3.1 SR latches

Storing a bit

A ***sequential circuit***'s output is dependent on the present and the past *sequence* of input values, which necessarily means the circuit stores at least one bit. In contrast, a ***combinational circuit***'s output is dependent only on the present *combination* of input values.

The following system stores a bit to control a lamp.

PARTICIPATION ACTIVITY 3.1.1: A lamp controlled by on and off buttons stores a bit to remember whether keep the lamp on or off.

Animation captions:

1. The circuit stores a bit that controls the lamp: 1 illuminates, 0 doesn't.
2. The circuit remembers that the off button was pressed, and stays 0 after release.
3. The circuit remembers that the on button was pressed, and stays 1 after release.

PARTICIPATION ACTIVITY 3.1.2: Lamp that stores a bit.

Consider the example above.

1) When the On button is pressed, what value should be stored in the lamp?
   - ○ 0
   - ○ 1

2) When the On button is pressed and then released, what value should be stored in the lamp?
   - ○ 0
   - ○ 1

3) Suppose a user pressed On and released, then pressed Off and released. What should be stored in the lamp?
   - ○ 0

- 1

4) Without knowing any history of the button presses, a person observes that neither button is currently being pressed. Does the person know whether the lamp should be illuminated?
   - Yes
   - No

SR latch

The simplest circuit for storing a bit is called a ***latch***. An ***SR latch*** stores one bit, with an input s to set the latch to 1, an input r to reset the latch to 0, and with the stored bit appearing on output q. S and s are for "set", and R and r for "reset".

Below is a circuit for an SR latch. Reminder: NOR outputs 0 if any input is 1.

PARTICIPATION ACTIVITY 3.1.3: An SR latch stores a bit.

Animation captions:

1. An SR latch can be designed with two cross-coupled NOR gates (the output of each is an input to the other), with s input to top gate and r input to the bottom (whose output is q).
2. When s = 1, r = 0, the top gate outputs 0 (1 NOR anything is 0). The bottom gate outputs 1 (because both inputs are 0's). Thus q is 1; the latch has been "set".
3. When s returns to 0, q remains 1. The top gate still has a 1 input (from the bottom gate), and 1 NOR anything is 0. Bottom gate still has both inputs 0's, so outputs 1.
4. Now if r = 1 and s = 0, the bottom gate outputs 0 (1 NOR anything is 0), so q = 0. (The top gate's inputs are both 0's, so the gate outputs 1). The latch has been "reset".
5. When r returns to 0, the bottom gate still has a 1 input (from the top gate), so q remains 0. The top gate still has both inputs as 0's, so outputs 1.

Figure 3.1.1: SR latch behavior.

s	r	q
0	0	Previously-stored bit
0	1	0 ("Reset")
1	0	1 ("Set")
1	1	Unknown

PARTICIPATION ACTIVITY 3.1.4: SR latch.

Indicate q's present value for the given input sequence. "s: 0..1" means s was 0 and is presently 1.

1) s: 0
r: 1
   - 1
   - 0

2) s: 0..0
r: 1..0
   - 1
   - 0

3) s: 0..0..1
r: 1..0..0
   - 1
   - 0

4) s: 0..0..1..0
r: 1..0..0..0
   - 1
   - 0

5) s: 0..0..1..0..1
r: 1..0..0..0..0
   - 1
   - 0

SR = 11 is problematic

s = 1 and r = 1 causes a problem. The 1's cause the NOR gates to output 0's. When s and r return to 0's, then both gates output 1's. Those 1's propagate back to the gate inputs, causing the gates to output 0's. Those 0's propagate back, causing 1's again. No bit was stored, and instead the latch oscillates. ***Oscillate*** means to change from 0 to 1 to 0 to 1 repeatedly. Due to different gate and wire delays, eventually the latch will settle into a stored 0 or 1, but which one is unknown.

PARTICIPATION ACTIVITY 3.1.5: SR latch problem: sr = 11 doesn't store a bit and will cause oscillation.

Animation captions:

1. In an SR latch, s =1 and r = 1 may cause a problem. Each NOR gate has a 1 input so outputs 0.
2. A problem occurs when input sr change from 11 to 00. At that moment, each NOR gate has two 0's input, so each NOR gate's output changes from 0 to 1.
3. Those NOR gates' output 1's propagate back to the NOR gates' inputs. Each NOR gate now has a 1 input, so the gate's output changes to 0.
4. The output q oscillates between 0 and 1. Because gates/wires delays vary slightly, eventually q will settle at either 0 or 1, but which one is unknown.

PARTICIPATION ACTIVITY 3.1.6: SR latch when s = 1, r = 1.

1) Setting s and r to 1's simultaneously initially sets q to 1.
   - ○ True
   - ○ False

2) q oscillates while s and r are both 1's.
   - ○ True
   - ○ False

3) If s and r are both 1's, and then both change to 0's, q may oscillate.
   - ○ True
   - ○ False

SR latches are uncommon

SR latches were previously common when gates were expensive. But with gates far cheaper (and smaller) today, the more robust D latch (discussed later), which extends

an SR latch, is more common.

Buttons connected to circuits

For expedient learning, this material creates some simple circuit examples. In real design, those simple circuits would be more complex. Ex: A designer would never connect buttons directly to an SR latch, due in part to the oscillation issue. Likewise, a signal from a button is typically noisy ("bouncing" between 1 and 0 when pressed) so needs some additional circuitry. The authors ask experienced designers to allow such expediency, and ask students to realize that building real-world circuits requires more knowledge.

Example: Lamp with on/off buttons

The earlier lamp example can be implemented using an SR latch.

PARTICIPATION ACTIVITY 3.1.7: Lamp with on/off buttons implemented using an SR latch.

Animation captions:

1. A lamp with buttons for On and Off can be implemented with an SR latch.
2. Pressing the On button sets the latch to 1. After release, the lamp stays illuminated.
3. Pressing the Off button resets the latch to 0. After release, the lamp stays un-illuminated.

PARTICIPATION ACTIVITY 3.1.8: Lamp implemented using an SR latch.

Consider the example above.

1) While the On button is being pressed, does the lamp illuminate?
   - ○ Yes
   - ○ No
   - ○ Oscillates

2) If the On button is pressed and then

released, does the lamp stay illuminated?

- Yes
- No
- Oscillates

3) If the user presses both On and Off buttons and then releases both of them, does the lamp illuminate?

- Yes
- No
- May oscillate

CHALLENGE ACTIVITY 3.1.1: Indicate q's value over time.

347136.1658324.qx3zqy7

Start

s

r

q

1 2

Check Next

3.2 Clocks, D flip-flops, and registers

D latch

A ***D latch*** stores one bit, with an input d having the bit to be stored, an input e that when 1 enables storing the bit, and with the stored bit appearing on output q. D and d are short for "data", and e for "enable".

An SR latch has an oscillation problem if both inputs were 1's then both change back to 0's. A D latch can be implemented using an internal SR latch with s = d and r = d'. Because sr = 11 is impossible, the SR latch oscillation problem is avoided. When e = 1, the AND gates pass d to s, and d' to r. So d = 1 sets the SR latch, and d = 0 resets. When e = 0, the AND gates pass 0's to s and r, so the SR latch maintains the stored bit.

PARTICIPATION ACTIVITY 3.2.1: D latch.

Animation captions:

1. A D latch can be implemented using an SR latch, plus some logic gates.
2. e = 1 enables storing d's bit value. If d = 0, the internal SR latch gets sr = 01, so is reset to 0.
3. If e = 1 and d = 1, the internal SR latch is set to 1.
4. If e = 0, the internal SR latch's gets sr = 00, so the stored bit is maintained.
5. If e = 0, then d is ignored. The internal SR latch's inputs are sr = 00, so the stored bit is maintained.

Figure 3.2.1: D latch behavior.

e	d	q
0	0	Previously-stored bit (d is ignored)
0	1	Previously-stored bit (d is ignored)
1	0	0 (d is stored)
1	1	1 (d is stored)

PARTICIPATION ACTIVITY 3.2.2: D latch.

Indicate q's present value for the given input sequence. "d: 0..1" means d was 0 and is presently 1.

1) d: 0
e: 1

- 1
- 0

2) d: 0..0
e: 1..0

- 1
- 0

3) d: 0..0..1
e: 1..0..0

- 1
- 0

4) d: 0..0..1..1
e: 1..0..0..1

- 1
- 0

5) d: 0..0..1..1..0
e: 1..0..0..1..0

- 1
- 0

Clock

Most digital circuits use an oscillating signal called a ***clock*** signal to control when to store bits (for reasons that will be clear later). Rather than bits being stored while an enable signal is high, bits are stored only at a clock's ***rising edge***: The instant a clock signal changes from 0 to 1.

- A ***clock cycle*** is the time between two rising edges, that time being the ***clock period***.
- ***Clock frequency*** is the cycles per second, in units of ***hertz*** (***Hz***) meaning cycles/second. Ex: A 1 MHz clock has 1 million cycles per second. Frequency is the inverse of period. Ex: A 1 microsecond (0.000001) period yields a frequency of 1 MHz (because 1 / 0.000001 s = 1,000,000 Hz).

PARTICIPATION ACTIVITY 3.2.3: Clocking synchronizes components.

Animation captions:

1. Sequential circuits use a clock to synchronize storing of bits to only occur on rising clock edges. Such synchronization greatly simplifies sequential circuit design.
2. One clock cycle is one rising edge to the next. Clock period is the time for one cycle. Frequency is 1 / period.
3. A special device called an oscillator generates a clock at a specific frequency. On a component, a small triangle indicates the clock input.

PARTICIPATION ACTIVITY 3.2.4: Clocks.

Consider the example above.

1) A change of the clock signal from 0 to 1 is called a rising _____ .
   - edge
   - cliff

2) The three boxes represent three stored bits. On each rising clock edge, those bits move one place to the _____ .
   - left
   - right

3) On each rising clock edge, the bits move _____ .
   - one at a time
   - simultaneously

4) A clock's period is the time for one _____.
   - edge
   - cycle

5) If a clock's period is 1 microsecond, the clock's frequency is _____.
   - 1 MHz
   - 1 GHz

6) A _____ indicates the clock input of a component.

- circle
- triangle

D flip-flops

A ***latch*** stores a new bit while an enable input is 1. A latch is said to be ***level sensitive***, storing when the enable's "level" is high. In contrast, a ***flip-flop*** stores a new bit only at the instant of clock input's rising edge. A flip-flop is said to be ***edge-triggered***.

PARTICIPATION ACTIVITY 3.2.5: D flip-flop.

Given a single D flip-flop with data input d, clock input clk, and output q.

1) d is 0, clk is 0, q is 0.
d changes to 1. Moments later, what is q?
   - 1
   - 0

2) d is 0, clk is 0, q is 0.
clk changes to 1. Moments later, what is q?
   - 1
   - 0

3) d is 0, clk is 0, q is 0.
d changes to 1, then clk changes to 1. Moments later, what is q?
   - 1
   - 0

4) d is 1, clk is 0, q is 1.
d changes to 0. Moments later, what is q?
   - 1
   - 0

A D flip-flop can be implemented by cascading two D latches, with the first latch's enable inverted. A flip-flop implemented by two latches, with the first latch's enable inverted, has a ***main-secondary*** arrangement. The first latch is the main, the second the secondary. Other implementations exist.

PARTICIPATION ACTIVITY 3.2.6: D flip-flop: Main-secondary implementation.

Animation captions:

1. A D flip-flop can be built as a first D latch whose output q is the input d of a second D latch. A clock input connects to the e input of each, inverted for the first.
2. While clk is 0, the first latch is enabled, so the input d value is stored into the latch. The second latch is disabled, so that value does not propagate into the second latch.
3. When clk rises from 0 to 1, the second latch becomes enabled (and first latch disabled). The value in the first latch (when clk was 0) gets stored in the second latch.

PARTICIPATION ACTIVITY 3.2.7: D flip-flop implemented as main-secondary.

Given a single D flip-flop implemented with a main-secondary arrangement.

1) d is 0, clk is 0, q is 1.
What is in the first latch?

- 1
- 0

2) d is 0, clk is 0, q is 1.
What is in the second latch?

- 1
- 0

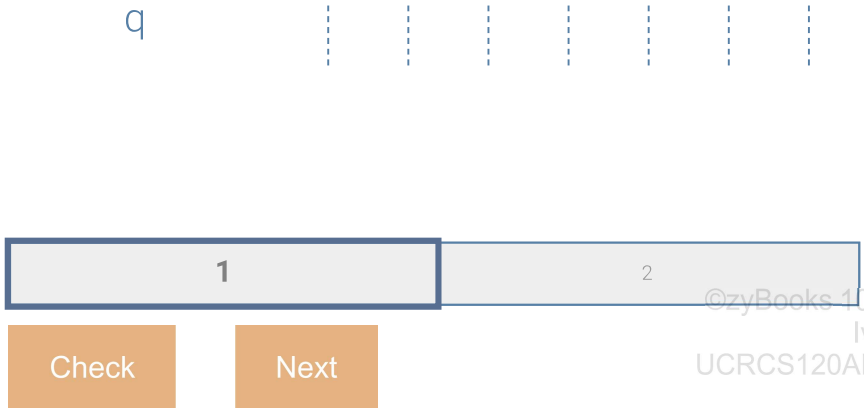
CHALLENGE ACTIVITY 3.2.1: Indicate q's value over time.

347136.1658324.qx3zqy7

Start

d

e



Basic register

Bits are usually stored in groups. A ***register*** is a circuit that stores a group of bits; a 3-bit register stores three bits. On a rising clock edge, all bits are stored simultaneously. Storing bits in a register is known as ***loading*** the register. A common register implementation uses flip-flops. (Registers are discussed more later).

PARTICIPATION ACTIVITY 3.2.8: 3-bit register using 3 flip-flops: Inputs only stored on rising clk.

Animation captions:

1. A register stores multiple bits. A common implementation uses one flip-flop per bit, all enabled by the same clk signal.
2. When clk rises, all 3 bits load simultaneously.
3. When clk falls, the register maintains those values, until the next clk rise.

PARTICIPATION ACTIVITY 3.2.9: Basic register.

Given a 3-bit register with data inputs d2, d1, d0, clock input clk, and outputs q2, q1, q0.

1) d2d1d0 is 101, and q2q1q0 is 000. clk rises. What does q2q1q0 become?

Check Show answer

2) d2d1d0 is 111, q2q1q0 is 000, clk rises so q2q1q0 becomes 111.

Then d2d1d0 changes to 010.
What does q2q1q0 become?

Check Show answer

3.3 Example: Flight attendant call button using a D flip-flop

Flight attendant call button design

A flight attendant call button has two buttons: call and cancel. If the call button is pressed, the light is turned on. If the cancel button is pressed, the light is turned off. If both are pressed at the same time, the call button gets priority, so the light is turned on. If neither button is pressed, the light does not change.

A D flip-flop and an additional circuit can be used to capture the desired behavior.

PARTICIPATION ACTIVITY 3.3.1: Flight attendant call button using a D flip-flop.

Animation content:

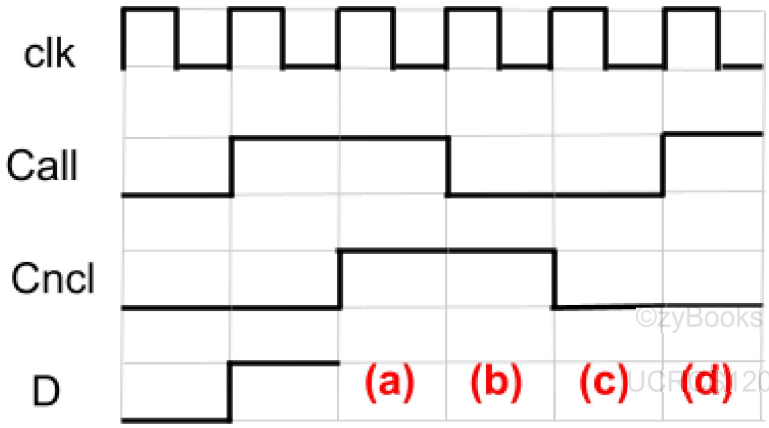
undefined

Animation captions:

1. If Call is pressed, a 1 is stored, and the blue light is turned ON.
2. If Cncl is pressed, a 0 is stored, and the blue light is turned OFF.
3. If neither button is pressed, the present value of Q is stored back in the flip-flop.
4. If both buttons are pressed, priority is given to Call, so a 1 is stored.

PARTICIPATION ACTIVITY 3.3.2: Timing diagram for the flight attendant call button example.

Complete the timing diagram for the example above.



1) (a)

- 0
- 1

2) (b)

- 0
- 1

3) (c)

- 0
- 1

4) (d)

- 0
- 1

Flight attendant call button: Combinational logic

A truth table is used to specify the combinational logic's functionality and then convert the design into a circuit.

PARTICIPATION ACTIVITY 3.3.3: Flight attendant call button example.

Animation content:

undefined

Animation captions:

1. The flight attendant call button system behavior can be captured as a truth table.
2. If Call = 0 and Cncl = 0, D = Q. If Call = 0 and Cncl = 1, D = 0. If Call = 1, D = 1 as Call has priority over Cncl.
3. D's equation is derived in sum-of-minterms form and reduced using boolean algebra to a simplified equation.
4. The simplified equation is converted into a combinational circuit which completes the flight attendant call button system.

PARTICIPATION ACTIVITY 3.3.4: Flight attendant call button example.

Refer to the animation above.

1) How many 3-input AND gates are needed for the sum-of-minterms equations for D?

Check Show answer

2) If the system behavior was changed such that when Call = 1 and Cncl = 1, D = 0, how many AND gates will be needed for the sum-of-minterms equation for D?

Check Show answer

3.4 FSMs

FSMs describe sequential behavior

A circuit is ***combinational*** if the circuit's output values depend solely on the present combination of input values. In contrast, a circuit is ***sequential*** if the circuit's output values depend not only on the present input values, but also on the sequence of past values.

An equation straightforwardly describes combinational behavior, but lacks the capability of describing sequential behavior. An ***FSM*** (***finite-state machine***) is a computation model capable of describing

sequential behavior.

PARTICIPATION ACTIVITY 3.4.1: FSM introduction.

Animation captions:

1. An FSM has inputs, outputs, states, state actions, transitions, and an initial state. This FSM has no inputs, output x, states s0 (action x = 0) and s1 (x = 1), and two transitions.
2. When executing, an FSM transitions to the next state on each rising clk.
3. Transitions may have an input condition. This FSM has input g. Transitions s0-s1 and s1-s0 have condition g. Transitions s0-s0 and s1-s1 have condition g' (not g).
4. At each rising clock, from a state, the FSM takes the transition with a true condition. When in s0, if g = 0, the transition back to s0 is taken.
5. From s0, if g = 1 instead, the transition to s1 is taken.
6. A FSM can describe behavior of a desired sequential circuit.

