Synchronous sequential circuits

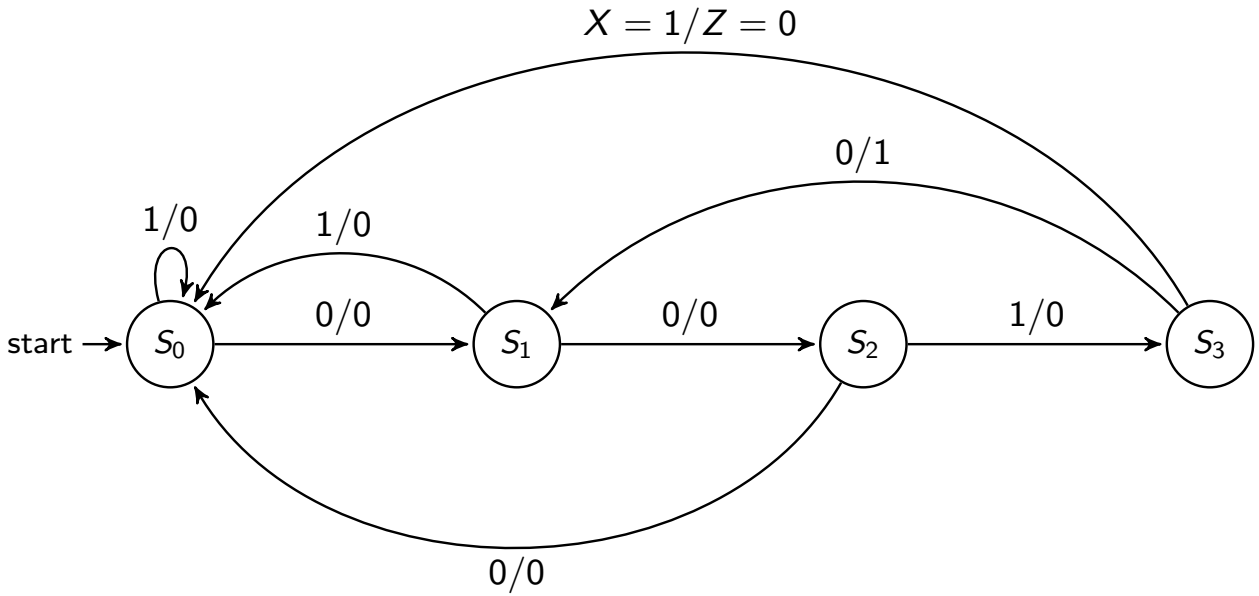
- Synchronous circuits are characterized by the presence of flip flops and clock signals.
- We've seen a few examples already — registers and counters are examples of synchronous, or clocked, sequential circuits.
- We will consider both the *analysis* and *design* of synchronous sequential circuits.

State diagrams

- The *state diagram* is a useful idea for the pictorial representation of how a synchronous sequential circuit should operate to perform a task.
- The state illustrates several things about a circuit:
  1. The states of the circuit;
  2. Transitions between states which depend on the circuit inputs;
  3. Required circuit output values which depend on the state and, possibly, the circuit inputs.
- The states of a sequential circuit are represented by *state bubbles* in the state diagram.
- The transitions from state to state are represented by *directed edges* or *arcs* in the state diagram.
- Circuit output values are labeled either inside of the state bubbles or on the arcs — it depends whether or not the outputs depend on only the state or on the state and the circuit inputs.
- The state diagram might also illustrate a *initial state* or *reset state* which is the desired start state for the circuit.

