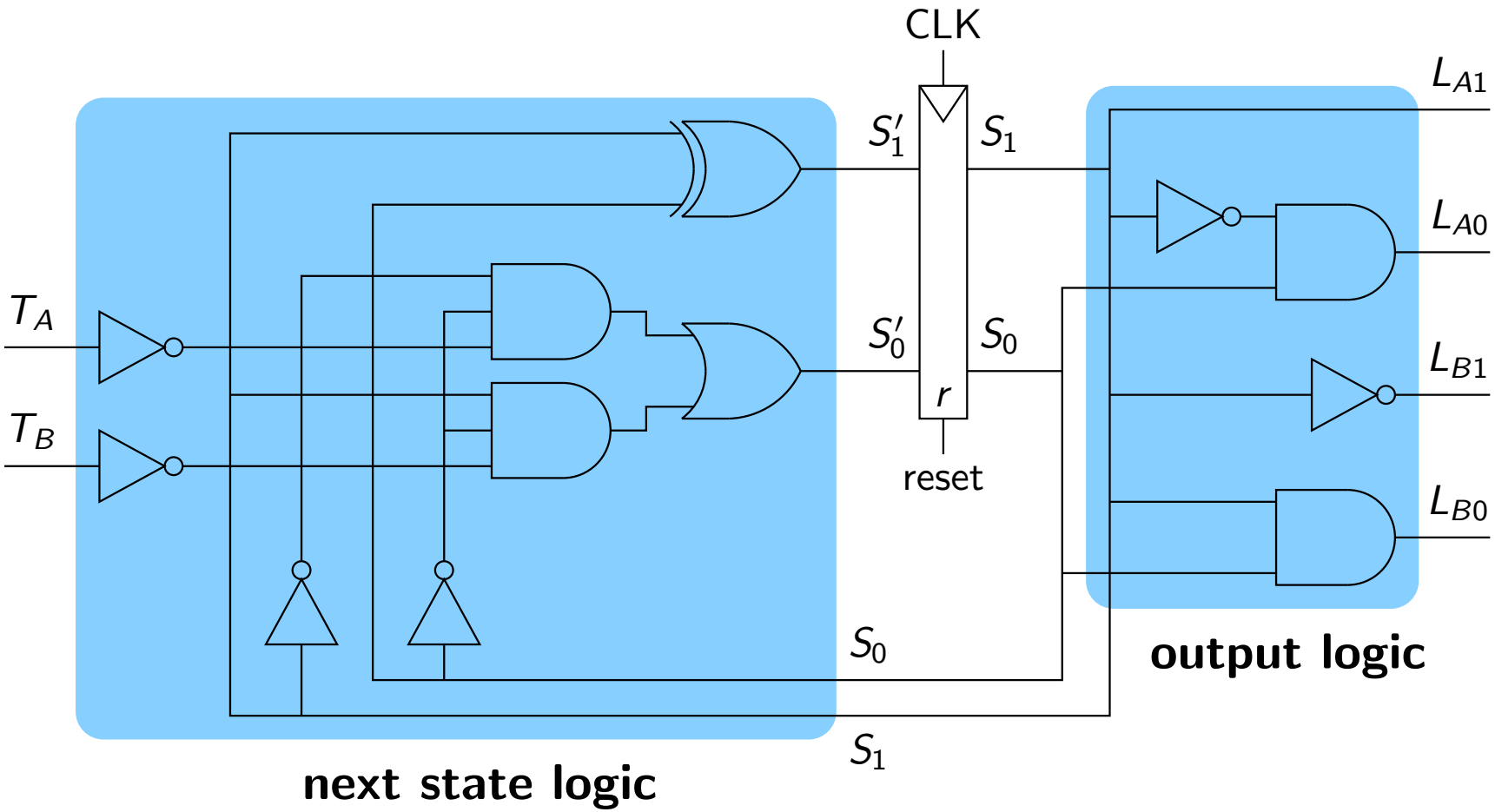
The schematics in Figure 3.26 in Harris & Harris are a more compact way of representing this particular implementation of the traffic light controller.

Review: General structure of a Moore FSM



It's not hard to see how the traffic light controller fits into this framework . . .

Moore FSM structure of traffic light controller



Timing diagram for the traffic light controller FSM

Figure 3.27 on page 128 of Harris & Harris is an excellent timing diagram showing behaviour of the FSM over ten example clock cycles.

There's too much detail in the diagram to make a legible slide from it.

Please study the diagram and the related discussion on pages 127 and 129 carefully! It really helps in explaining typical FSM operation as time progresses.

Outline of Slide Set 8

Review of steps in design of a Moore FSM

This is what we went through for the traffic light controller example. The same sequence of steps is more or less applicable to all FSM design problems.

The steps are listed on the next two slides.