PARTICIPATION ACTIVITY 3.4.2: Sequential behavior and FSMs.

Indicate whether the behavior should be captured as an equation or as an FSM.

1) If inputs a b c[f] are 011, output y = 1. Else y = 0.
   - ○ Equation
   - ○ FSM

2) Output y is initially 0. If input a = 0, then a = 1, then a = 0, y becomes 1.
   - ○ Equation
   - ○ FSM

3) When a user presses a button, a light turns on. When the user releases the button, the light turns off.
   - ○ Equation
   - ○ FSM

4) When a user first presses a button, a light turns on and stays on, even after button release. The next button press turns the light off. The next button press turns the light on, and so forth[h].

- Equation
- FSM

FSM execution

An FSM consists of inputs, outputs, states, state actions, transitions, and an initial state. When executing ("running"), the FSM is always in one state. A ***state*** is the present "situation" of a digital system, akin to a person's state being sleeping, eating, or working. The FSM starts in an initial state. Every time an implicit clock input rises, the FSM changes to a next state pointed to by a transition whose condition evaluates to true (1). A transition without a listed condition has an implicit condition of true.

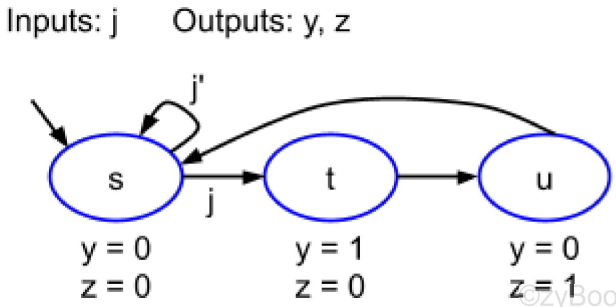
PARTICIPATION ACTIVITY 3.4.3: FSM execution and timing diagrams.

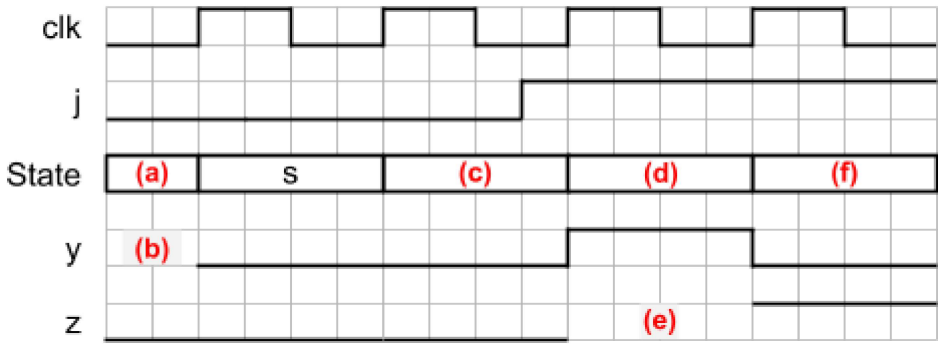
Animation captions:

1. A timing diagram can illustrate the FSM's current state and output value.
2. The initial state is graphically indicated by a transition coming from nothing.
3. An FSM transitions to the next state at each rising clk.
4. Transitions may have an input condition.
5. At each rising edge, an FSM changes to a next state pointed to by a transition whose condition evaluates to true (1).
6. At each rising edge, an FSM changes to a next state pointed to by a transition whose condition evaluates to true (1).

PARTICIPATION ACTIVITY 3.4.4: FSM execution.

Complete the timing diagram.





1) (a)

Check Show answer

2) (b)

Check Show answer

3) (c)

Check Show answer

4) (d)

Check Show answer

5) (e)

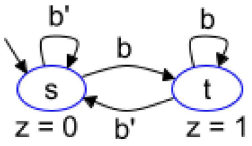
Check Show answer

6) (f)

Check Show answer

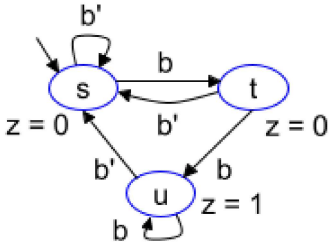
PARTICIPATION ACTIVITY 3.4.5: FSM descriptions.

Inputs: b
Outputs: z



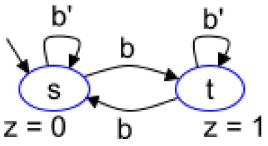
(a)

Inputs: b
Outputs: z



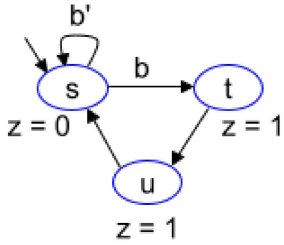
(b)

Inputs: b
Outputs: z



(c)

Inputs: b
Outputs: z



(d)

Match the above FSMs to their English descriptions.

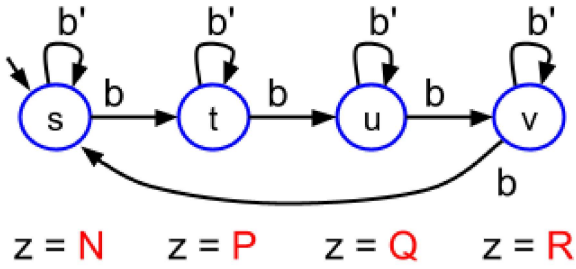
(a) (c) (b) (d)

	When b is 1, z is 1 for two clock cycles.
	When b is 1, z is 1. When b is 0, z is 0.
	When b is 1 for at least two cycles, z becomes 1 and remains 1 until b is 0.
	When b is 1, z repeatedly changes between 1 and 0. When b is 0, z maintains the current output value.

Reset

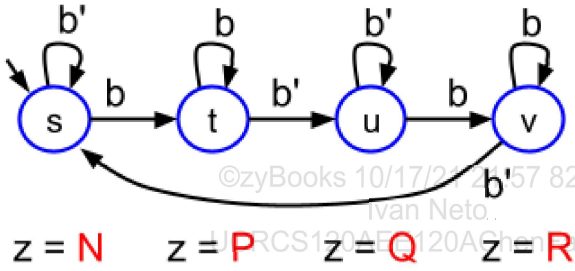
PARTICIPATION ACTIVITY 3.4.6: FSM output assignments.

Inputs: b
Outputs: z



(1)

Inputs: b
Outputs: z



(2)

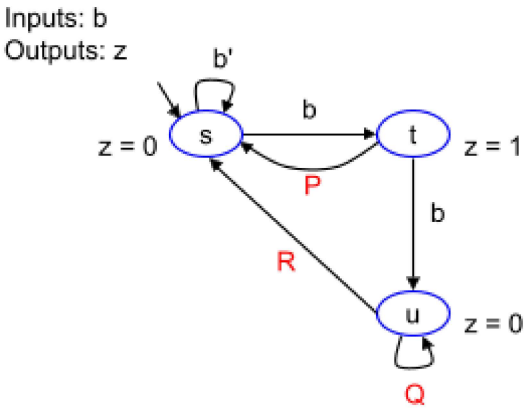
The above FSMs have input b and output z. If a user presses a button, b becomes 1. If a user releases a button, b becomes 0. To turn a light on, the FSM sets z to 1. To turn a light off, the FSM sets z to 0.

Determine the values of N, P, Q, and R given the desired FSM behavior.

1) For FSM (1): A light is initially off. If a user presses a button, the FSM will repeatedly blink the light on and off.
   - NPQR = 1111
   - NPQR = 1010
   - NPQR = 0101

2) For FSM (2): A light is initially on. The first button press turns the light off. The second button press turns the light on. Subsequent button presses switch the light off and on.
   - NPQR = 1010
   - NPQR = 0110
   - NPQR = 1001

PARTICIPATION ACTIVITY 3.4.7: FSM transition conditions.



For each unique 0-to-1 transition of b, the above FSM sets z to 1 for exactly one clock cycle. Determine the missing transition conditions.

1) (P)

- b
- b'

2) (Q)

- b
- b'

3) (R)

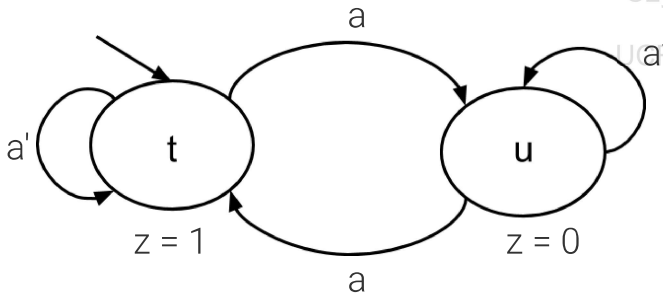
- b
- b'

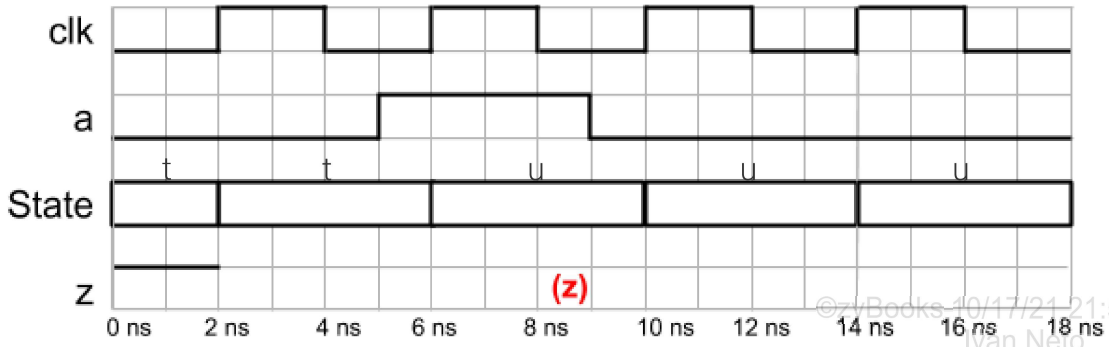
CHALLENGE ACTIVITY 3.4.1: FSMs.

347136.1658324.qx3zqy7

Start

Indicate the value of z for the missing clock cycles.





(z) = Ex: 0001

1 2 3

Check Next

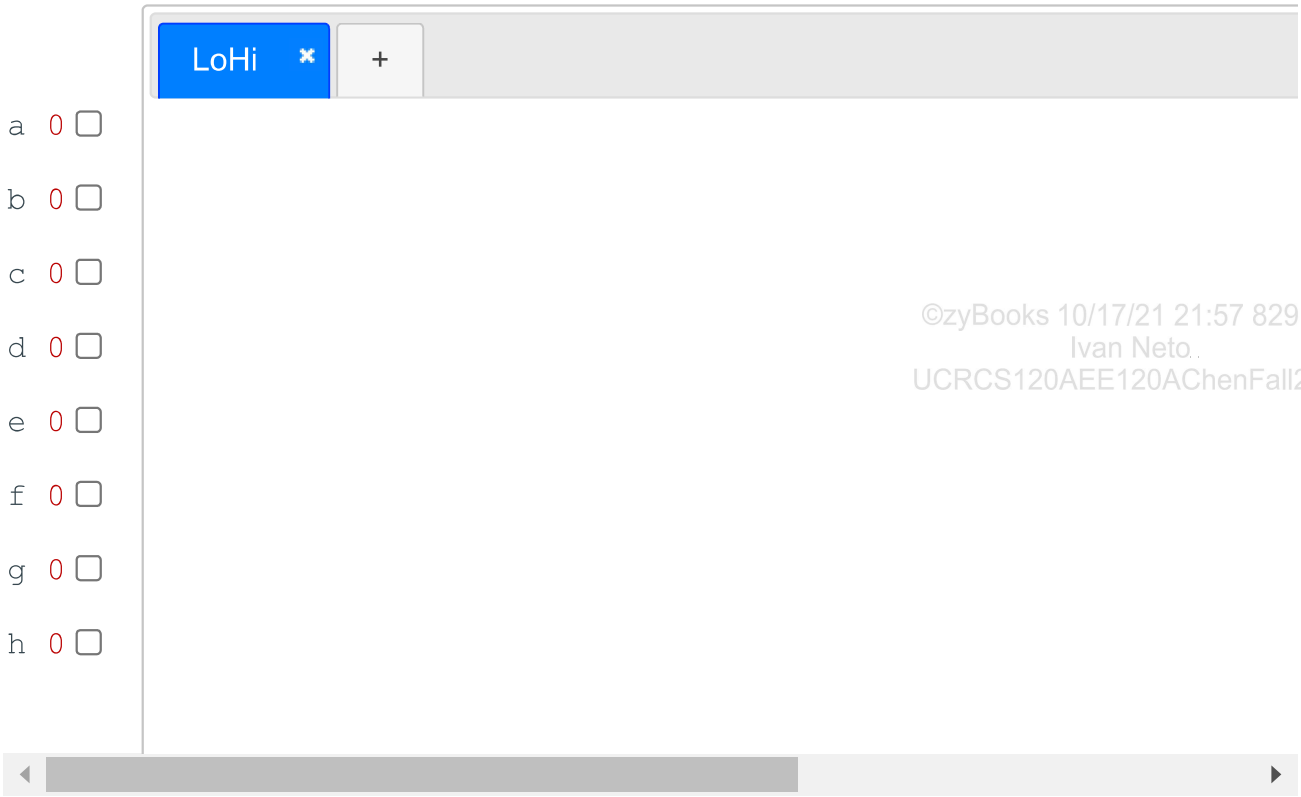
3.5 FSM simulator

This material uses a simple browser-based FSM simulator.

PARTICIPATION ACTIVITY 3.5.1: FSM simulator. Full screen

- Press "Simulate", observe FSM executing. Press "a" to change to 1, note state and output changes. Press "a" again. Press "End simulation".
- Modify FSM by adding action "t = 1" to Lo, "t = 0" to Hi. Simulate.
- Modify FSM by inserting state All, sets s = 1, t = 1. Delete transition with a', insert transition with a' from Hi to All. Insert transition to stay in All while c', and another to go to Lo when c. Simulate.

Simulate Pause Insert state Insert transition



PARTICIPATION ACTIVITY 3.5.2: FSM simulator basics.

1) "Period" indicates the FSM's clock period.
   - True
   - False

2) The FSM's initial state is set by double-clicking the state.
   - True
   - False

3) After inserting a state, actions can be added by typing in the "Actions" box on the far right.
   - True
   - False

4) A state or transition is deleted by dragging the item off-screen.
   - True
   - False

5) When an FSM is executing (by pressing "Simulate"), the values of inputs a, b, c, ..., cannot be changed.

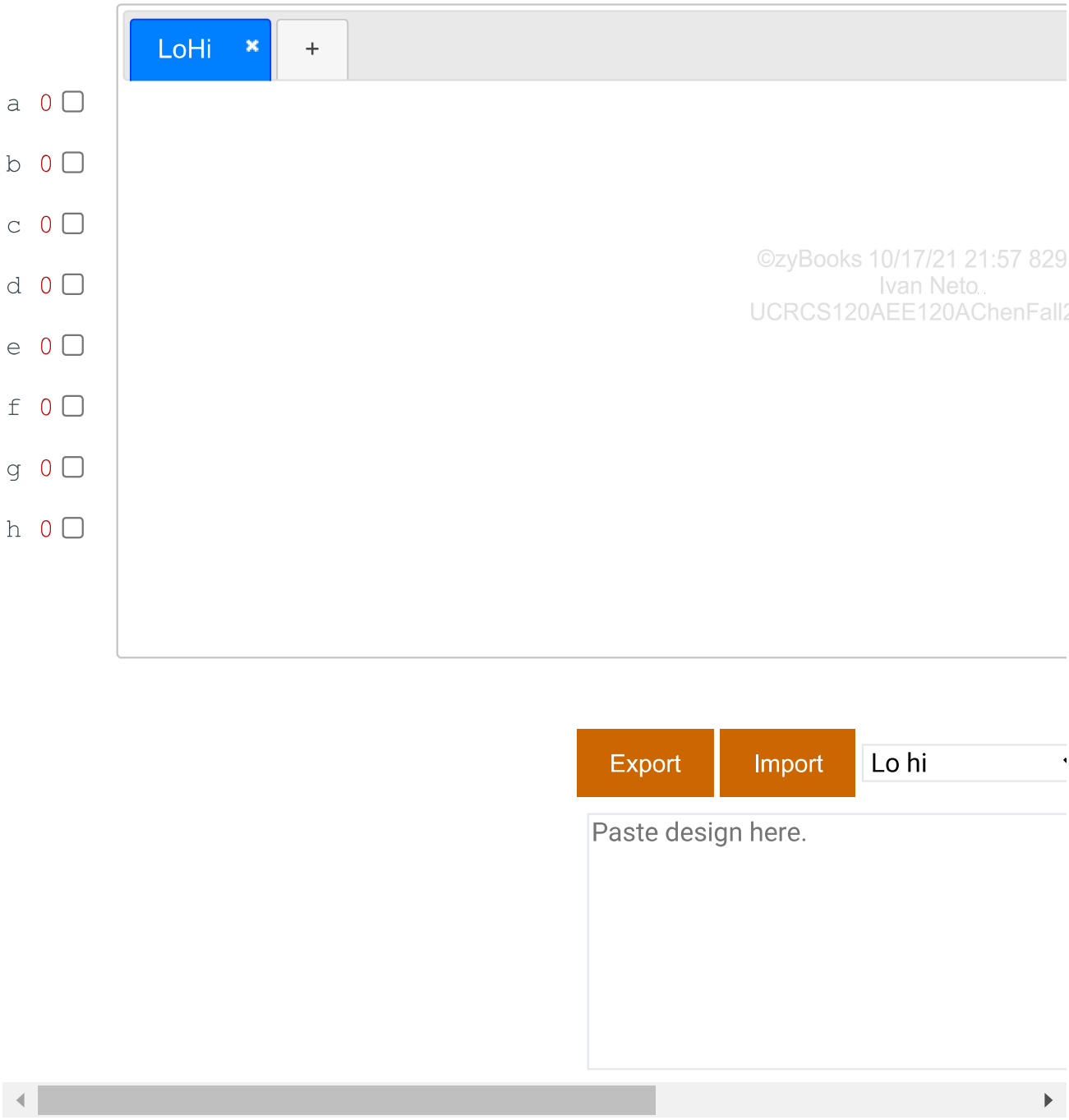
- True
- False

PARTICIPATION ACTIVITY 3.5.3: FSM simulator with export/import.