State diagrams

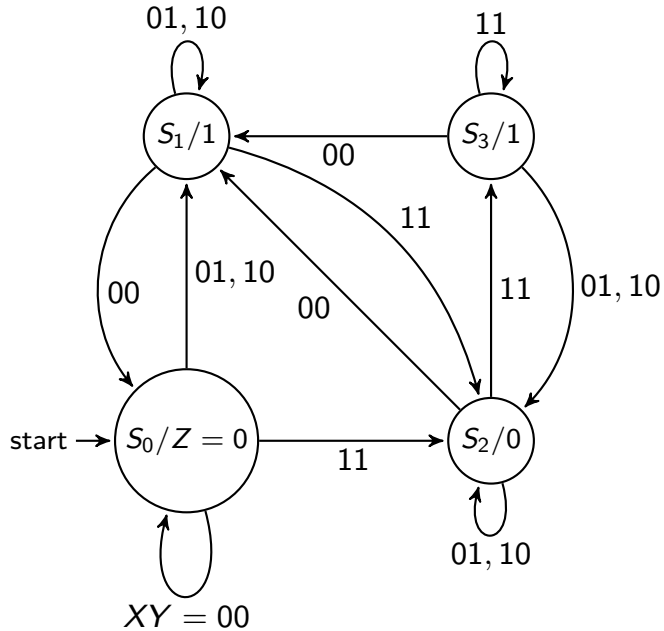
- The following is an example of a state diagram.



- This state diagram is for a design that requires 4 states denoted S_0, S_1, S_2 and S_4. The initial state of the design is S_0 denoted by the arc labeled "start" and pointing at S_0 from "nowhere".
- The circuit has one input X and one output Z — this is deduced by examining the edges where we see notation like $X = value/Z = value$.
  - When the circuit outputs depend on both the state and the circuit inputs, their values are labeled on the *arcs*.
  - Such state diagrams are often called *Mealy state diagrams*.

State diagrams

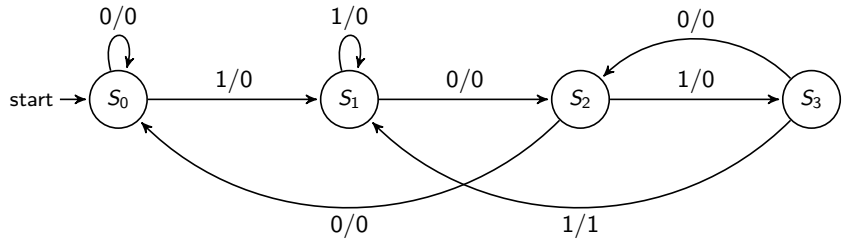
- Another example of a state diagram.



- State diagram for a design with requires 4 states: S_0, S_1, S_2 and S_4. Initial state is S_0 denoted by the arc labeled "start" and pointing at S_0.
- Circuit has two input X and Y — deduced by looking at the arcs. No outputs labeled on the arcs!
- Circuit has one output Z — deduced by looking *inside* state bubbles where we see notation like $state/Z = value$.
  - Outputs labeled inside of a state bubble only depend on the current state.
  - When the circuit outputs depend only on the state and **not** on circuit inputs their values are labeled inside the *state bubbles*.
  - Such state diagrams are often called *Moore state diagrams*.

State tables

- The *state table* describes the behaviour of a sequential circuit in tabular form.
- It shows the same information as a state diagram.
- State diagram...