Moore FSM steps, slide 1 of 2

- Start with a **word description** of what the FSM will do. If necessary, refine the word description so that it is **complete** and **unambiguous**.
- Use the word description to create a **state transition diagram**.
- Use the state transition diagram to make a **state transition table**.
- Decide on a **state encoding**: Each state must be represented by a unique sequence of 1's and 0's.

Moore FSM steps, slide 2 of 2

- Update the state transition table to reflect the state encoding. The result is a **truth table** for the **next state logic**. Use K-maps or some other method to find equations for the next state logic.
- Make an **output table** to list what the output bits are for each the state encodings. Use K-maps or some other method to find equations for the **output logic**.
- Design combinational logic to implement the next state and output equations. If you're taking a course such as ENEL 353, make a schematic. (In the "real world", it's more likely that you would describe the next state and output logic in a language such as VHDL.)

Outline of Slide Set 8

Next FSM Example: A "divide-by-3 counter"

This example is taken from Section 3.4.2 of Harris & Harris.

We'll use the example

- to review the steps in designing an FSM, starting from a word description of system behaviour;
- to illustrate how the choice of state encoding may influence the relative complexity of next state and output logic.

Word description of the divide-by-3 counter: The only inputs are clock and reset signals. The output should be 1 during every third cycle of the input clock, and 0 during other cycles of the input clock.

Divide-by-3 counter: Waveforms and a state transition diagram

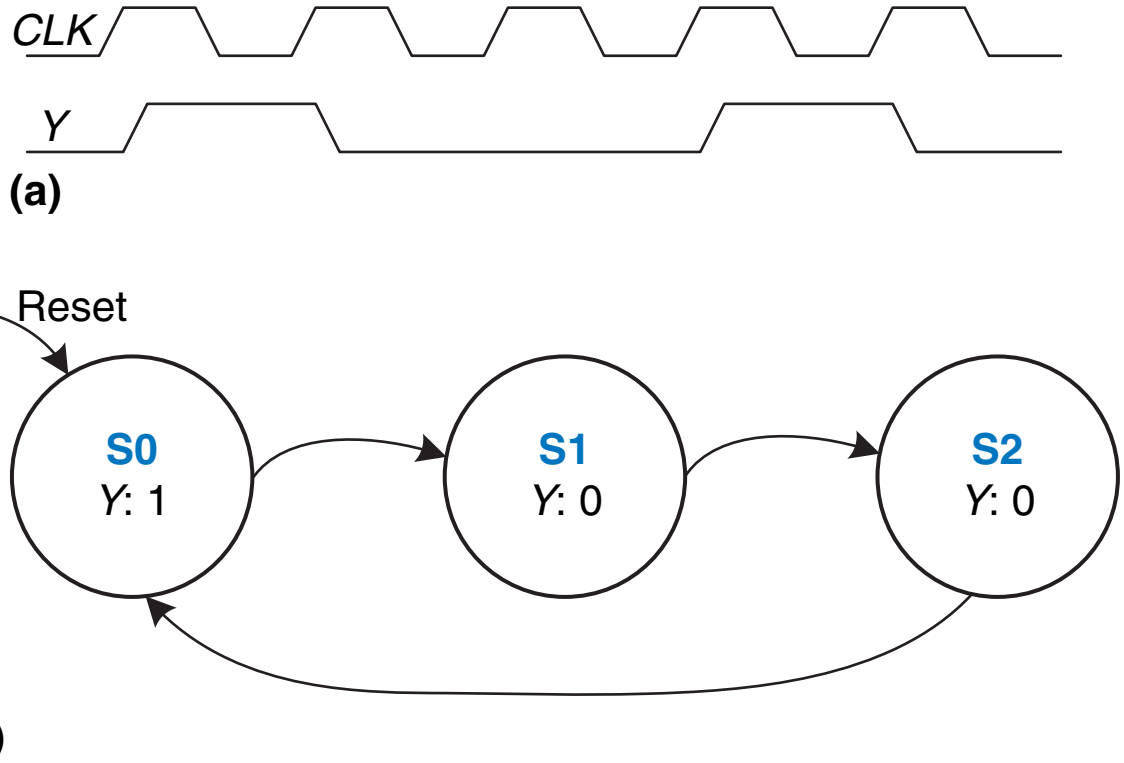


Image is taken from Figure 3.28 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.

"A *divide-by-N* counter has one output and no inputs."

The title of this slide is a quote from page 129 of Harris and Harris.

Does it make sense to say there are **no inputs? Every** FSM needs a **clock** input. And the state transition diagram for the divide-by-3 counter also shows a **reset** input.

Important: In discussion of FSMs, it's traditional to count the number of inputs as the number of system input signals used by **the next state logic**. For whatever reason, important input signals—such as CLK and reset—that go straight to the state register do not count in the number of FSM inputs.

Divide-by-3 counter with binary state encoding

Let's finish the design of the counter using unsigned binary encoding for the states S0, S1, and S2.