Full screen

- FSMs can be saved. Press "Export" and copy-paste the exported text. Modify the FSM somehow. Then paste the text into the box and press "Import" -- the previously-exported FSM is restored. (Users can save the exported text in a file or email for future use.)

Simulate Pause Insert state Insert transition



PARTICIPATION ACTIVITY 3.5.4: FSM simulator with export.

1) Pressing "Export" saves the current FSM to a file.

- True
- False

3.6 Capturing behavior with FSMs

To ***capture*** behavior means for a designer to describe desired behavior in some form. Designers commonly use an FSM to capture sequential behavior. This section presents common sequential behaviors captured as FSMs.

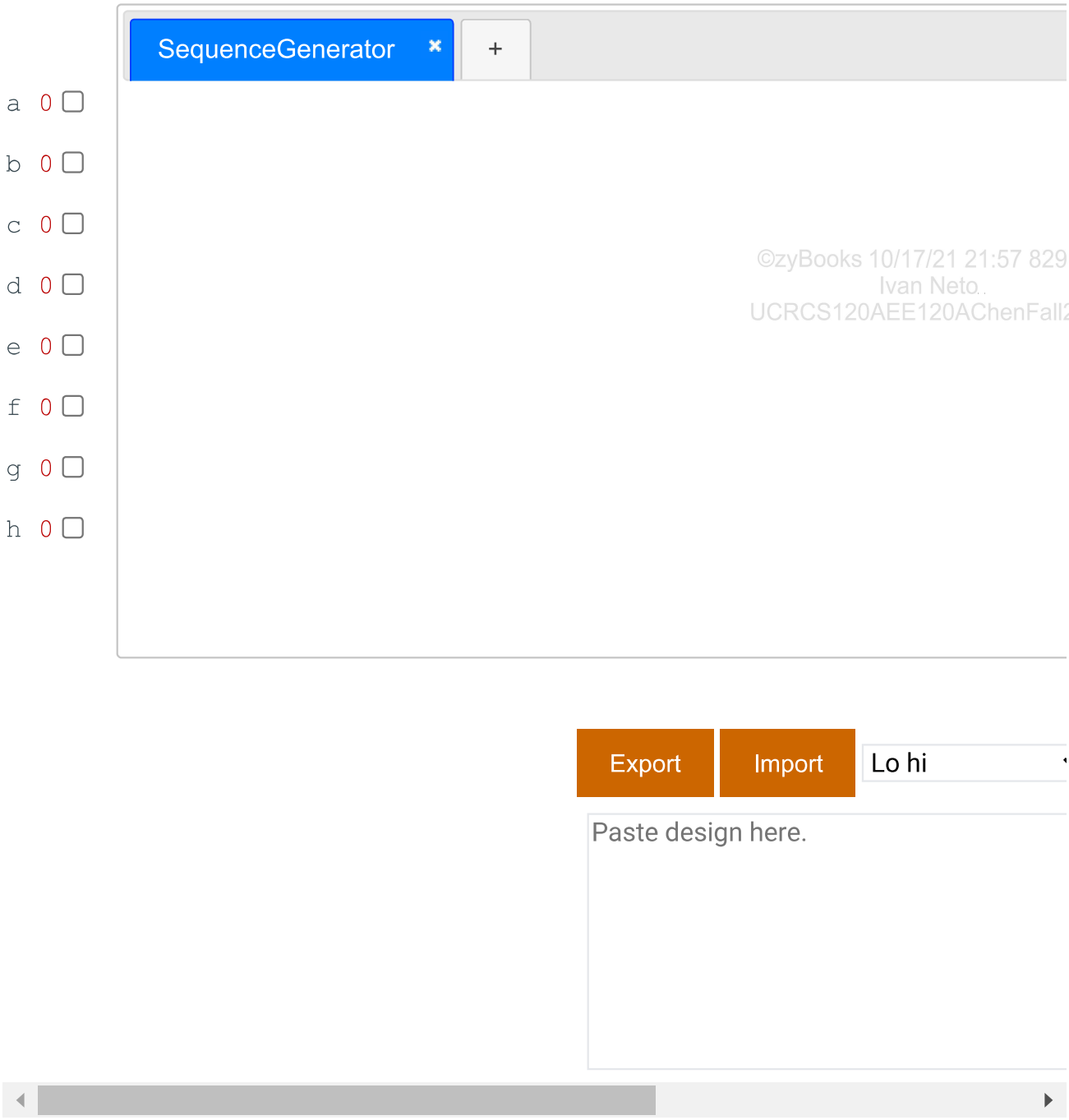
Common sequential behavior: Sequence generator

A common sequential behavior captured as an FSM is generating a sequence of patterns on several outputs, such as for a festive lights display, a dancing fountain show, or to address a particular device among many devices. The following FSM repeatedly generates the sequence 000, 101, 010, 111 on outputs s, t, u.

PARTICIPATION ACTIVITY 3.6.1: Common behavior: Sequence generator. Full screen

Press "Simulate" to observe the pattern on outputs s, t, u.

Simulate Pause Insert state Insert transition



For a festive lights display, a designer might use a slow clock frequency like 1 Hz (meaning a cycle is 1 sec).

PARTICIPATION ACTIVITY 3.6.2: Sequence generator.

Consider the example above.

1) What is output on s, t, u when the FSM starts executing?

- 000
- 101

2) What is output on s, t, u after the outputs 111 appear?

- 000
- 111

3) While the FSM executes, when does the FSM stop outputting the sequence?

- After state q
- Never

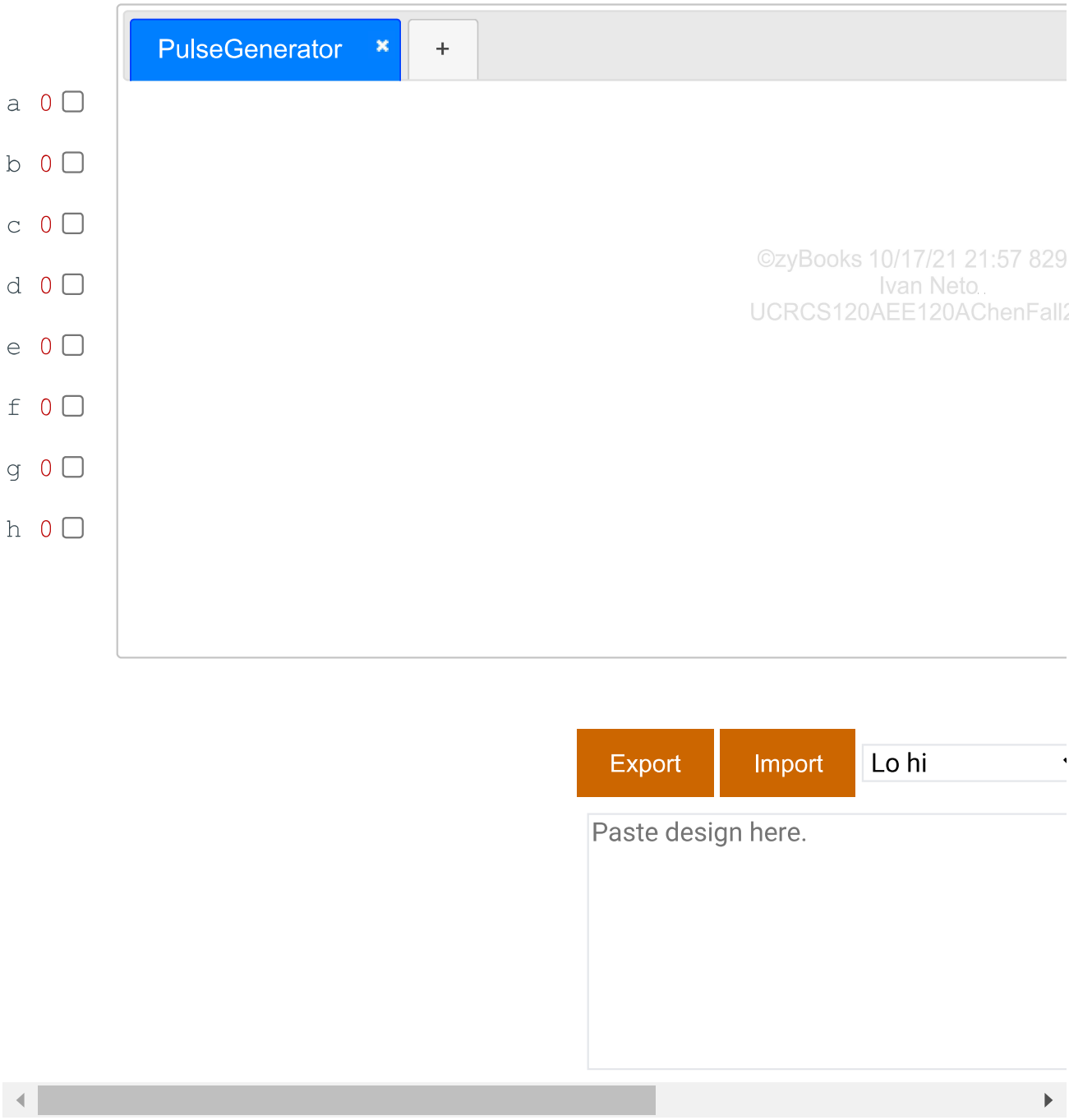
Common sequential behavior: Pulse generator with enable

A common sequential behavior captured as an FSM is generating a pulsing output when an input is 1. A ***pulse*** is the changing of a signal from 0 to 1 and back to 0. Examples might include a pulsing light on a police car, a dance strobe light, a blinking light above a cashier requesting manager attention, or a beeping sound indicating a refrigerator door is open. An input that controls whether a behavior occurs is called an ***enable*** input.

PARTICIPATION ACTIVITY 3.6.3: Common behavior: Pulse generator with enable. Full screen

Press "Simulate", then set a to 1 and observe that output s pulses. Set a to 0 to stop the pulsing.

Simulate Pause Insert state Insert transition



PARTICIPATION ACTIVITY 3.6.4: Pulse generator with enable.

Consider the example above.

1) When in Off, if a is 0, do pulses appear on s?
   - Yes
   - No

2) When a is 1, does the FSM stay in state PulseHigh?

- Yes
- No

3) While pulsing, if a becomes 0, the FSM soon returns to Off.

- Yes
- No

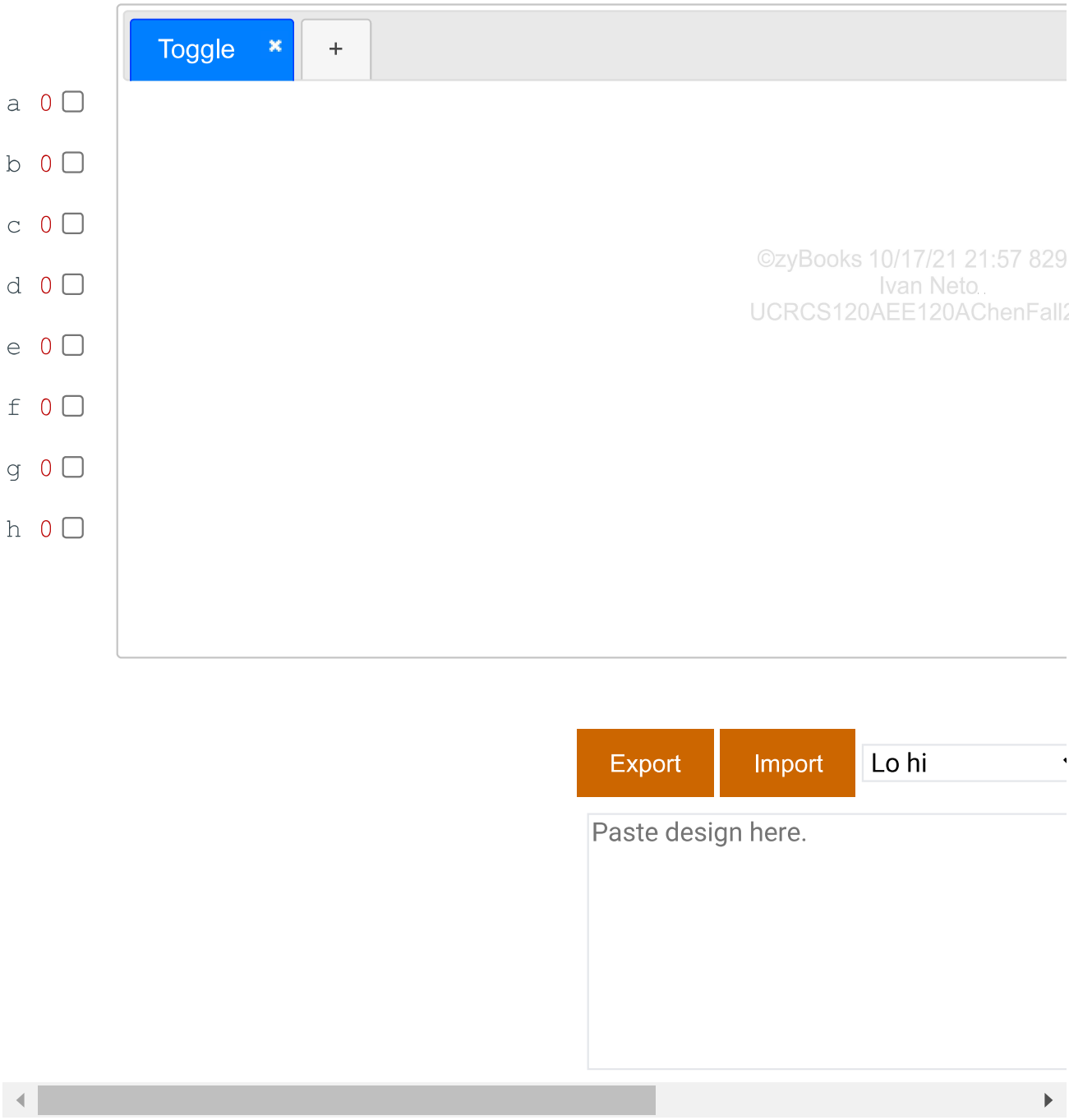
Common sequential behavior: Toggle

A common sequential behavior captured as an FSM toggles a bit whenever some input condition occurs, such as a button whose presses turn a light on or off, lock or unlock a door, or cause an alarm to sound or not sound. ***Toggle*** means to change to the opposite situation, such as from 0 to 1 or from 1 to 0. In the FSM below, after the FSM detects input a rising from 0 to 1, the FSM waits in state On1 or Off1 for a to fall, so that each pulse only causes one toggle regardless of the pulse's duration.

PARTICIPATION ACTIVITY 3.6.5: FSM behavior: Toggle when input rises. **Full screen**

Press Simulate. Then set a with 1, and notice that the output toggles from 0 to 1. Set a with 0, then again with 1, and notice the output toggles from 1 to 0. Repeat a few more times.

Simulate Pause Insert state Insert transition



PARTICIPATION ACTIVITY 3.6.6: Toggle.

Consider the example above.

1) Suppose the FSM just started execution and so is in state Off2, with s = 0. Then input a rises from 0 to 1. Does the output s toggle?

- ○ Yes
- ○ No

2) When in state On1, if input a remains 1 for multiple clock cycles, then output s will toggle from 1 to 0 to 1 to 0 and so on.

- True
- False

3) When in state On1, if input a falls to 0, the output s will toggle from 1 back to 0.

- True
- False

4) While in On2, if input a rises to 1, the output s will toggle from 1 to 0.

- True
- False

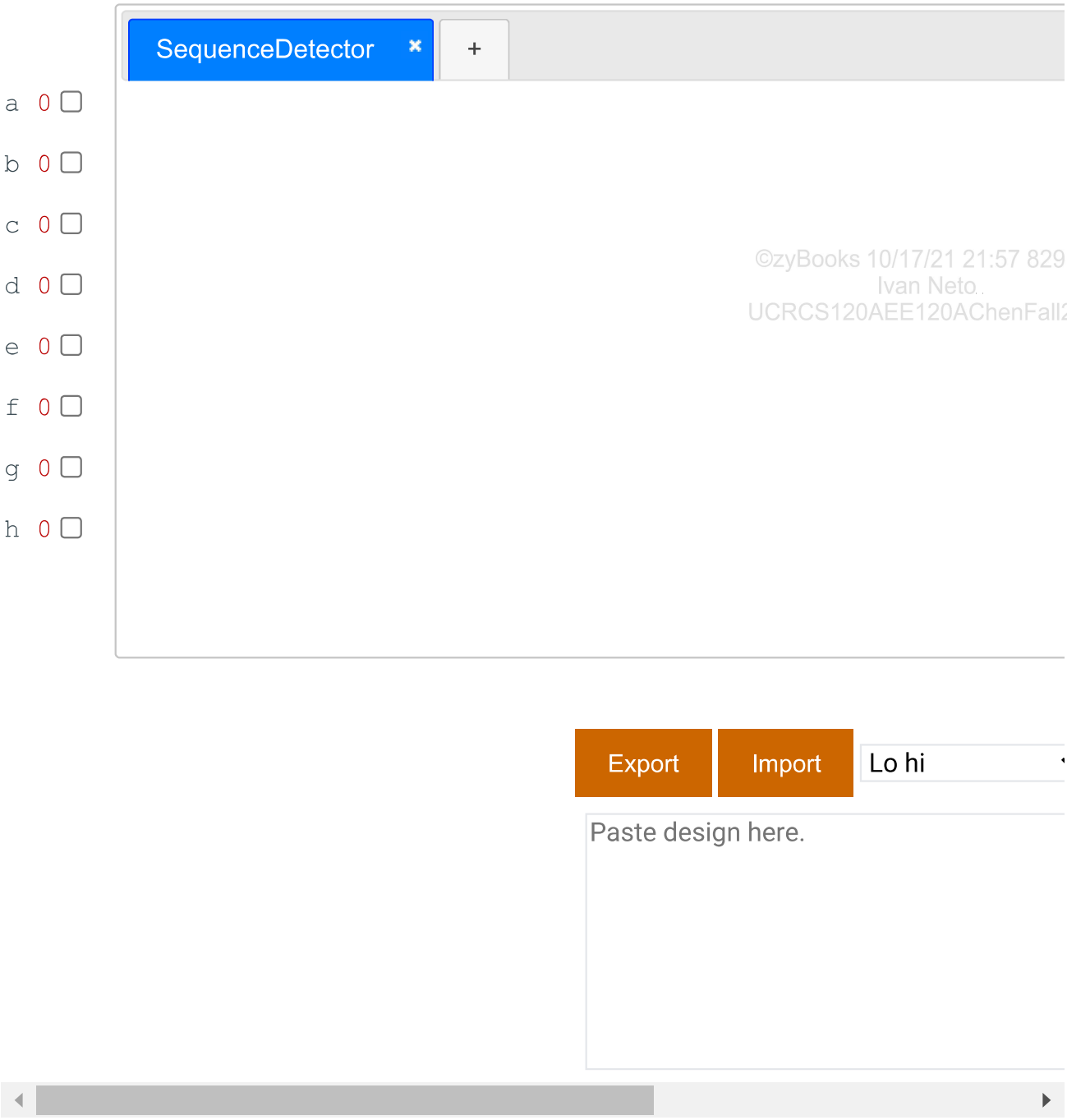
Common sequential behavior: Sequence detector

A common behavior captured as an FSM detects a specific sequence on an input bit, such as the sequence 101. Such a sequence detector might be used to inform a particular device, like a remote-controlled light or TV, to turn on, with difference devices detecting different sequences.