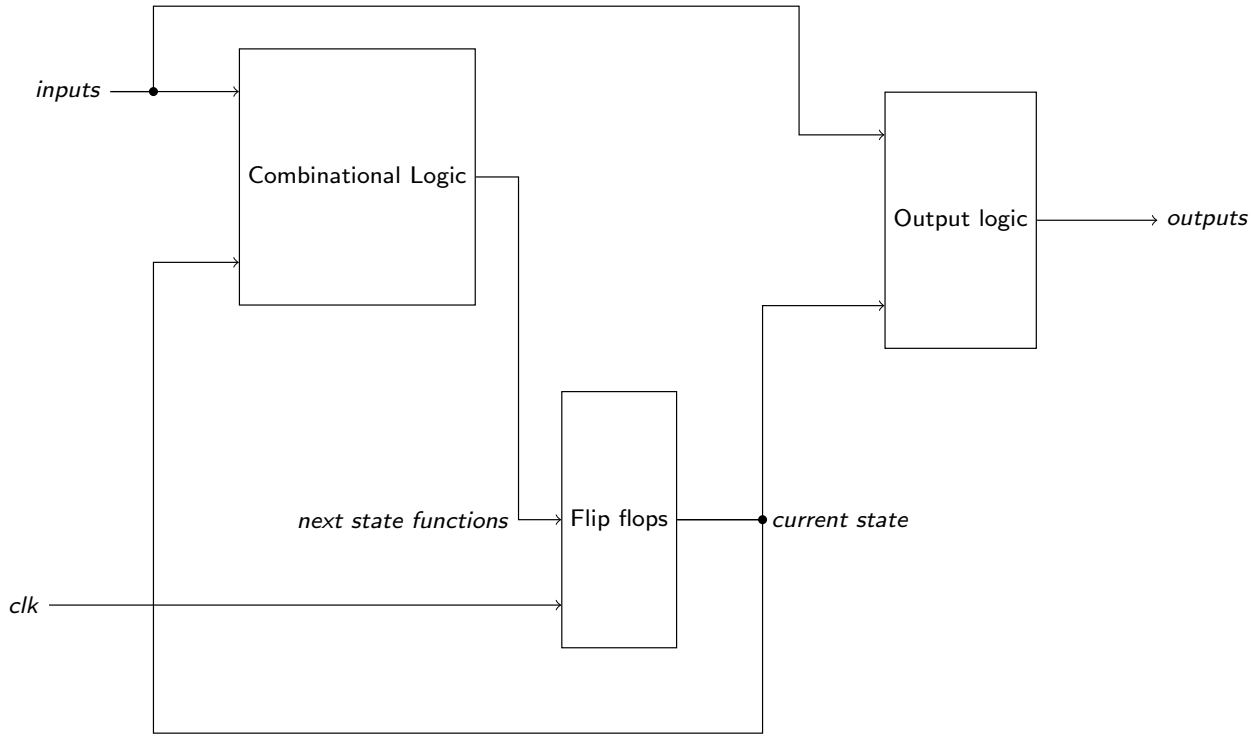
- Corresponding state table...

Current State	**Next State**		**Output** Z	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
S_0	S_0	S_1	0	0
S_1	S_2	S_1	1	1
S_2	S_0	S_3	0	0
S_3	S_2	S_1	0	1

- From the state diagram there is clearly one input — we've called it X. There is clearly one output — we've called it Z.
- This state table shows states *symbolically*; e.g., they have names like S_0, S_1, and so forth.

Structure of a sequential circuit

- Synchronous sequential circuits have a structure to them:



- The **current state** is represented as a binary pattern at the output of the flip flops.
- Based on the circuit inputs and the current state, we have **next state functions** (combinational logic gates) which are connected to flip flop inputs. These functions determine the **next state** upon the arrival of the active clock edge.
- We have output logic (combinational logic gates) which based on the state and (possibly) determine the circuit outputs.