Unlike what is done in the textbook, we'll put the unused state 11 into our truth tables with the goal of using don't-care outputs to simplify the next-state and output equations.

Divide-by-3 counter with *one-hot* encoding

One-hot state encoding requires one flip-flop for each state. In any state, a single state bit is TRUE and all the others are FALSE.

One-hot state encodings for a system with **three** states:
001, 010, 100.

One-hot state encodings for a system with **four** states:
0001, 0010, 0100, 1000.

Let's complete the divide-by-3 FSM design using one-hot state encoding, then make a few remarks.

We'll assume that we can build a state register out of resettable DFFs and/or settable DFFs, whatever is needed to make the reset logic work.

Outline of Slide Set 8

Sequence detection problems

Most textbooks on digital design present several FSM design problems that are worded something like this example: "Design an FSM that will have an output of 1 only when the input is 0, but was 1 for the previous three clock cycles."

Example 3.7 from Harris & Harris: "Alyssa P. Hacker owns a pet robotic snail with an FSM brain. The snail crawls from left to right along a paper tape containing a sequence of 1's and 0's. On each clock cycle, the snail crawls to the next bit. The snail smiles when the last two bits it has crawled over are, from left to right, 01. Design the FSM [. . .]"

Problems of this kind are great practice for students, because solving them requires careful thought about states and state transitions.

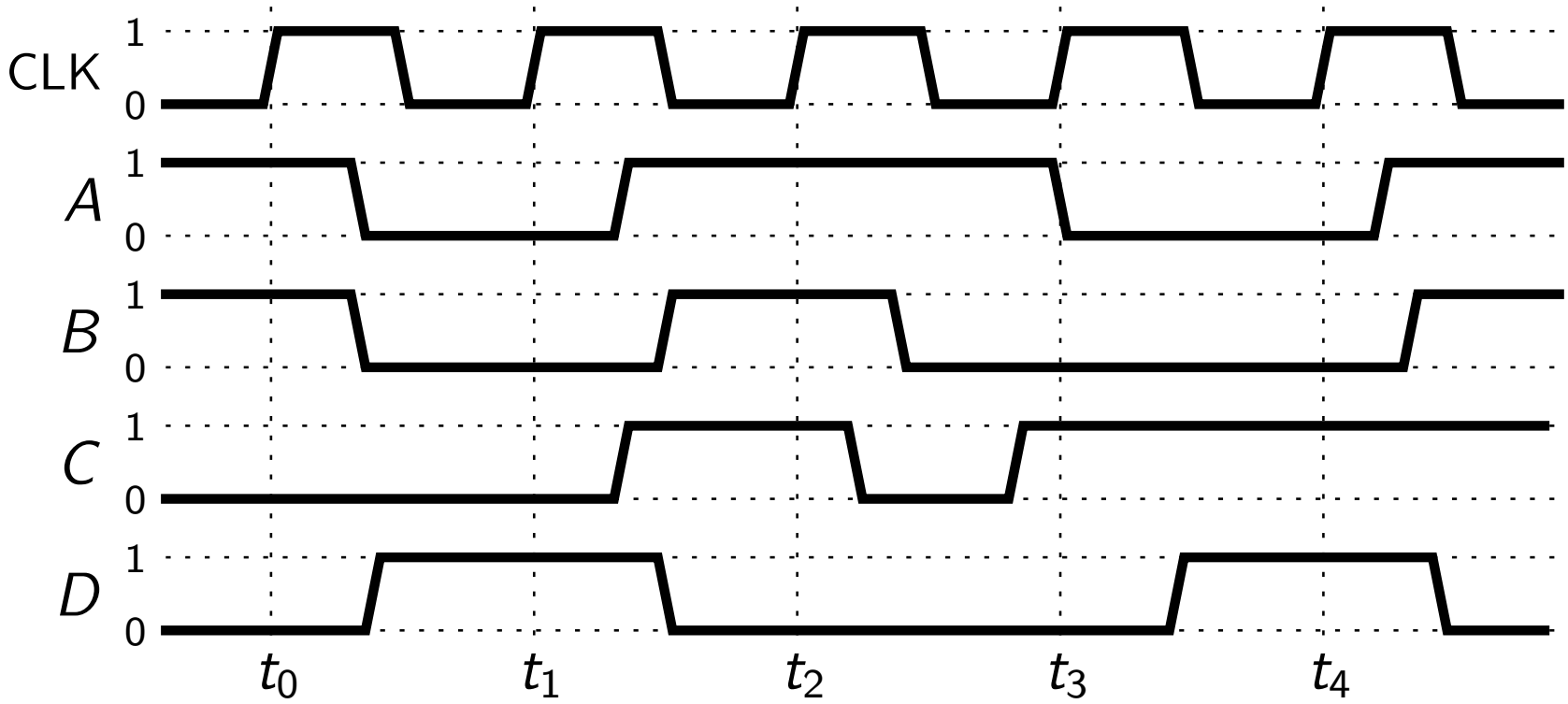
Useful assumptions for sequence detection problems

First, let's assume that **the clock period is long** compared to switching times for the available circuit elements. That way, we can ignore timing problems caused by excessive delays in circuit elements.