PARTICIPATION ACTIVITY 3.6.7: FSM example: Sequence detector. Full screen

Press "Simulate", then set a = 1, then 0, then 1, each for only 1 clock cycle, and note that s pulses to 1 for one clock cycle. Try other input patterns on a, and note that s does not pulse; only the sequence 101 causes an output pulse. Note: You must time your clicks to switch once per clock cycle.

Simulate Pause Insert state Insert transition



PARTICIPATION ACTIVITY 3.6.8: Sequence detector.

Consider the example above.

1) The sequence 101 on input a can occur at any rate to cause output s to pulse.
   - Yes
   - No

2) The sequence 101101 causes two pulses on s.

- Yes
- No

3.7 Example: Flight-attendant call button using an FSM

Another section presented a flight-attendant call button system designed using a D flip-flop. The flight attendant call button system can also be designed using an FSM.

A flight attendant call button has two buttons: call and cancel. If the call button is pressed, the light is turned on. If the cancel button is pressed, the light is turned off. If both are pressed at the same time, the call button gets priority, so the light is turned on. If neither button is pressed, the light does not change.

PARTICIPATION ACTIVITY 3.7.1: Flight attendant call-button example using an FSM.

Animation content:

undefined

Animation captions:

1. The FSM has inputs Call and Cncl for the call and cancel buttons, and output L to control the light. Call is given priority if both buttons are pressed at the same time.
2. The FSM has two states: LightOff (L = 0) and LightOn (L = 1). LightOff is the initial state.
3. The FSM stays in LightOff until Call is 1. If Call = 1, the FSM transitions to LightOn.
4. In LightOn, the FSM transitions back to LightOff if Cncl is 1 and Call is 0, as Call has priority. The transition's condition is CnclCall'.
5. In LightOn, if the transition condition CnclCall' is false ((CnclCall')'), the FSM stays in LightOn state.

PARTICIPATION ACTIVITY 3.7.2: Flight attendant call button example using an FSM.

1) What is output on L when Call = 1?

- 0
- 1

2) When in LightOn, what values of Call and Cncl will cause the FSM to transition to LightOff?

- Call = 0, Cncl = 1
- Call = 1, Cncl = 1

3) What is output on L when the FSM just starts executing?

- 0
- 1

4) When in LightOn, what values of Call and Cncl will cause the FSM to stay in LightOn?

- Call = 1, Cncl = 0
- Call = 0, Cncl = 1

3.8 FSM examples

Example: Construction sign

A highway construction sign illuminates LEDs to tell drivers to move over, as illustrated below. The behavior can be captured as an FSM to generate the necessary output sequence. Input a = 1 activates the sequence, controlled by a switch on the sign.

PARTICIPATION ACTIVITY 3.8.1: Move-over highway construction sign.

Animation content:

Construction sign has 4 LEDs horizontally aligned. Drivers are told to move to the right by blinking LEDs in the following pattern: leftmost LED blinks, then LED to one position to the right blinks, then LED one more position to the right blinks, then rightmost LED blinks. A controller can be designed to coordinate the light sequence with input a and outputs s, t, u, v. Each output controls an LED in the pattern. Output s connected to the leftmost LED, followed by output t, output u, and output v connected to the rightmost LED.

Animation captions:

1. This highway sign has four LEDs that can be illuminated.
2. The sign generates an output sequence that informs drivers to move right. The sequence would then repeat.
3. A sequential circuit, described by an FSM, can control the LEDs.

PARTICIPATION ACTIVITY 3.8.2: Move-over highway construction sign.

Full screen

Press simulate, observe the output pattern that gives the illusion of an item moving downwards. Try extending to have the fourth state set 3 outputs to 1's (illuminating an arrowhead).

Simulate Pause Insert state Insert transition

a 0

b 0

c 0

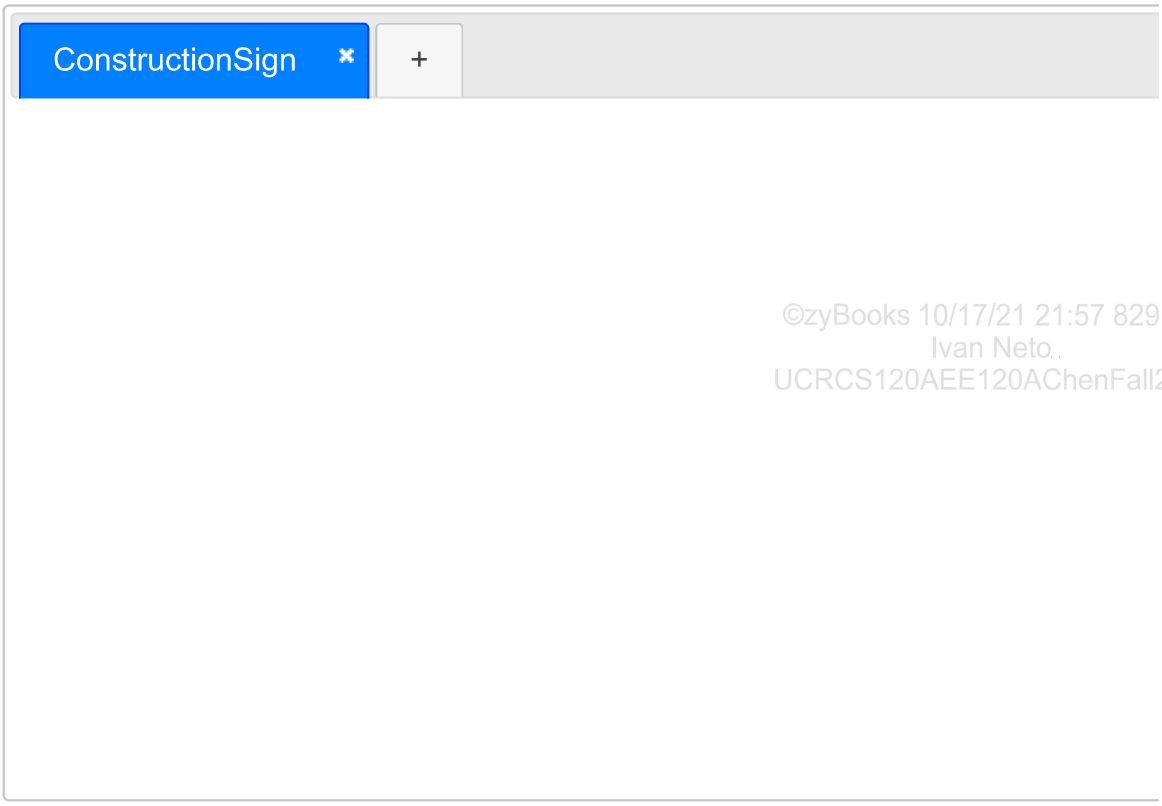
d 0

e 0

f 0

g 0

h 0



PARTICIPATION ACTIVITY 3.8.3: Construction sign.

Consider the example above.

1) Suppose the FSM just started execution and so is in state m. If a is 0, how many LEDs are illuminated?

- 0
- 1

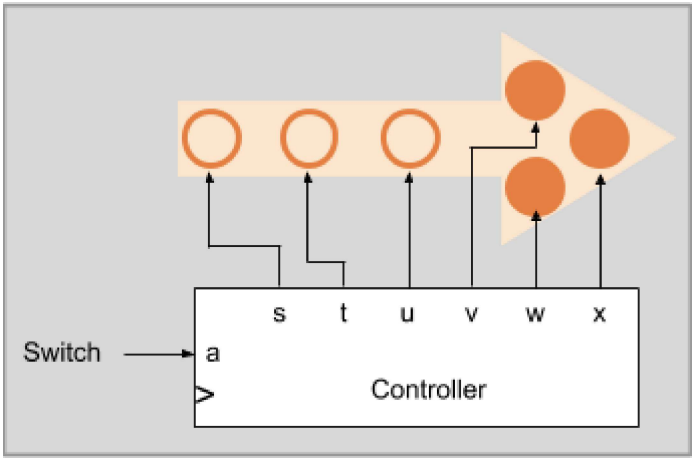
2) Suppose a = 1 and the FSM is in state

n, and then a changes to 0. What is the FSM's next state?

- m
- p

PARTICIPATION ACTIVITY 3.8.4: Move-over highway sign.

The move-over highway sign will be extended to replace the single rightmost LED with an arrowhead consisting of three LEDs, v, w, x. The fourth state should light all three LEDs. Match the output assignments with the state.



stuvwx = 000111 **stuvwx = 010000** **stuvwx = 001000** **stuvwx = 100000**

	State m
	State n
	State p
	State q

Reset

Example: Garage door opener

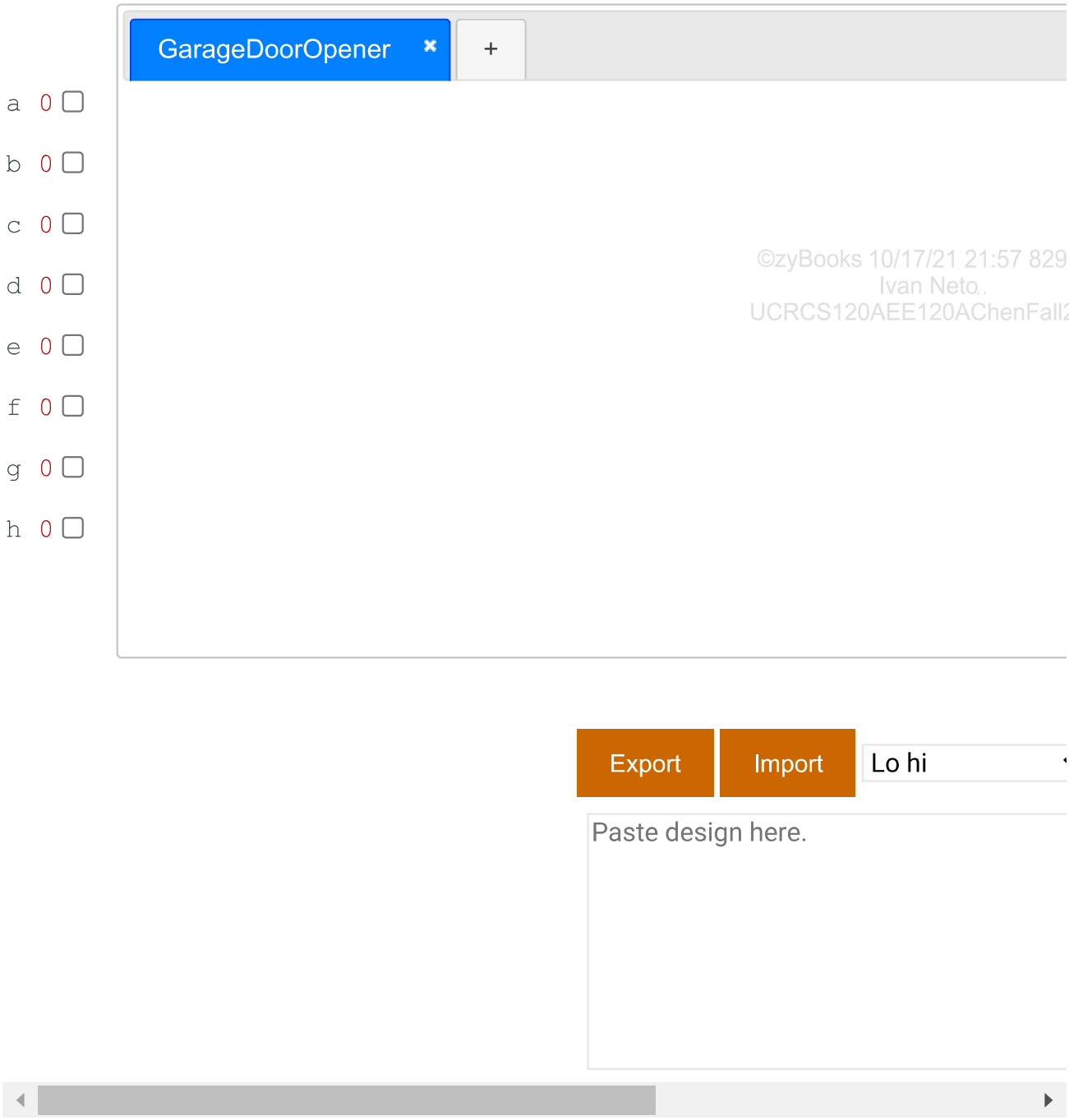
A garage door opener has a single button causing a toggle: For a press, the door toggles from closed to open, or from open to closed. For this example, pressing the button causes a = 1. The door's output s = 1 raises the door until input b = 1 indicates fully-open, and output t = 1 lowers the door until input c = 1 indicates fully-closed.

PARTICIPATION ACTIVITY 3.8.5: FSM example: Sequence generator.

Full screen

Press "Simulate". Set a = 1 and then back to 0 (pressing the button and releasing) and watch the output s = 1 open the door until input b = 1 (you set b = 1 to simulate the door reaching the opened position). Then set a = 1 and back to 0 again, and watch output t = 1 close the door until (you set) input c = 1.

Simulate Pause Insert state Insert transition



PARTICIPATION ACTIVITY 3.8.6: Garage door opener.

Consider the example above.

1) In state Closed, pressing the button (a = 1) causes the FSM to transition to state _____.
   - ○ Closing
   - ○ Opening

2) Normally, a person using a real garage

door would set b = 1 to indicate the door is fully open.

- ○ True
- ○ False

3) When in state Open, can any input values cause the FSM to transition to state Opening?

- ○ Yes
- ○ No

4) A typical garage door has a sensor to detect if something is under the door, stopping the door from closing. Suppose the input from the sensor is d = 1, meaning something is under the door. What state's output transitions would be modified to include d?

- ○ Opening
- ○ Closing

Example: Level to pulse converter

A ***level to pulse converter*** converts a longer-than-one-cycle input pulse to a one-cycle output pulse. When that output is then input to other systems, a one-cycle pulse can simplify those systems' FSMs.

PARTICIPATION ACTIVITY 3.8.7: Level to pulse converter.

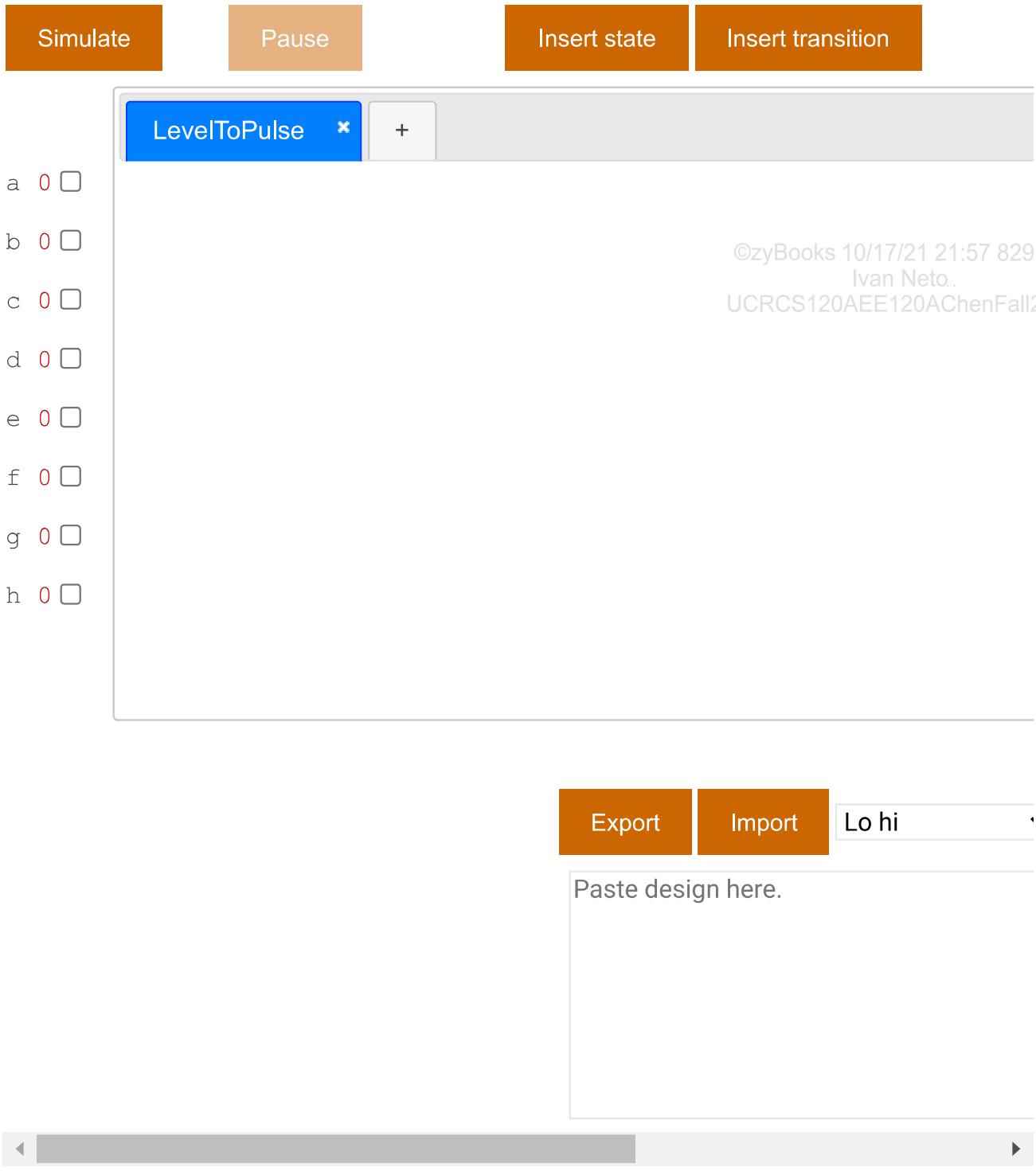
Animation captions:

1. A button press may last multiple clock cycles.
2. A level to pulse converter outputs a one-cycle pulse no matter how long the input pulse is.