Sequential circuit analysis

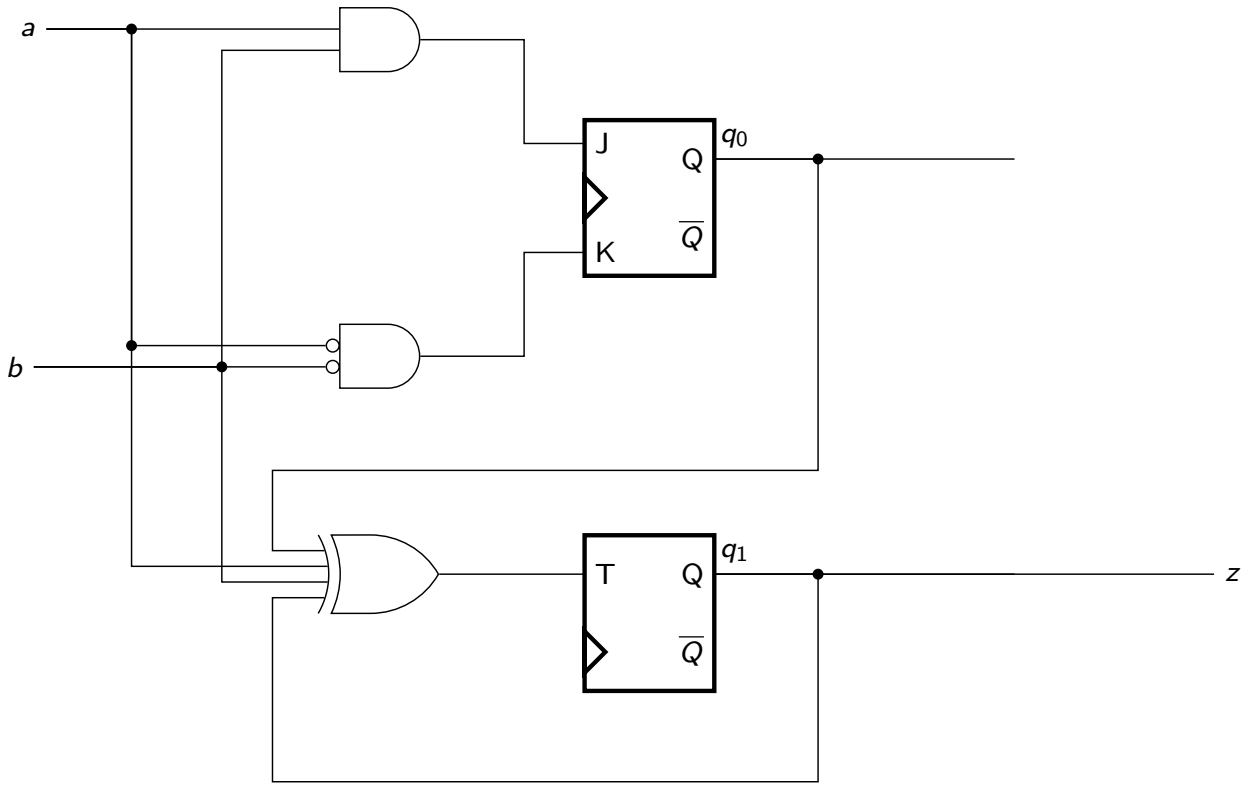
- Circuit analysis implies figuring out what a circuit is doing. We might want to do this for a few reasons:
  - We are given a circuit and we simply want to know what it is doing.
  - We are given a circuit which we are told performs a certain operation and we want to figure out if it is correct.
  - We want to get the "abstract" behaviour of the circuit so that we could reimplement it a different way.
- For a synchronous sequential circuit, analysis implies determining a symbolic state table or symbolic state diagram given a circuit drawing.

Sequential circuit analysis

- The procedure involves the following steps:
  - Identifying the flip flops. This tells us the potential number of states.
  - Identifying the logic that produces the circuit outputs and writing down the output equations.
  - Identifying the next state functions and writing down their logic equations. This tells us how the circuit transitions from state to state.
  - Obtaining a state table and/or state diagram.
  - Abstracting away anything which is particular to the implementation. For example, how the states have been encoded.
- Again, our goal is to obtain the state table or state diagram since that's what specifies the behaviour of the circuit.

Sequential circuit analysis

- Consider the following circuit...



Sequential circuit analysis

- Observations...
  - Circuit has two inputs a and b.
  - Circuit has one output z.
  - Circuit has two flip flops — this implies that, at most, this circuit is implementing a state diagram with 4 states maximum.
- Because we are given a circuit, we don't know the "description" of the states... We only know that they are being represented by the binary values 00, 01, 10 and 11.