Second, let's assume that the input will be "reasonable", in the following senses:

- The input makes **at most one** $0 \rightarrow 1$ or $1 \rightarrow 0$ transition within any single clock cycle.
- The input **never** makes a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition really near in time to a rising clock edge. (This eliminates uncertainty about DFF D input values when those values are being copied to DFF Q outputs.)

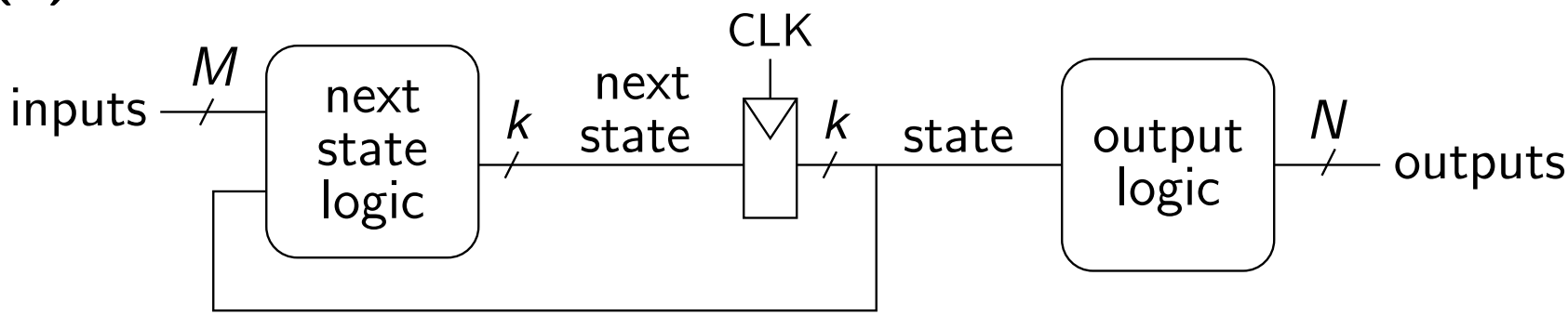
Which of A, B, C and D are "reasonable" inputs for sequence detection?



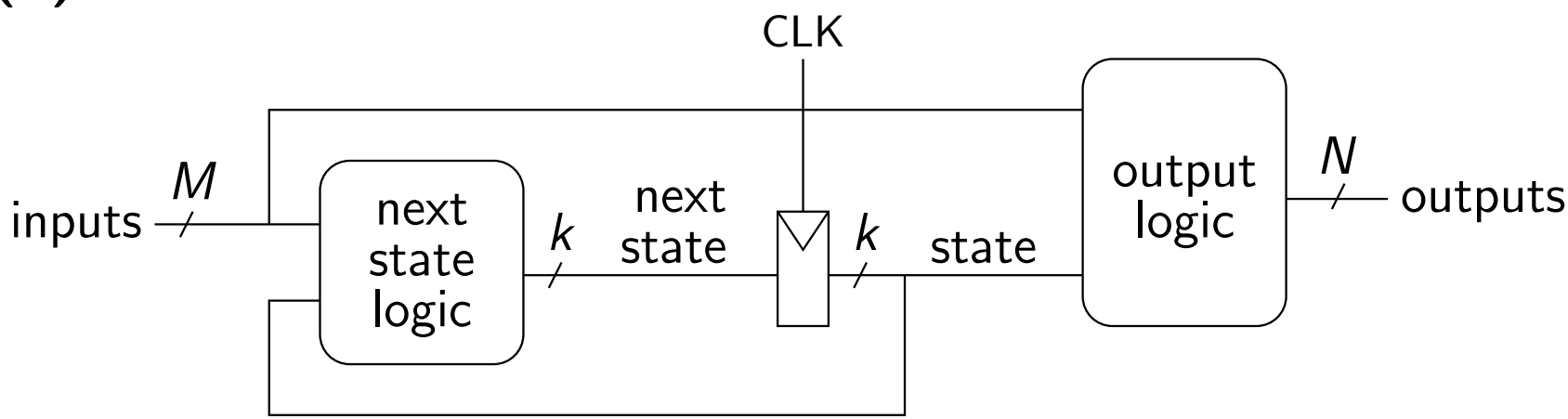
Review: Moore and Mealy FSM structures

Which is which? How can you tell?

(a)



(b)



A sequence detection problem

Problem statement: Design an FSM to detect the following sequence of bits on the FSM input: 1 for three clock cycles followed by a 0. The output should be 1 when the sequence has been detected and 0 at other times.

We'll consider both Moore and Mealy FSM designs.