PARTICIPATION ACTIVITY 3.8.8: FSM example: Level to pulse converter.

Full screen

Press "Simulate". Set a = 1, and notice a one-cycle output pulse on s. Set a = 0 again. Repeat.



PARTICIPATION ACTIVITY 3.8.9: Level to pulse converter.

Consider the example above.

1) The FSM begins executing, and a = 0. What is the current state?

Check Show answer

2) The FSM begins executing, and a = 0. What is the value on output s?

Check Show answer

3) The FSM is in WaitRise, and then input a changes from 0 to 1. What is the next state?

Check Show answer

4) Upon entering state Pulse, for how many clock cycles will s = 1?

Check Show answer

5) If a changes from 0 to 1 and then stays 1 for 10 clock cycles, s will become 1 for one clock cycle, then s will be 0. During those 10 clock cycles, how many more times does s become 1?

Check Show answer

Example: Button debouncer

A ***button debouncer*** outputs a single pulse when a button is pressed, ignoring the multiple small pulses (bounces) that may occur due to a button's mechanical imperfections.

PARTICIPATION ACTIVITY 3.8.10: A button debouncer outputs a single pulse when a button is pressed.

Animation captions:

1. Mechanical buttons experience "bounce": A single press initially yields 1's and 0's before reaching a steady 1.
2. An FSM might thus see one press as two presses.
3. A debouncer circuit filters out the bouncing, outputting a steady 1.
4. One debounce approach uses an FSM. After the first 1, the FSM skips the input for the known bounce time, in this case one additional cycle, holding the output at 1.
5. When input a first rises to 1, a state sets s with 1. The input is then skipped for one cycle (the known bounce time), and then the output stays 1 with a after then.

PARTICIPATION ACTIVITY 3.8.11: Button debouncer.

Consider the example above.

1) Pressing a button may cause the output to rapidly change between 1 and 0 for a short time.
   - True
   - False

2) The button had a bounce time of about ____ cycle.
   - 1
   - 3

3) The skip-one-cycle approach works for any button and any clock period.
   - True
   - False

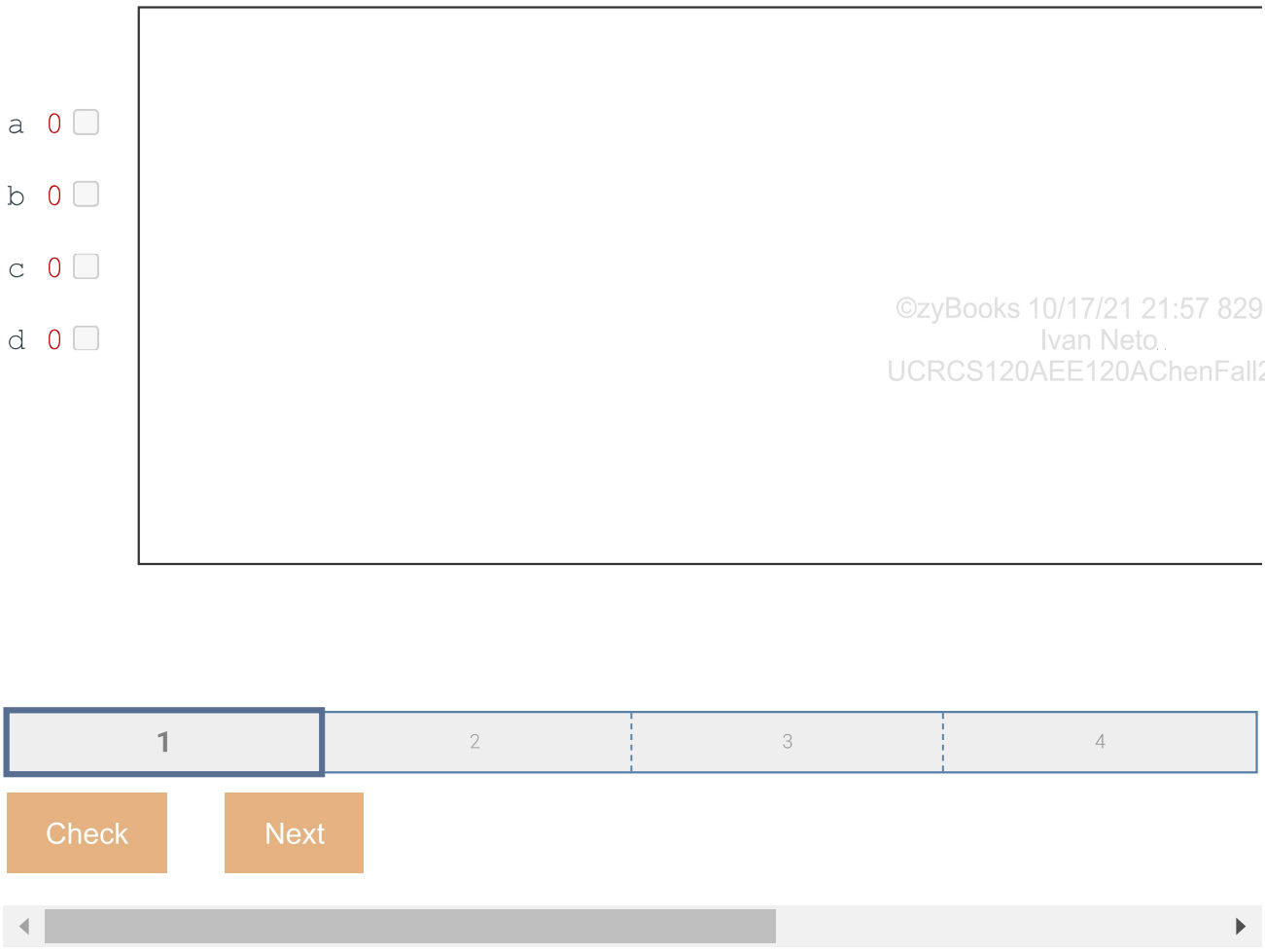
CHALLENGE ACTIVITY 3.8.1: Capturing behavior with an FSM. Full screen

Press "Start" to begin. Capture the indicated behavior with an FSM. Press "Check" to verify.

347136.1658324.qx3zqy7

Start

Simulate Pause Insert state Insert transition



3.9 FSMs to circuits (design)

Controllers

Earlier sections showed that an equation can be converted to a combinational circuit, consisting of logic gates.

Similarly, an FSM can be converted to a sequential circuit known as a ***controller***, consisting of a register and combinational logic.

A ***state register*** is the register in a controller, holding an FSM's present state. Each state requires a unique bit encoding, which is then stored in the state register. Three or four states require 2 bits (00, 01, 10, 11). Five, six, seven, or eight states require 3 bits (000, 001, ..., 111). N states require $\log_2 N$ bits.

A controller's combinational logic computes: (1) the FSM's outputs, based on the present state, and (2) the next state, based on the present state and the FSM's inputs.

To convert an FSM to a controller, a designer first connects the state register with a combinational logic block. Using the FSM, a designer can fill a truth table for the controller's combinational logic,

derive each output's equation, and convert each equation to a combinational circuit, to complete the controller's design.

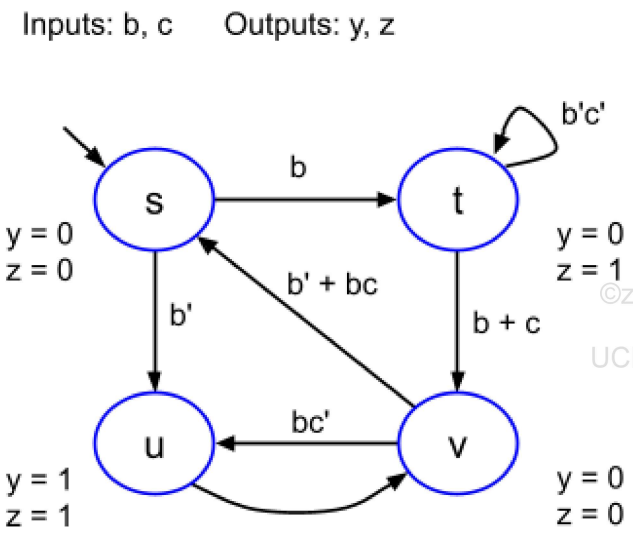
PARTICIPATION ACTIVITY 3.9.1: Converting an FSM to a sequential circuit.

Animation captions:

1. Each state is represented by a unique bit encoding. Two bits are needed to uniquely encode three (or four) states. Here, state s gets 00, t gets 01, u gets 10.
2. The controller has a 2-bit state register, plus combinational logic with FSM input b and output y and with inputs p1 p0 from the state register and n1 n0 to the state register.
3. A table can represent the controller's FSM. Input columns are p1 p0 b, with rows for all eight combinations (000, 001, ...). The first two rows correspond to state s, etc.
4. The table's outputs are n1 n0 y. The designer can first fill in the y column for each state's output action: y = 0 for s's two rows, y = 1 for t's, and y = 1 for u's.
5. Then, the designer can fill in n1 n0 for the appropriate next state. In state s (00), if b is 0, the next state is s (00); if b is 1, the next state is t (01).
6. The designer can then convert each output column to an equation and minimize, yielding n1 = p1'p0, n0 = p1'p0'b, and y = p1'p0 + p1p0'.
7. Finally, the designer can convert each equation to a circuit in the combinational logic. The controller's circuit (state register plus combinational logic) implements the FSM.

Converting behavior (like an FSM) to a circuit (like a controller) is called ***design***. (In contrast, converting a circuit to behavior is called *analysis*).

PARTICIPATION ACTIVITY 3.9.2: Controller basics.



1) The FSM has four states.

True

False

2) When implementing the FSM as a controller, a minimum of three bits are required to encode the states.

True

False

3) A valid encoding of the states is:
s: 11
t: 01
u: 10
v: 00

True

False

4) A valid encoding of the states is:
s: 11
t: 01
u: 11
v: 00

True

False

5) A state register holds the FSM's output values.

True

False

6) In the controller's circuit, the logic's inputs are b and c.

True

False

7) In the controller's circuit, the logic's outputs are y and z.

True

False

8) More FSM transitions likely means more state register bits.

True

False

9) More FSM outputs likely means more logic.

True

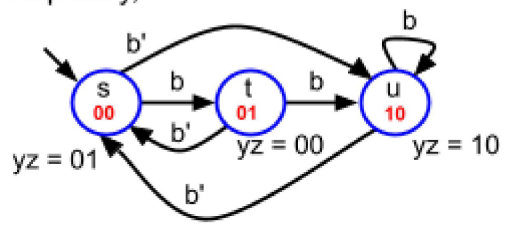
False

Converting an FSM to a controller

As seen above, a designer can convert an FSM to a sequential circuit using a controller consisting of a state register plus combinational logic. The designer first converts to a table, then to equations and finally fills in the combinational circuit.

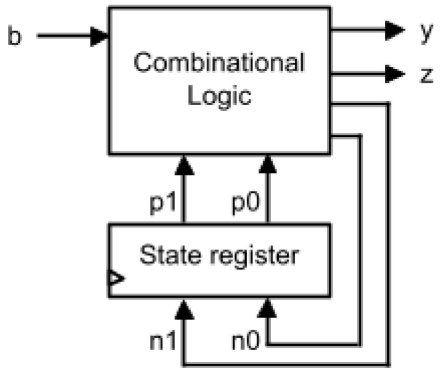
PARTICIPATION ACTIVITY 3.9.3: Converting an FSM to a truth table.

Inputs: b
Outputs: y, z



*State encodings are in red

	p1	p0	b	n1	n0	y	z
s	0	0	0	1	0	0	1
	0	0	1	0	(F)	0	1
t	0	1	0	0	0	(C)	(D)
	0	1	1	1	0	0	(E)
u	1	0	0	(G)	0	1	0
	1	0	1	1	0	1	0
unused	1	1	0	0	0	0	0
	1	1	1	0	0	0	0



Complete the truth table for the given FSM.

1) (C)

1

0

2) (D)

- 1
- 0

3) (E)

- 1
- 0

4) (F)

- 1
- 0

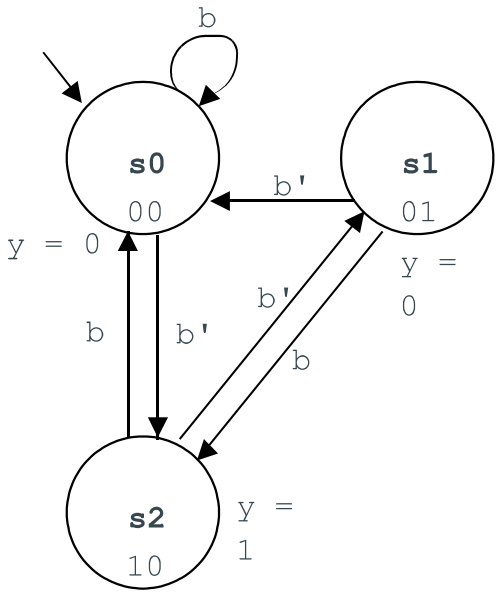
5) (G)

- 1
- 0

CHALLENGE ACTIVITY 3.9.1: Convert the FSM to a truth table.

347136.1658324.qx3zqy7

Start



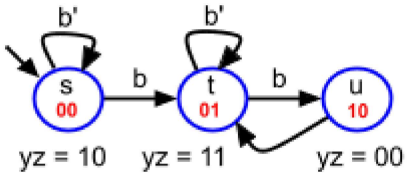
	p1	p0	b	n1	n0	y
s0	0	0	0	1	0	0
	0	0	1	0	0	0
s1	0	1	0	0	0	0
	0	1	1	1	0	0
s2	1	0	0	0	1	1
	1	0	1	0	0	1
Unused	1	1	0	0	0	0
	1	1	1	0	0	0

1 2 3 4 5 6

Check Next

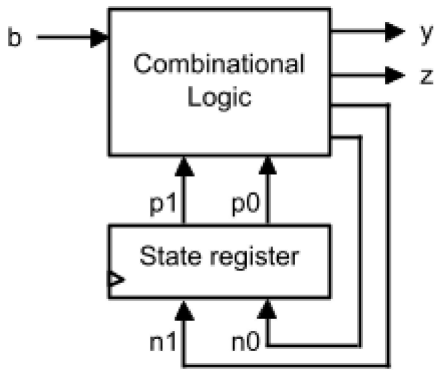
PARTICIPATION ACTIVITY 3.9.4: Converting a table to circuits for a controller's combinational logic.

Inputs: b
Outputs: y, z



*State encodings are in red

	p1	p0	b	n1	n0	y	z
s	0	0	0	0	0	1	0
	0	0	1	0	1	1	0
t	0	1	0	0	1	1	1
	0	1	1	1	0	1	1
u	1	0	0	0	1	0	0
	1	0	1	0	1	0	0
unused	1	1	0	0	0	0	0
	1	1	1	0	0	0	0



Type only the ? part. Use given variable orderings in terms.

1) n1 = ?

Check Show answer

2) n0 = p1'p0'b + p1'p0b' + ? (type one term)

Check Show answer

3) y = ?

Check Show answer

4) z = ?

Check Show answer

3.10 Example: Laser surgery system using an FSM

Three-cycle-high laser timer FSM

Consider the design of a controller used within a laser surgery system for scar removal or corrective vision. Such systems work by turning on the laser for a precise amount of time when a surgeon presses a button. Assume the laser has to stay on for exactly 30 ns. If the system's clock period is 10 ns, the laser needs to be on for 3 clock cycles each time the button is pressed.

The three-cycles-high laser timer system has an input b and output x. The input b connects to a button that is synchronized with the clock and stays high for exactly one clock cycle. The output x turns on the laser when x = 1. When the three-cycles-high laser timer detects b = 1, the controller will set x = 1 for exactly 3 clock cycles, thus turning on the laser for 30 ns.

PARTICIPATION ACTIVITY 3.10.1: Three-cycles-high laser timer FSM.

Animation content:

undefined

Animation captions:

1. In a user surgery system, each time a surgeon presses a button, a controller will turn on the laser for exactly three clock cycles.
2. The FSM starts in the Off state with output x = 0. When b = 1, the FSM transitions to On1, On2, On3 states, on every rising clock edge. In On1, On2, and On3, output x = 1.
3. The timing diagram shows that when b = 1, x = 1 for exactly three clock cycles.

PARTICIPATION ACTIVITY 3.10.2: Three-cycles-high laser timer.

1) If the button b was pressed for two consecutive clock cycles, the output x is 1 for how many clock cycles?

- 3
- 6

2) If the button b was pressed for 8 clock cycles, the output x is 1 for how many total clock cycles?

- 3
- 6

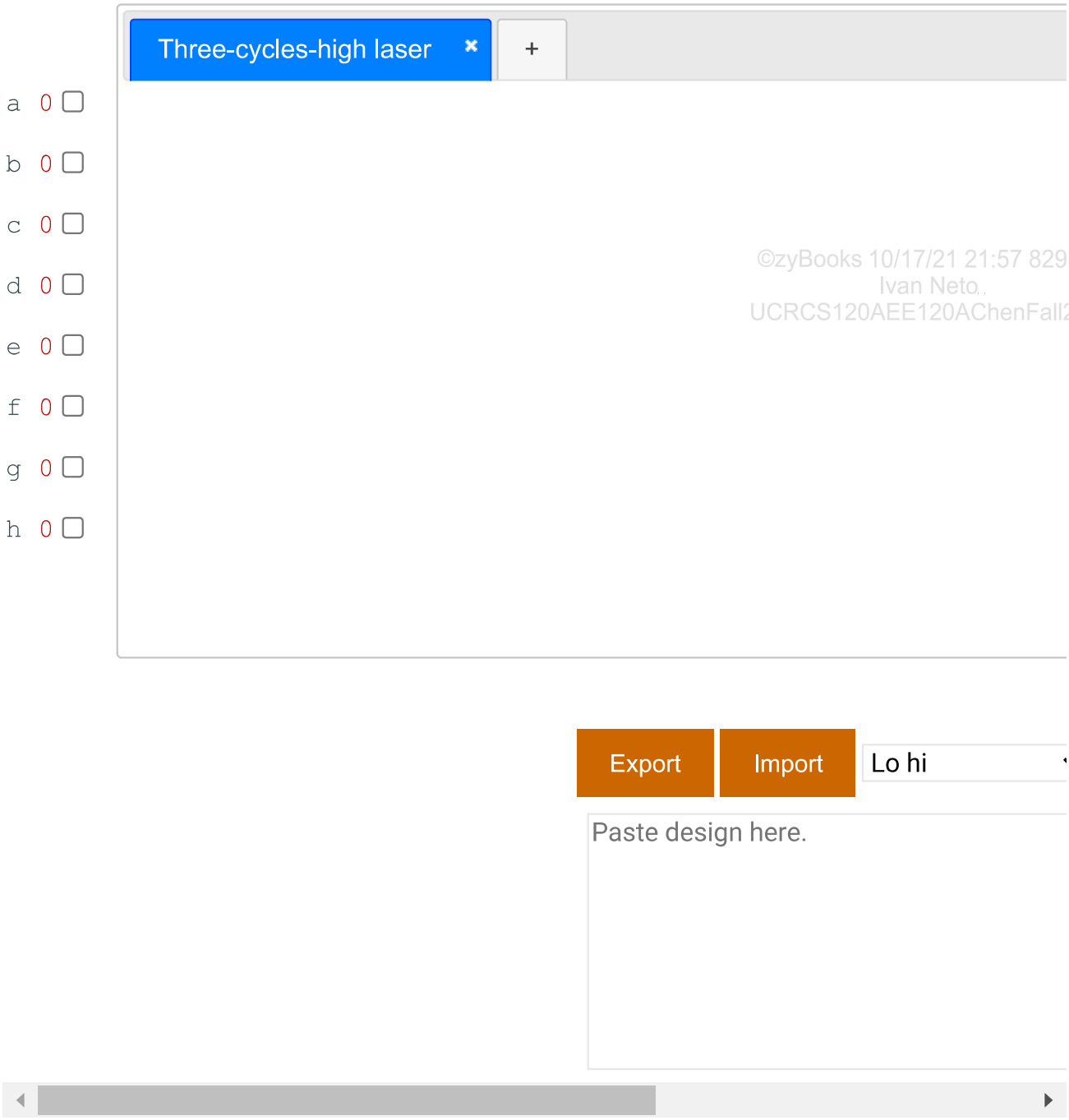
3) In state On2, if the button is not pressed (b = 0), the FSM stays in On2.