For each of the Moore and Mealy designs, what does the problem statement mean, in terms of the current input value and the values of the input at recent clock edges?

Let's make a timing diagram to be really clear about how the Moore and Mealy outputs will react to a typical input signal.

Sequence detection example: Moore FSM

Let's describe the states of the system.

Let's make a state transition diagram.

In order to give more time to the Mealy FSM design problem, we won't complete the Moore FSM design work.

Sequence detection example: Mealy FSM

Let's make a list of states appropriate for a Mealy FSM solution to the problem.

Let's make a state transition diagram.

Let's make some general notes about how state transition diagrams for Mealy FSMs differ from state transition diagrams for Moore FSMs.

Sequence detection example: Mealy FSM, continued

Let's choose unsigned binary encoding of the states.

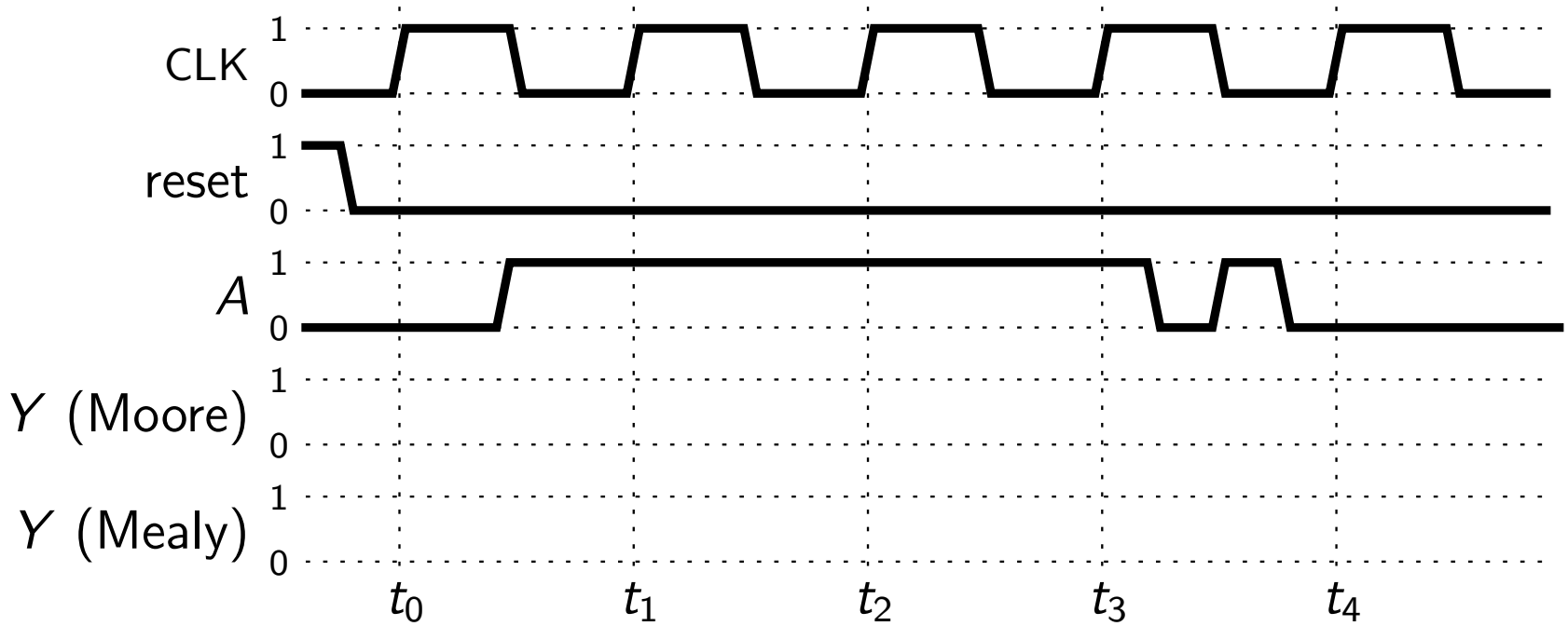
Let's make a combined state transition and output table. (We'll go straight to the version that is based on our chosen state encoding, because it's pretty easy to do without a version that lists the states symbolically.)

Let's find next-state and output equations.

Let's draw a schematic.

Sequence detection FSMs with an "unreasonable" input signal

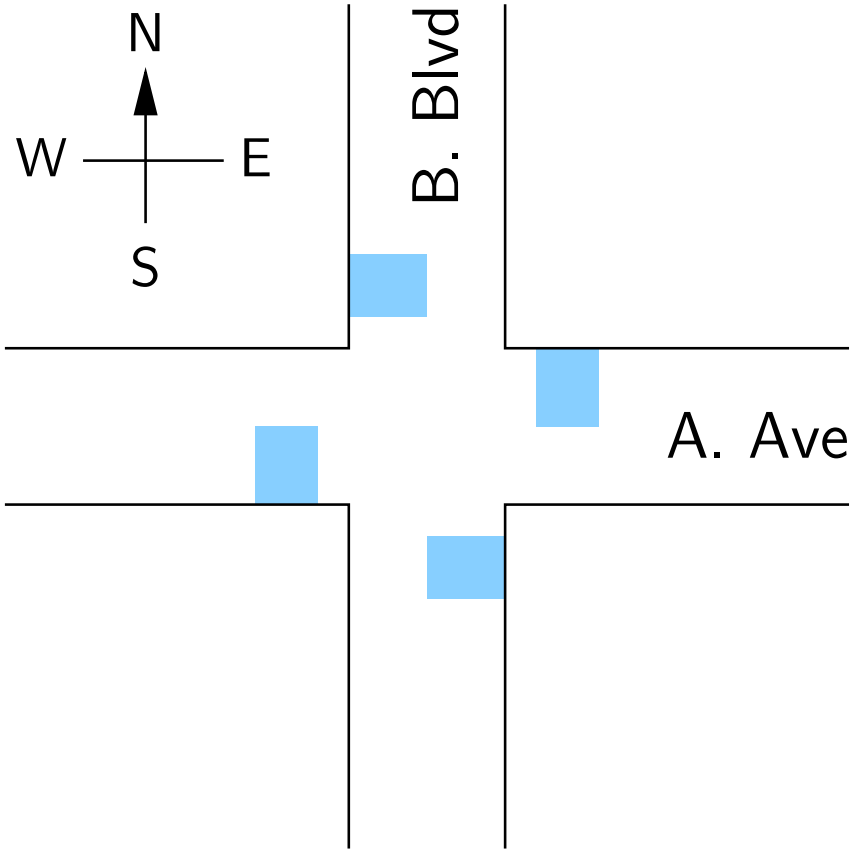
Let's determine what the Moore and Mealy FSMs will do if the input A is as shown . . .



Let's make a few remarks.

Outline of Slide Set 8

A revised traffic light problem



In **regular mode**, the system behaves like the system that's already been designed.

In **parade mode**, the system advances through the regular sequence until it gets to red for A and green for B, then **stays in that state**.

Let's make a block diagram showing the inputs and outputs of the new, more complex traffic light controller.

Revised traffic light problem: Most obvious FSM solution

One approach to the problem is to make a new design with eight states:

- four states for regular mode;
- four more states for parade mode.

Why are four different states needed for parade mode?

The eight-state FSM idea leads to the messy state transition diagram on the next slide ...

8-state transition diagram for traffic light system with "parade mode"