- True
- False

PARTICIPATION ACTIVITY 3.10.3: FSM example: Three-cycles-high laser timer. Full screen

Press "Simulate". Set b = 1, and notice a three-cycle output pulse on x. Set b = 0 again. Repeat.

Simulate | Pause | Insert state | Insert transition



Three-cycles-high laser timer controller

The three-cycles-high laser time FSM can be converted to a controller using a state register and combination logic.

PARTICIPATION ACTIVITY 3.10.4: Three-cycles-high laser timer controller.

Animation content:

undefined

Animation captions:

1. The FSM captured has four states and used two state bits. The inputs to the FSM are b, s1, and s0, and the outputs are x, n1, and n0.
2. In state 00, x = 0. If b = 0, the controller stays in state 00 or Off state. If b = 1, the controller goes to On1 state, so n1n0 = 01.
3. In state 01, x = 1 for b = 0 or 1 and the next state is 10. In state 10, x = 1 for b = 0 or 1 and the next state is 11. In state 11, x = 1 for b = 0 or 1 and the next state is 00.
4. From the table, the combinational logic can be implemented by deriving equations for x, n1 and n0.
5. The final implementation of the three-cycles-high laser controller is shown with a sequential circuit.

PARTICIPATION ACTIVITY 3.10.5: Three-cycles-high laser controller.

1) When s1s0 = 10, what is n1n0?

Check Show answer

2) When s1s0 = 00 and b = 1, what is n1n0?

Check Show answer

3.11 Example: Secure car key

Secure car key

Car keys are embedded with a computer chip that implements a secure key. When the key is in the vehicle and the ignition is pressed, the car's computer sends a radio signal to the key asking for an identifier. The chip in the key responds by sending an identifier (ID). If the car's computer does not receive a response, or if the ID is different than the expected ID, the car will not start. For this example, consider a simple ID of 1011 (real IDs are typically 32-bits or more). The key sends the ID serially, with one bit sent each clock cycle.

PARTICIPATION ACTIVITY 3.11.1: Secure car key system.

Animation content:

undefined

Animation captions:

1. The car's computer sends a radio signal to the secure key asking for the key's ID. The secure key responds by sending the ID.
2. The secure car key can be implemented using an FSM. The input a is the request from the car computer for the ID. The FSM remains in the Wait state if a = 0.
3. When input a = 1, the FSM enters K1 and outputs s = 1. Then, at every clock rising clock edge, the FSM proceeds to K2, K3, and K4.
4. The timing diagram shows when input a = 1, the FSM enters K1 and outputs s = 1. The FSM then proceeds through K2, K3, and K4, outputting the other bits of the ID each clock cycle.

PARTICIPATION ACTIVITY 3.11.2: Secure car key example.

Consider the animation above.

1) If the ID was 32 bits, how many states will the FSM require?
   - 2^{32}
   - 5
   - 33

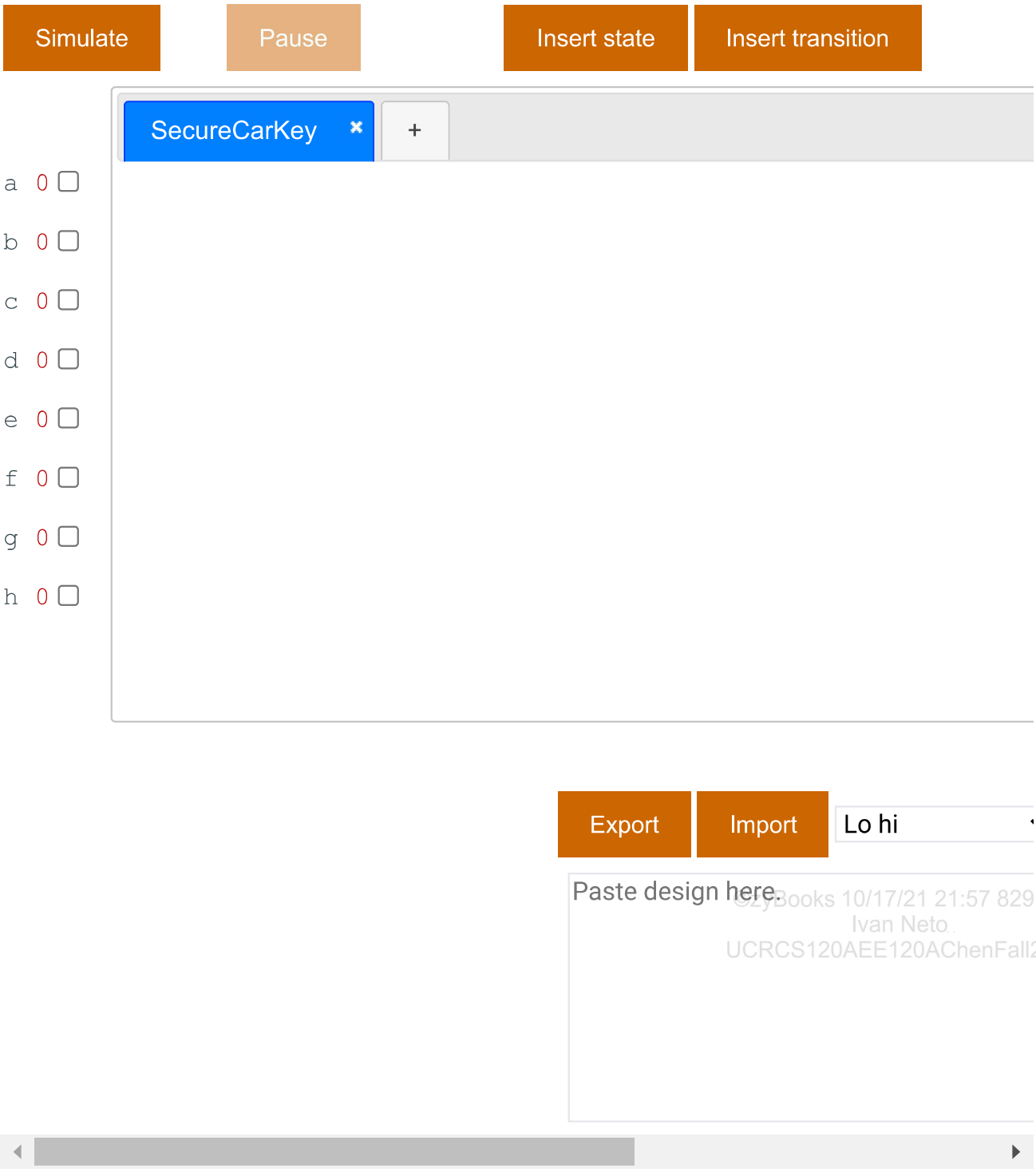
2) What happens if a = 1 for 10 clock cycles?
   - The FSM transmits the 4-bit ID once and waits in the Wait state.
   - An error will occur
   - The FSM transmits the 4-bit ID twice, and then waits in the Wait state.

3) When in K1, if a = 0, what is the next state?
   - Wait
   - K1
   -

K2

PARTICIPATION ACTIVITY 3.11.3: FSM example: Secure car key. Full screen

Press "Simulate". Set a = 1, and notice FSM traverses through K1, K2, K3, K4 and outputs s = 1011. Set a = 0 again. Repeat.



Secure car key controller implementation

The secure car key controller can be converted to a controller using a state register and combination logic.

PARTICIPATION ACTIVITY 3.11.4: Secure car key controller.

Animation content:

undefined

Animation captions:

1. The FSM has 5 states, thus requires a 3-bit state register that can represent 8 states. Three state encodings are unused. A binary encoding is chosen.
2. The three state bits are s2, s1, and s0, and the input to the FSM is a. The output is s, and the next state bits are n2, n1, and n0.
3. The FSM can be converted to a truth table. For the unused states, output s = 0, and next state bits n2, n1, n0 = 0.
4. The three state encodings 101, 110, 111 are unused, and the outputs for the unused states are set to 0.

PARTICIPATION ACTIVITY 3.11.5: Secure car key controller.

Consider the animation above.

1) For a 256-bit ID, how many states will the FSM require if a binary encoding is assumed?

Check Show answer

2) For a 32-bit ID, how many outputs will the FSM require? (Do not consider the next state bits.)

Check Show answer

3) How many state encodings would

be unused for FSM for a 16-bit ID using 5 bits?

Check Show answer

3.12 Reducing states

Equivalent states

Two states that have the same output, and the same next state for the same input values, are ***equivalent states***. One of the two equivalent states can be removed, which may reduce the size of a controller's state register and/or logic.

PARTICIPATION ACTIVITY 3.12.1: Equivalent state basics.

Animation captions:

1. States with the same output values and same next state are equivalent. Here, states s and u both output 0.
2. And, states s and u both always transition to t.
3. Thus, states s and u are equivalent. A designer can replace state s by u, thus reducing the FSM from three states to two states.

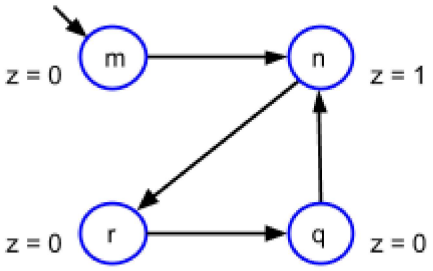
PARTICIPATION ACTIVITY 3.12.2: Equivalent states when input values are involved.

Animation captions:

1. To find equivalent states, a designer notices states s and u both output 0.
2. The designer must check if states s and u have the same next state for each possible input value. If b is 0, states s and u both transition back to themselves.
3. If b is 1, states s and u both transition to t.
4. Thus, states s and u are equivalent. The designer can replace s by u.

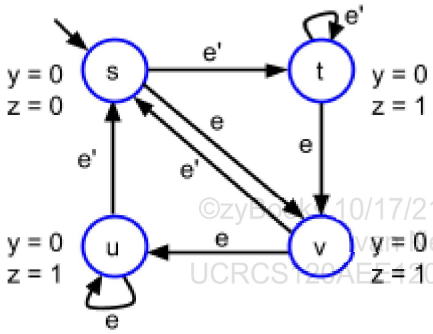
PARTICIPATION ACTIVITY 3.12.3: Equivalent states.

Inputs: none
Outputs: z



(a)

Inputs: e
Outputs: y, z



(b)

1) States m and n.

- Equivalent
- Not equivalent

2) States m and q.

- Equivalent
- Not equivalent

3) States r and q.

- Equivalent
- Not equivalent

4) States u and v.

- Equivalent
- Not equivalent

5) States s and t.

- Equivalent
- Not equivalent

Partitioning method to reduce states

Given two equivalent states, either state can be removed. That state's outgoing transitions can also be removed, since the same transitions must also exist on the other state. Any incoming transitions are simply repointed to the other state.

A designer can apply a ***partitioning method*** to find and eliminate equivalent states. The method first groups states per output values; states with different output values cannot be equivalent. Then, for

each state in a group, the method checks next states' groups; if different, those states cannot be equivalent and the group is split. The method continues checking every group, until no group was split.

PARTICIPATION ACTIVITY 3.12.4: Partitioning method to reduce states, for an FSM with no inputs.

Animation captions:

1. Initially, group states by output values.
2. List next states and their groups.
3. If next states' groups differ, can NOT be equivalent.
4. Split group per next states' groups. Update next states' groups.
5. Merge equivalent states.

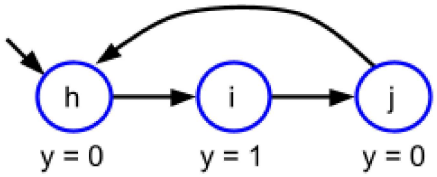
PARTICIPATION ACTIVITY 3.12.5: Partitioning method to reduce states, for an FSM with inputs.

Animation captions:

1. Initially, group states by output values.
2. Add a column for inputs.
3. List next states and their groups based on the transition conditions.
4. Finish listing next states and their groups based on the transition conditions.
5. If next states' groups differ, can NOT be equivalent.
6. Split group per next states' groups. Update next states' groups.
7. If next states' groups differ, can NOT be equivalent.
8. Merge equivalent states.

PARTICIPATION ACTIVITY 3.12.6: Reducing states with the partitioning method (with no inputs).

Inputs: none
Outputs: y



G1 y = 0

State	Next state: Group
(M)	i : (P)
j	(Q) : (R)

G2 y = 1

State	Next state: Group
(N)	j : G1

1) (M)

- h
- i

2) (N)

- h
- i

3) (P)

- G1
- G2

4) (Q)

- h
- i

5) (R)

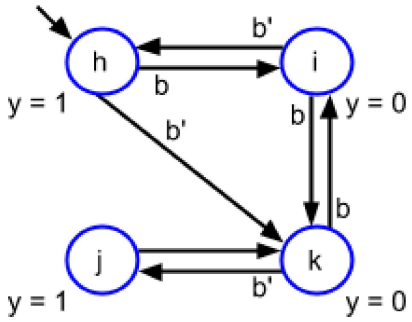
- G1
- G2

6) States h and j are equivalent.

- True
- False

PARTICIPATION ACTIVITY 3.12.7: Reducing states with the partitioning method (with inputs).

Inputs: b
Outputs: y



	State	Inputs b	Next state: Group
G1 y = 0	i	0	h : G2
	k	0	j : G2
	i	1	(M) : G1
	k	1	(N) : (P)

	State	Inputs b	Next state: Group
G2 y = 1	h	0	k : G1
	j	0	(Q) : (R)
	h	1	i : G1
	j	1	k : G1

1) (M)

- h
- k

2) (N)

- i
- j

3) (P)

- G1
- G2

4) (Q)

- k
- j

5) (R)

- G1
- G2

6) States i and k are equivalent.

- True
- False

7) States h and j are equivalent.

- ○ True
- ○ False

3.13 State encodings

Binary state encodings

A ***binary encoding*** uses the fewest bits possible for a given number of states. Ex: 2 bits for 4 states, or 3 bits for 8 states.

An enormous number of encodings are possible. For just 4 states and 2 bits, 4 · 3 · 2 · 1 = 24 possible encodings exist: first state may be 00, 01, 10, 11 so has 4 choices, second state has 3 choices, third 2, and fourth 1. 8 states has 8! or 40,320 possible encodings.

Designers commonly wish to reduce the combinational logic's size. A designer might implement a few encodings to see which yields smaller size. Below, size estimates assume 6 transistors for a 2-input AND or OR gate, and 2 transistors for a NOT gate.

An encoding that reduces the changing bits during transitions may increase i(j + j') opportunities, yielding fewer gates.

PARTICIPATION ACTIVITY 3.13.1: Different state encodings may reduce gates.

Animation captions:

1. A straightforward first encoding.
2. Logic block's circuit equations for this encoding.
3. A second encoding, and the circuit equations.
4. The second encoding reduces circuit size.

PARTICIPATION ACTIVITY 3.13.2: State encodings.

1) A designer should try all possible encodings of states, to find the encoding with fewest transistors.

- ○ True
- ○ False

2) State m transitions to state n. If state m's encoding is 011, which is likely a better state encoding for state n?
   - 100
   - 111

3) The transistor count for n1 = p0 + p1' is 6.
   - True
   - False

4) The transistor count for n0 = p0b + p1'p0' is 22.
   - True
   - False

5) The transistor count for z = p0 is 0.
   - True
   - False

One-hot state encodings

A ***one-hot encoding*** uses N bits for N states, with each encoding having exactly one bit set to 1. A one-hot encoding uses more bits and thus a larger state register than a binary encoding, but may yield fewer gates, especially for relatively few states.

PARTICIPATION ACTIVITY 3.13.3: One-hot state encoding.

Animation captions:

1. 4 states means 4 bits in the encoding.
2. Logic block equations for this encoding.
3. One-hot reduces logic-block transistors vs. earlier binary encodings (though state register is larger).

Above, a shortcut was used to derive the circuit equations, namely recognizing that not-shown input combinations are invalid ("don't care minterms") so can be set to output 1's as desired, causing other variables to be minimized away via i(j + j') opportunities.

PARTICIPATION ACTIVITY 3.13.4: One-hot encoding.

Consider the example above.

1) How many states does the FSM have?

Check Show answer

2) How many bits are required for a one-hot encoding of the four-state FSM?

Check Show answer

3) Is a state encoding of 1100 valid for a one-hot encoding? Type: yes or no

Check Show answer

4) How many gates are needed to implement the combinational logic block?

Check Show answer

5) Assume a flip-flop has 10 gates. How many gates are needed to implement the state register?