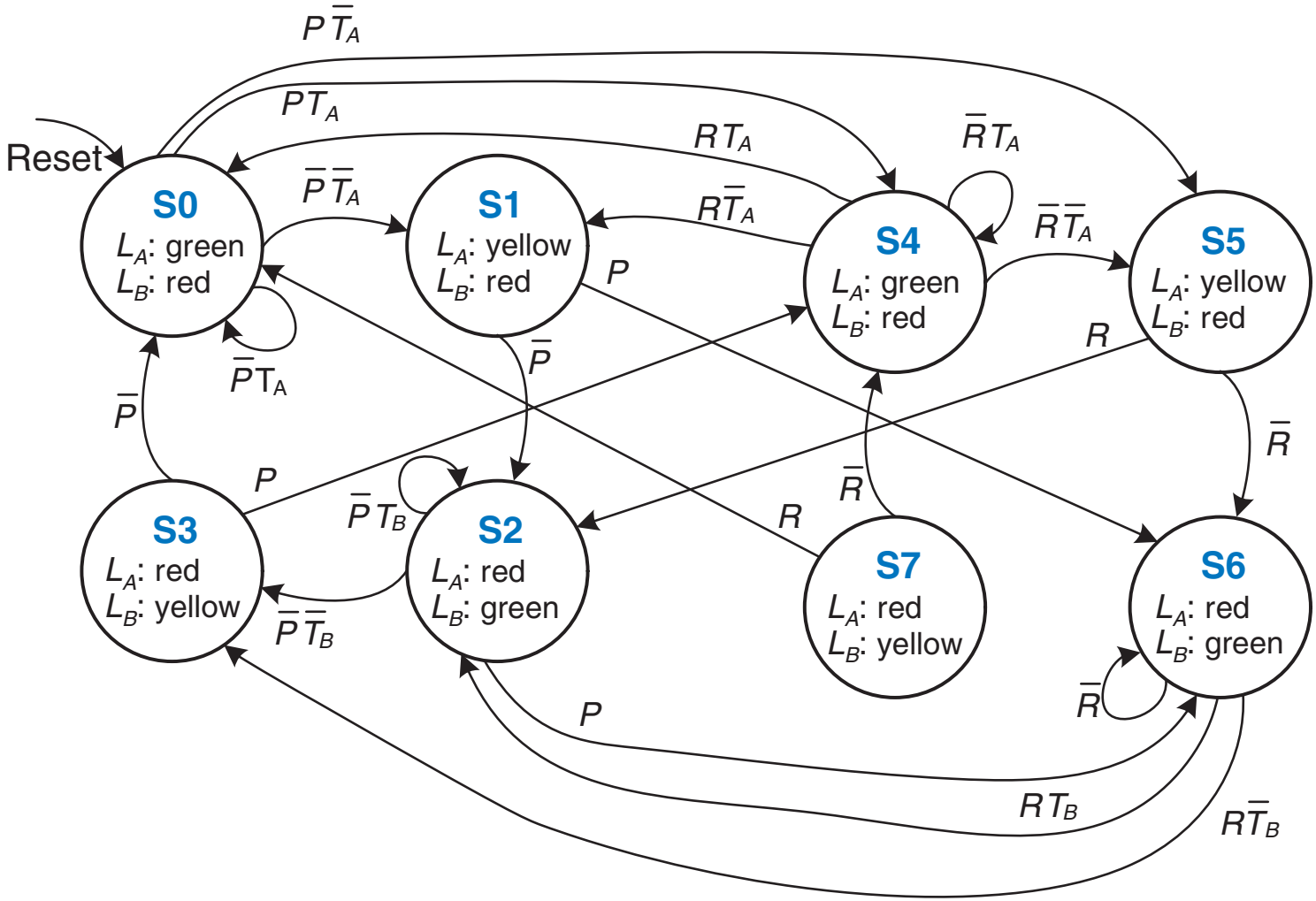


Image is taken from Figure 3.34 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.

Factoring FSMs

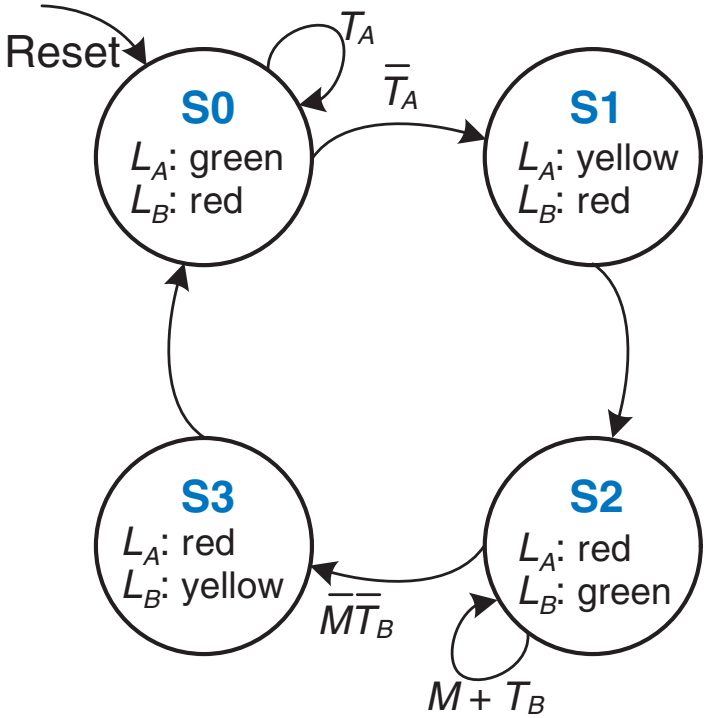
The parade-mode example is set up to make two main points:

- Trying to solve every synchronous sequential problem with a **single** FSM may result in unreasonably complex FSM designs.
- FSMs are nevertheless a great design tool. Sometimes it make sense to design a system as a **collection** of simple, **collaborating** FSMs.

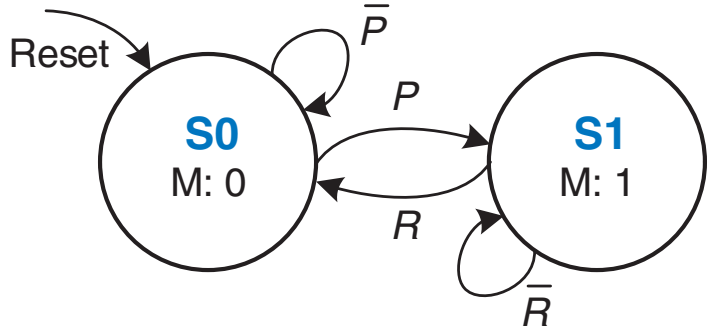
Factoring is the name given to the approach described in the second of the above two points.

Let's draw a block diagram to show how the traffic light controller with parade mode can be designed using two simple FSMs that work together.

State transition diagrams for factored FSM design



Lights FSM



Mode FSM

Image is taken from Figure 3.34 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.