Check Show answer

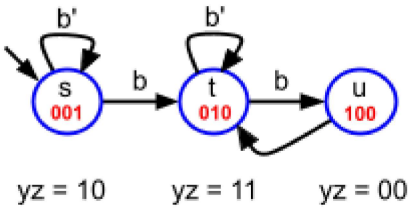
6) Does one-hot encoding always yield the fewest gates? Type: yes or no

Check Show answer

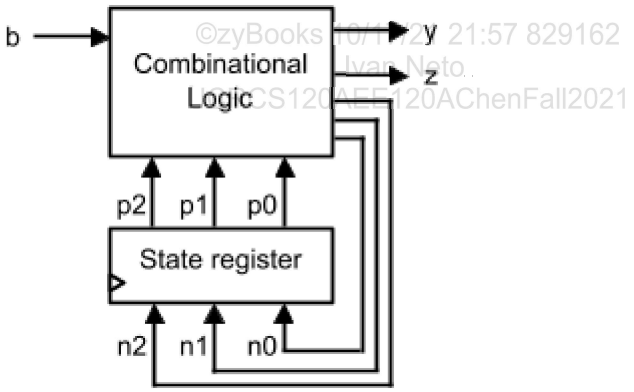
Check Show answer

PARTICIPATION ACTIVITY 3.13.5: One-hot encoding: Example.

Inputs: b
Outputs: y, z



*State encodings are in red



	p2	p1	p0	b	n2	n1	n0	y	z
s	0	0	1	0	0	0	1	(P)	0
	0	0	1	1	0	(Q)	0	1	0
t	0	1	0	0		(R)		1	1
	0	1	0	1	1	0	0	1	1
u	1	0	0	1	0	1	0	0	0
	1	0	0	0	0	1	0	0	0

1) (P)

Check Show answer

2) (Q)

Check Show answer

3) (R). Provide answer as: ???

Check Show answer

4) y = p2'p1'p0 + ? Type only the ? part.

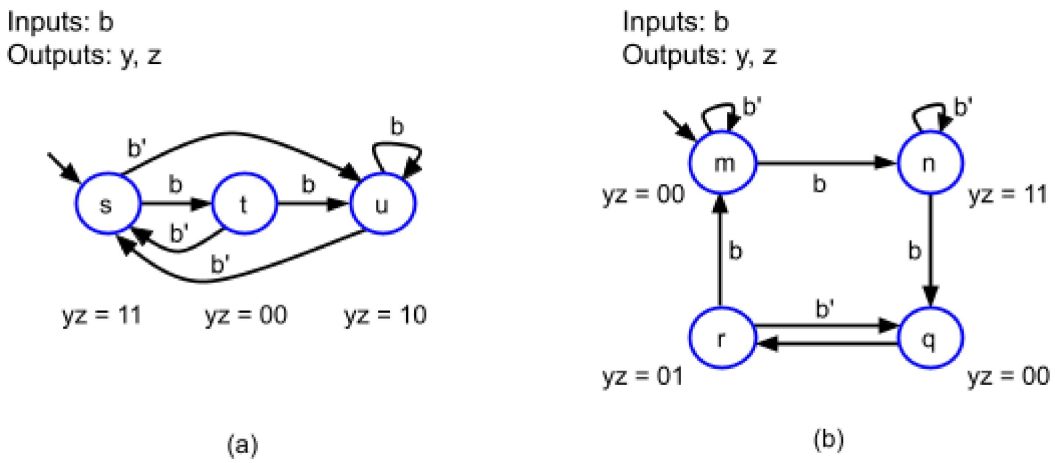
Check Show answer

Output encoding

If each state has a unique output value combination, then those output values may also be used for the state encoding, an approach known as ***output encoding***. Ex: If the above FSM's state s sets yz = 10, t sets yz =00, u sets yz = 11, and v sets yz = 01, then an output encoding is s: 10, t: 00, u: 11, v: 01.

Output encoding may reduce gates, since the same gates in the controller's combinational logic can generate the external outputs and the next state outputs.

PARTICIPATION ACTIVITY 3.13.6: Output encoding.



1) Use output encoding for FSM (a). What is state u's encoding?

- 10
- 01
- 00

2) Use output encoding for FSM (a). What is state t's encoding?

- 10
- 01
- 00

3) Use output encoding for FSM (b). What is state q's encoding?

- 00
-

11

- Output encoding is not possible.

4) In what part of a controller might output encoding reduce transistors?

- State register
- Logic block
- Output encoding cannot reduce transistors.

3.14 Mealy FSMs

A ***Moore*** action appears in an FSM's state, as in the FSMs seen in earlier sections. In contrast, a ***Mealy*** action appears on an FSM's transition. In other words, a Moore action's output value is a function of the present state, while a Mealy action's output value is a function of the present state and present input values.

PARTICIPATION ACTIVITY 3.14.1: Mealy actions in an FSM.

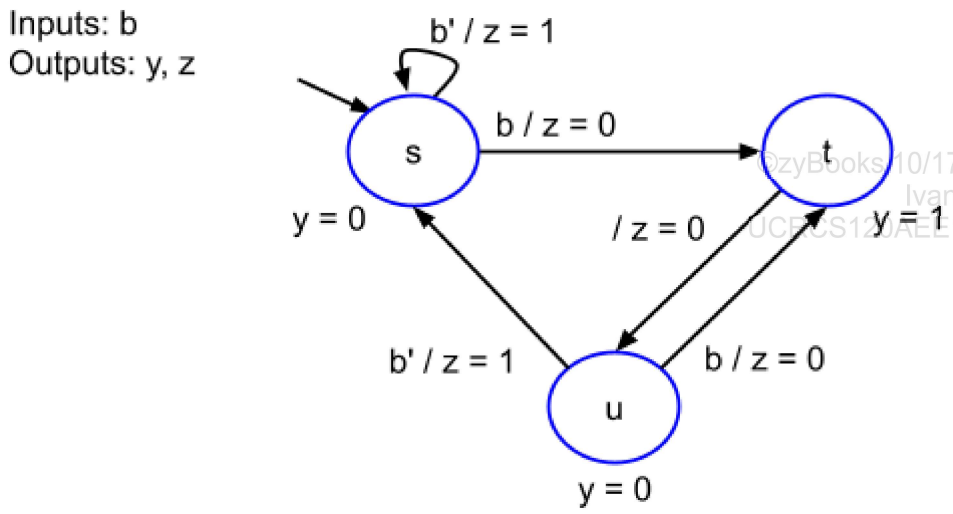
Animation captions:

1. FSM toggles between outputting x = 0 and x = 1 (perhaps turning a lamp off and on). Pressing button a goes to the x = 1 state. Pressing b returns to the x = 0 state.
2. When a button is pressed that will cause a toggle, the FSM outputs y = 1 (perhaps illuminating an LED) using Mealy actions.
3. When in state s, x = 0. Also, if a', then y = 0.
4. y's value changes in immediate response to input a.
5. Present state determines x's value. Present state and present inputs determine y's value.
6. y's value changes in immediate response to input b.

In general, Mealy actions may have the benefit of capturing some behaviors using fewer states, and of having output values change immediately in response to input value changes. However, Mealy actions also have the drawback of possible glitches on outputs, of no guarantee that an output value will last at least one clock cycle, and potential timing problems if multiple circuits FSMs are connected. A solution to these problems is to place a flip-flop at each output, though that delays output updates by a clock cycle. This material does not emphasize Mealy actions.

Note: Most textbooks refer to Mealy *FSMs* (rather than Mealy actions), where all actions are on transitions. Likewise, a Moore FSM has all actions in states.

PARTICIPATION ACTIVITY 3.14.2: FSM with Mealy and Moore actions.



1) Before the first rising clock, so in the initial state, what is y?
   - 0
   - 1

2) Before the first rising clock, so in the initial state, what is z?
   - 0
   - 1
   - Need b's present value to determine z's value.

3) When is the transition " / z = 0" from t to u taken?
   - Rising edge of clk.
   - Only when b is 0 and rising edge of clk.
   - Only when b is 1 and rising edge of clk.

4) An output value of 1 on z is guaranteed to last at least one clock cycle.
   - True
   - False

5) An output value of 1 on y is guaranteed

to last at least one clock cycle.

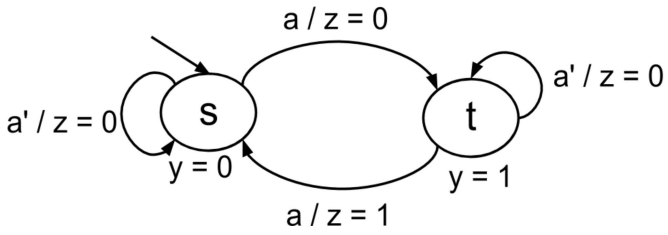
- True
- False

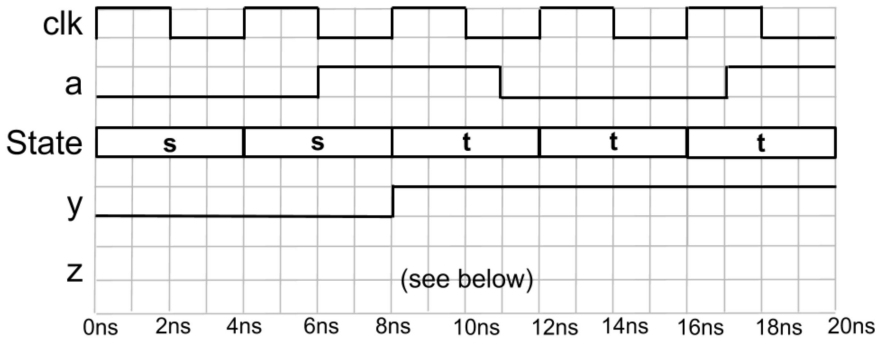
CHALLENGE ACTIVITY 3.14.1: Mealy FSMs.

347136.1658324.qx3zqy7

Start

Input: a
Outputs: y, z





Enter the time (in nanoseconds) at which z changes from 0 to 1:

Ex: 9 ns

1 2

Check Next

3.15 FSM issues

Exactly one true-condition leaving a state

For a given FSM state, *exactly* one outgoing transition should have a true condition at any given time: No more, no fewer.

PARTICIPATION ACTIVITY 3.15.1: FSM transitions: Exactly one condition true.

Animation captions:

1. When in s: Exactly one of a, a' will be true at any time.
2. When in t: More than one of a, b could be true. Which should be taken?
3. When in t: Fewer than one of a, b could be true.
4. Make t's transitions such that exactly one is true at any time.

PARTICIPATION ACTIVITY 3.15.2: FSM outgoing transitions.

Assume an FSM has inputs a and b.

1) State s has two outgoing transitions, one with condition ab, another with a'b. The transitions are correctly formed.
   - ○ True
   - ○ False

2) State s has two outgoing transitions, one with condition a, another with b. The transitions are correctly formed.
   - ○ True
   - ○ False

3) State s has two outgoing transitions, one with condition ab, another with (ab)'. The transitions are correctly formed.
   - ○ True
   - ○ False

4) State s has three outgoing transitions, one with condition ab, another with a'b',

and a third with a'b. The transitions are correctly formed.

- True
- False

Implicit output assignment

Typing an assignment for every output in every state can be cumbersome and distracting. A common shorthand is: If an output is not explicitly assigned in a state, the output is implicitly assigned with 0.

Sometimes a designer still explicitly assigns an output with 0 for clarity.

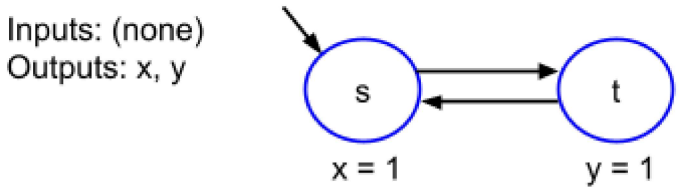
PARTICIPATION ACTIVITY 3.15.3: FSM outputs are implicitly assigned with 0.

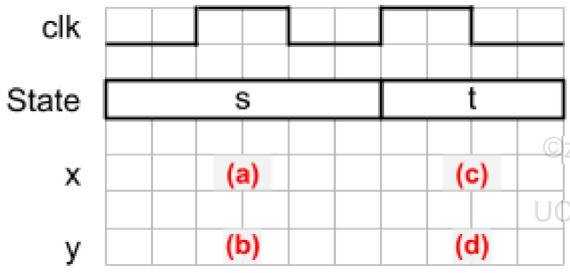
Animation captions:

1. Explicitly assigning every output can be cumbersome and distracting.
2. If designer omits outputs, those outputs are implicitly assigned with 0. Designer still showed x = 0 for clarity.

PARTICIPATION ACTIVITY 3.15.4: Implicit FSM output assignments.

Complete the timing diagram.





1) (a)

- 0
- 1

2) (b)

- 0
- 1

3) (c)

- 0
- 1

4) (d)

- 0
- 1

FSM startup

One may wonder how an FSM circuit is started up in an initial state. Assuming an initial state's encoding is all 0's, then the circuit merely needs to start with 0's in the state register. Registers typically have a clear input, so the challenge is to set the clear input to 1 during startup. Systems typically have electrical means for setting a signal to 1 for a few microseconds during startup. See for example: Power-on reset (Wikipedia).

The clear input may be synchronous (clearing on a clock edge), as discussed in other sections. Or, the input may be asynchronous (clearing independently of the clock), achieved via special flip-flop design. The pros/cons are not discussed here.

If the initial state has different encoding than all 0's, the state register's flip-flops may be initialized using reset and set inputs that may be present on certain flip-flops.

PARTICIPATION ACTIVITY 3.15.5: FSM initial state.

1) If an FSM's initial state s is encoded as 111, then the FSM circuit's state register should upon startup be loaded with ???.

- 000
- 111

2) An FSM's initial state s is encoded as 000, so the FSM's circuitry sets the state register's clear input to 1 for a few microseconds upon startup. Which is a typical method?

- Special circuitry sets the register's clear signal to 1 upon startup.
- The initial state should have the action "clear = 1".
- The initial state should have a transition back to itself.

3.16 Controller clock frequency

A controller's clock frequency cannot be arbitrarily fast. After a rising clock edge loads the state register, time is needed for the register's new output values to pass through the gates, thus generating new input values to the state register. The next rising clock edge should not occur before that time, otherwise strange values may be loaded.

PARTICIPATION ACTIVITY 3.16.1: Controller clock frequency.

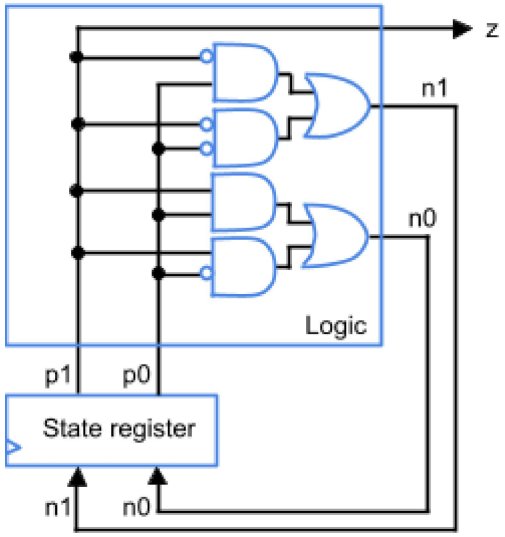
Animation captions:

1. After the state register is loaded, new values take time to pass through the gates.
2. If the clock frequency is too fast, a wrong state value may be loaded.
3. A sufficiently slow clock is necessary to allow new input values to arrive.

The clock's period must be longer than the longest delay from the state register's output to the state register's input, known as the ***critical path***. If each gate has delay of 1 ns (and ignoring delays of inverters and wires for simplicity), the above controller circuit has a critical path of 1 ns + 1 ns = 2 ns. Thus, the clock's period should be longer than 2 ns, meaning the clock's frequency should be slower than (1 / 2 ns) = 500 MHz. ***Clock frequency*** is the inverse of the clock's period: 1 / (clock period).

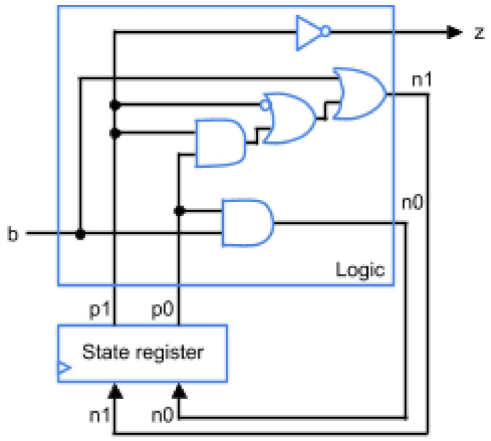
PARTICIPATION ACTIVITY 3.16.2: Critical path.

1) What is the controller's critical path?



- 2 ns
- 3 ns
- 6 ns

2) What is the controller's critical path?



- 1 ns
- 3 ns
- 4 ns

3) What is the fastest clock frequency given a critical path of 5 ns?

- 5 ns
- 5 MHz
- 200 MHz

4) What is the clock frequency given a critical path of 1 ns?

- 1000 MHz
-

100 MHz

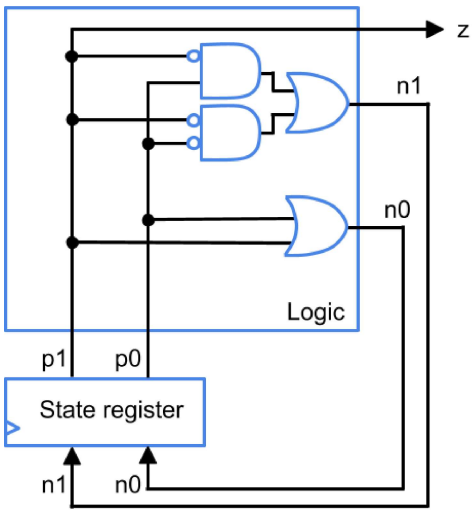
10 MHz

In practice, the critical path should include delays of wires and inverters. The clock period should also be slightly longer to account for flip-flop setup and hold times (not discussed here). And, some additional "safety" time (e.g., 10%-20%) should be added for variations due to manufacturing and temperature.

CHALLENGE ACTIVITY 3.16.1: Controller clock frequency.

347136.1658324.qx3zqy7

Start



Controller's critical path = Ex: 5 ns

Assume 1 ns delay for each gate. Ignore delays of inverters and wires.

1 2 3

Check Next

3.17 Circuits to FSMs (analysis)

Basic sequential circuit analysis

A designer can convert a sequential circuit, consisting of a register and combinational logic, to an FSM. The designer writes the logic as an equivalent truth table. The designer gives a state name to each register-bit combination, lists the output values with each state, and draws each transition. Converting from a circuit to behavior (like an FSM) is called ***analysis***.

PARTICIPATION ACTIVITY 3.17.1: Converting a simple circuit to FSM behavior.

Animation captions:

1. A designer may want to understand the behavior of a circuit. Converting a circuit to behavior is called analysis.
2. The designer first labels the register output (p) and input (n), and then determines equations for the combinational circuit's outputs x and n (x = p, n = p').
3. The designer creates a truth table, with combinational logic inputs on the left and outputs on the right, and fills each output column based on the output's equation.
4. The designer creates a state for each state register bit combination (here just 0 and 1), then assigns the output from the table.
5. The designer also inserts transitions based on the table.
6. The designer then names the states. The designer usually sets the state with a 0 encoding as the initial state.

PARTICIPATION ACTIVITY 3.17.2: Converting a circuit to FSM behavior.

Consider the example above.

1) The initial circuit has a register, making the circuit a _____ circuit.
   - combinational
   - sequential

2) Analyzing a sequential circuit will result in an _____ .
   - equation
   - FSM

3) The designer labeled the register output as p (for "present" state), and input as n (for "next" state). What equation did the designer derive for n?
   - n = p'
   - n = p

4) The designer created a truth table for the combinational logic, whose input was just p. The designer listed all combinations of p, which is just 0 and 1. For p of 0, what is n?
   - 0
   - 1

5) Given the truth table, the designer created a state for each possible state register value. Since the register only had one bit p, the only values for p were 0 and 1, yielding ____ states.
   - 2
   - 4

6) For p of 0, the truth table shows x to be ____ .
   - 0
   - 1

7) For p of 0, the truth table shows n to be 1. Thus, the designer drew a transition from the state encoded as 0 to the state encoded as ____ .
   - 0
   - 1

A larger example

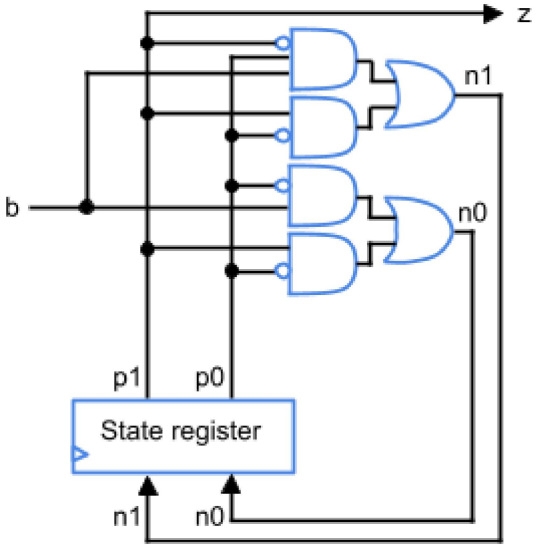
PARTICIPATION ACTIVITY 3.17.3: Converting a circuit to an FSM.

Animation captions:

1. Convert logic into a truth table.
2. Fill in the truth table.
3. Give each register-bit combination a state name.
4. Create corresponding states.
5. List output values for each state.
6. Draw transitions for each state.
7. Draw transitions for each state.

8. Assume the 00 state is the start state.

PARTICIPATION ACTIVITY 3.17.4: Converting a circuit to a truth table.



p1	p0	b	n1	n0	z
0	0	0	0	0	(Q)
0	0	1	0	1	0
0	1	0	(R)	(S)	0
0	1	1	(T)	0	0
1	0	0	1	1	(U)
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

1) (Q)
- 1
- 0

2) (R)
- 1
- 0

3) (S)
- 1
- 0

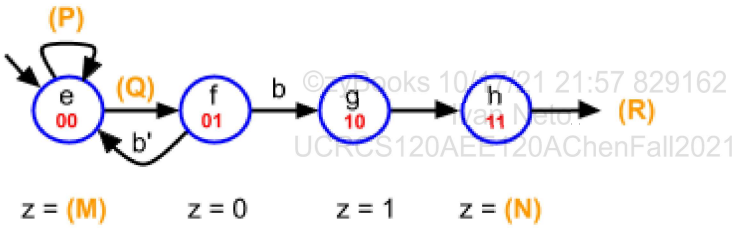
4) (T)
- 1
- 0

5) (U)
- 1
- 0

PARTICIPATION ACTIVITY 3.17.5: Converting a truth table to an FSM.

	p1	p0	b	n1	n0	z
e	0	0	0	0	0	0
	0	0	1	0	1	0
f	0	1	0	0	0	0
	0	1	1	1	0	0
g	1	0	0	1	1	1
	1	0	1	1	1	1
h	1	1	0	0	0	1
	1	1	1	0	0	1

Inputs: b
Outputs: z



*State encodings are in red

1) What is the value (M)?

Check Show answer

2) What is the value (N)?

Check Show answer

3) What is the transition condition (P)?

Check Show answer

4) What is the transition condition (Q)?

Check Show answer

5) What is the next state (R)? Provide answer as: e, f, g, or h

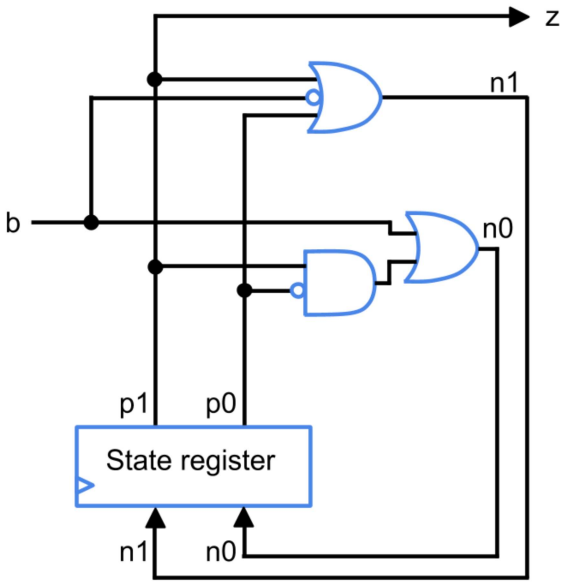
Check Show answer

CHALLENGE ACTIVITY 3.17.1: Circuits to FSMs (analysis).

347136.1658324.qx3zqy7

Start

Given the circuit below, fill in the missing values of the truth table.



p1	p0	b	n1	n0	z
0	0	0			
0	0	1			
0	1	0			(P)
0	1	1			
1	0	0			
1	0	1	(Q)	(R)	(S)
1	1	0			
1	1	1			

(P) = Ex: 0 or 1
(Q) =
(R) =
(S) =

1 2

Check Next

3.18 Non-ideal flip-flop behavior

Setup times and hold times

Wires and logic gates within a flip-flop introduce delays. A flip-flop imposes restrictions on the change in the flip-flop's inputs to ensure correct operations.

- The ***setup time*** is time the input of a flip-flop must be stable before a clock edge.
- The ***hold time*** is time the input of a flip-flop must be stable after a clock edge.

The critical path for a sequential circuit includes the setup time. The clock period must be greater than the sum of the longest combinational path and the setup time.

PARTICIPATION ACTIVITY 3.18.1: Setup time and hold time.

Animation content:

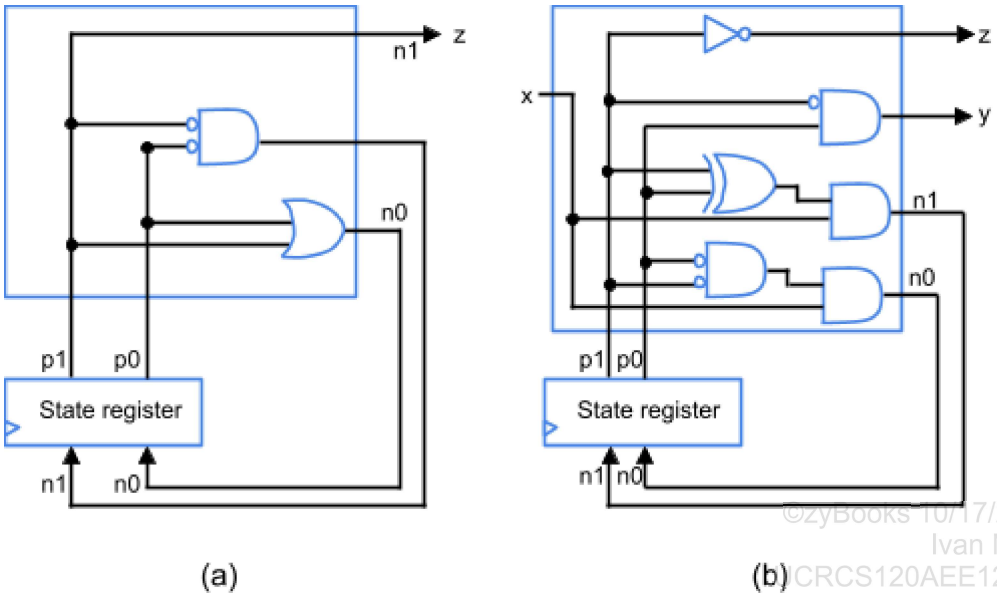
undefined

Animation captions:

1. A flip-flop input (D) must be stable for a minimum amount of time before a clock edge arrives, known as the setup time.
2. A flip-flop input (D) must also remain stable for a minimum amount of time after a clock edge arrives, known as the hold time.

PARTICIPATION ACTIVITY 3.18.2: Setup time and hold time.

Assume each gate delay is 1 ns and a flip-flop's setup time is 0.5 ns. Inverters are ignored.



1) What is the delay of the longest combinational path for (a)?

- 0.5 ns
- 1 ns
-

1.5 ns

2) If the clock period is 4 ns, will the setup time be violated?
   - Yes
   - No

3) If the clock period is 1 ns, will the setup time be violated?
   - Yes
   - No

4) What is the critical path delay for (b)?
   - 0.5 ns
   - 2 ns
   - 2.5 ns

5) If the clock period is 2 ns, will the setup time be violated?
   - Yes
   - No

6) If the clock period is 3 ns, will the setup time be violated?
   - Yes
   - No

Metastability

If a flip-flop's setup and hold time are violated, the flip-flop can enter a metastable state. A ***metastable state*** is in a state other than a stable 0 or a stable 1. A flip-flop in a metastable state can have a voltage that is a value between 0 and 1, and that voltage may oscillate.

PARTICIPATION ACTIVITY 3.18.3: Violation of setup time and hold time.

Animation content:

undefined

Animation captions:

1. D changes too close to the rising clock edge, resulting in a setup time violation
2. u changes to 1 after the inverter delay, and r changes to 1 after the AND gate delay.
3. Almost simultaneously, r changes to 0 as the clock changes to 0.
4. r's brief to change to 1 and back to 0 (called a glitch) causes q to change to 0 and back to 1. The glitch of 0 on q causes a glitch of 1 on q'.
5. The glitch continues around the cross-coupled NOR gates oscillating between a 0 and 1. q will eventually settle to 0 or 1 or cause the flip-flop to enter a metastable state.

PARTICIPATION ACTIVITY 3.18.4: Metastability.

Consider the above animation.

1) If D remains high until after the rising clock edge, the setup time will not be violated.
   - ○ True
   - ○ False

2) The feedback on the cross-coupled NOR gates causes the output to oscillate between 1 and 0.
   - ○ True
   - ○ False

Asynchronous inputs

Metastability is a problem primarily when a flip-flop is connected to asynchronous inputs. An ***asynchronous input*** is an input that is not synchronized with a circuit's clock. Ex: A button input can be pressed at any time by a user, which is not synchronized with the clock. An asynchronous input can change at any time, including during the setup time and hold time, which can lead to metastability.

While controlling the changes of external inputs connected to a circuit is not always possible, a common way to synchronize an asynchronous input is to connect the asynchronous input into a synchronizer flip-flop. The output of the synchronizer flip-flop can then be used as the input to the circuit. A synchronizer flip-flop is typically an extremely fast flip-flop with very small setup and hold times, which minimizes metastability.

PARTICIPATION ACTIVITY 3.18.5: Using synchronizer flip-flops to minimize chance of metastability.

Animation content:

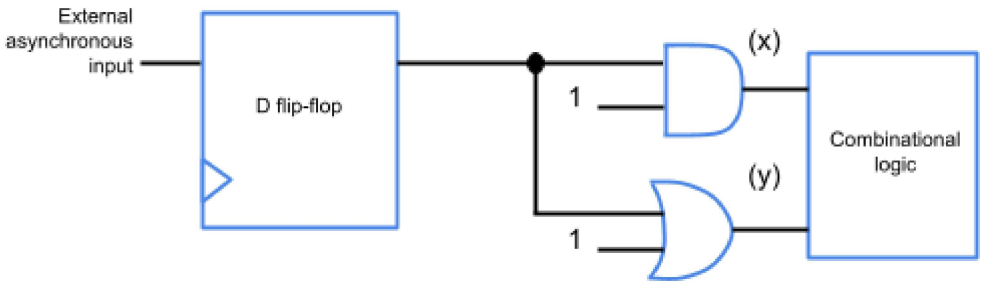
undefined

Animation captions:

1. An asynchronous input arriving at a circuit may not be synchronized with the circuit's clock. Changes for those inputs may violate a flip-flop's setup or hold times causing the circuit to become metastable.
2. Inserting a single synchronizer flip-flop can reduce metastability problems. The synchronizer can experience the same problem, but the problem is now limited to a single flip-flop.
3. A flip-flop will not remain metastable for a long time, and settle into a 1 or a 0. Multiple synchronizers can be added to further reduce the chance of the circuit becoming metastable.

PARTICIPATION ACTIVITY 3.18.6: Metastability.

Assume an external asynchronous input causes the D flip-flop to go into a metastable state.



1) What is the value of (x)?
   - 0
   - 1
   - Unknown

2) What is the value of (y)?
   - 0
   - 1
   - Unknown

3) If a synchronizer is added to the circuit, the chance of the D flip-flop going into metastable state _____.
   - reduces
   - increases

Mean time between failures (MTBF)

Even with multiple flip-flops, metastability is always a possibility. The likelihood of failures due to metastability can be minimized, but not completely eliminated. Designers often rate their designs using a measure called mean time between failures or MTBF. Designers typically aim for MTBFs of many years. Designers of serious high-speed digital circuits should study the problem of metastability, and modern solutions to the problem, thoroughly.

3.19 Product Profile: Pacemaker

Basic pacemaker

A pacemaker provides electrical stimulation to the heart to help regulate the heart's beating, steadying a heart whose body's natural "intrinsic" pacemaker is not working properly, perhaps due to disease. Implantable pacemakers, which are surgically placed under the skin, are worn by over 2 million Americans. The battery of a pacemaker lasts ten years or more. Pacemakers have improved the quality of life as well as lengthened the lives of many millions of people.

PARTICIPATION ACTIVITY 3.19.1: The human heart and a pacemaker.

Animation content:

undefined

Animation captions:

1. A human heart has two atria (left and right) and two ventricles (left and right).
2. A pacemaker has a sensor to detect a natural contraction in the heart's right ventricle. If a contraction is not detected within 0.8 s, the pacemaker will deliver an electrical stimulation using an output wire.
3. If the heart does not contract naturally within 0.8 seconds of the last contraction (natural or paced), the pacemaker forces a contraction.

PARTICIPATION ACTIVITY 3.19.2: Pacemaker basics.

1) Which ventricle is the pacemaker connected to in the above example?
   - left
   - right

2) If a contraction is detected every 0.8 seconds for 1 minute, how many output stimulations does the pacemaker send to the heart?
   - 0
   - 75

Basic pacemaker controller

A basic pacemaker can be implemented as an FSM controlling a timer.

PARTICIPATION ACTIVITY 3.19.3: Basic pacemaker controller.

Animation content:

undefined

Animation captions:

1. The heart has two atria, ra and la, and two ventricles, rv and lv. An input s to the pacemaker from rv is set to 1 when a contraction is sensed in the right ventricle.
2. The pacemaker consists of a controller and a timer. The timer's input t, resets the timer when t = 1. On reset, the timer counts down from 0.8 seconds.
3. If the timer counts down to 0, the timer sets the output z to 1. If the timer is reset before reaching 0, z is not set to 1, and the timer starts counting down again.
4. The controller sets output p to 1 to cause a paced contraction. The controller has an input s, which is 1 when a contraction is sensed in the right ventricle.
5. The controller resets the timer in state ResetTimer by setting t = 1. The controller then stays in Wait until the timer does not reach 0 (z') and as long as a contraction is not detected (s').
6. If the controller detects a contraction (s), the timer is reset and starts counting again.
7. If the timer has reached 0 (z), the controller transitions to Pace, which sets p = 1, causing a paced contraction.

PARTICIPATION ACTIVITY 3.19.4: Basic pacemaker controller.

1) How long does the FSM wait in the Wait state?
   - 0.8 seconds
   - until a contraction is detected
   - 0.8 seconds OR until a contraction is detected

2) From the ResetTimer state, when will the FSM transition to the Wait state?
   - When a contraction is detected (s = 1)
   - When the timer reaches 0 (z = 1)
   - On the next rising clock edge

3) Can the FSM skip the Pace state more than once?
   - No
   - Yes

Atrioventricular pacemaker

The atria receive blood from the veins and contract to push the blood into ventricles. The atrial contractions occur just before the ventricular contractions. Many pacemakers, known as atrioventricular pacemakers, sense and pace the ventricular contractions and the atrial contractions.

PARTICIPATION ACTIVITY 3.19.5: An atrioventricular pacemaker.

Animation content:

undefined

Animation captions:

1. An atrioventricular pacemaker has two sensors (sa to sense the right atrium, sv to sense the right ventricle), and two output wires for electrical stimulation (pa connected to the atrium, pv connected to the ventricle).
2. Two timers are used, one for the right atrium and one for the right ventricle. The controller initially resets TimerA in ResetTimerA, by setting ta = 1, and then waits for a contraction or the timer to reach 0.
3. If the controller detects a contraction (sa) the controller skips the pacing of the atrium. If TimerA reaches 0 first, then the controller transitions to PaceA, setting pa = 1, causing a

contraction.

4. After an atrial contraction (natural or paced), the controller resets TimerV in ResetTimerV by setting tv =1. If a natural ventricular contraction occurs, the controller skips pacing of the ventricle.
5. If TimerV reaches 0 first, then the controller transitions to PaceV, setting pv = 1, causing a contraction in the ventricle. The controller then returns to the atrial states.

PARTICIPATION ACTIVITY 3.19.6: Atrioventricular pacemakers.

1) How many sensors does the pacemaker shown in the above example have?

- 1
- 2
- 4

2) When does the FSM transition to the PaceV state?

- When TimerA reaches 0.
- When a contraction is sensed in the ventricle.
- When TimerV reaches 0 and a contraction is not sensed.

©zyBooks 10/17/21 21:57 829162
Ivan Neto
UCRCS120AEE120AChenFall2021