Outline of Slide Set 8

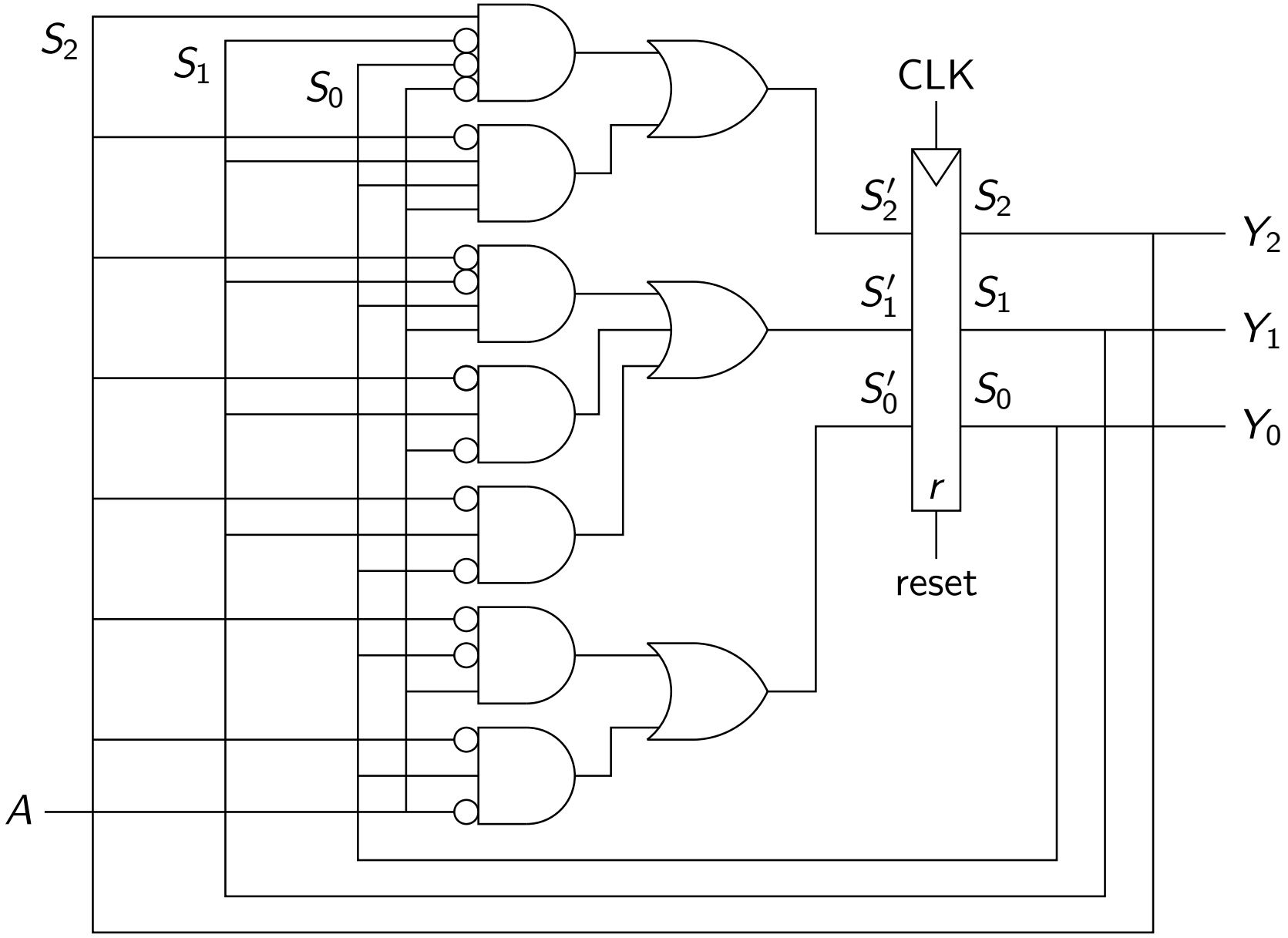
Deriving an FSM from a schematic

The next few slides closely follow the process used in Section 3.4.5 of the textbook, but use a completely different example schematic. (It's a good idea to study the textbook example as well as the lecture example.)

The first two steps in "reverse engineering" the schematic of an FSM are:

- Examine circuit, stating inputs, outputs, and state bits.
- Write next-state and output equations.

Let's perform those two steps for the schematic on the next slide . . .


S_2
S_1
S_0
CLK
S_2'
S_2
Y_2
S_1'
S_1
Y_1
S_0'
S_0
Y_0
r
reset
A

The next three steps are:

- Use next-state and output equations to create next-state and output tables. (The next-state table is ready for us on this slide, to save us all some boredom.)
- Reduce the next-state table to eliminate unreachable states.
- Assign each valid state bit combination a name.

Let's perform the last two of the above steps.

S_2	S_1	S_0	A	S_2'	S_1'	S_0'
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

Completion of the example FSM derivation problem

The final three steps are:

- Rewrite next-state and output tables with state names.
- Draw state transition diagram.
- State in words what the FSM does.

Let's work through these steps.