Sequential circuit analysis

- We write logic equations for the circuit output in terms of the flip flop outputs (the *current state*) and the *circuit inputs*.

$$z = q_1$$

- We write logic equations for the flip flop inputs (the *next state functions* in terms of the flip flop outputs (the *current state*) and the *circuit inputs*.

$$\begin{aligned} j_0 &= ab \\ k_0 &= a'b' \end{aligned}$$

$$t_1 = a \oplus b \oplus q_0 \oplus q_1$$

We need these equations so we can figure out the transitions from one state to another state when the clock edge arrives.

Sequential circuit analysis

- Recall how these flip flops work...

J	K	$Q(t+1)$
0	0	$Q(t)$
0	0	0
0	0	1
1	1	$\overline{Q}(t)$

T	$Q(t+1)$
0	$Q(t)$
1	$\overline{Q}(t)$

- Table showing current state and flip flop input values...

current state	t_1			
$q_1 q_0$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$
00	0	1	0	1
01	1	0	1	0
10	1	0	1	0
11	0	1	0	1

current state	$j_0 k_0$			
$q_1 q_0$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$
00	01	00	10	00
01	01	00	10	00
10	01	00	10	00
11	01	00	10	00

- Table showing next state... derived from the flip flop input values...

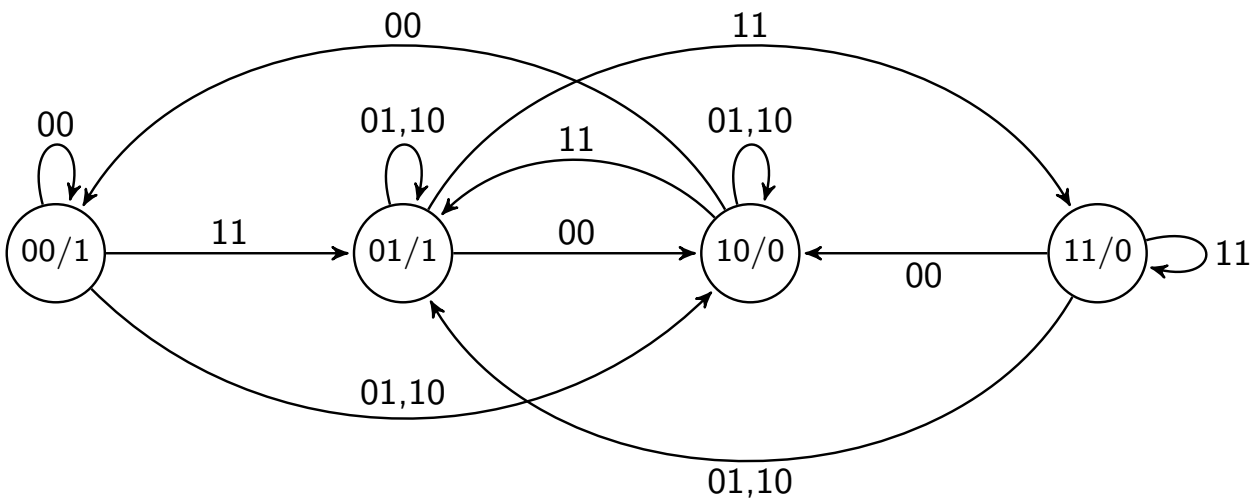
current state	*next state* $(q_1 q_0)$			
$q_1 q_0$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$
00	00	10	01	10
01	10	01	11	01
10	00	10	01	10
11	10	01	11	01

Sequential circuit analysis

- State table... shows next state *and* circuit outputs...

current state	*next state* ($q_1 q_0$)				*outputs* (z)			
$q_1 q_0$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$
00	00	10	01	10	0	0	0	0
01	10	01	11	01	0	0	0	0
10	00	10	01	10	1	1	1	1
11	10	01	11	01	1	1	1	1

- State diagram (from the circuit, it is unclear what the initial state should be)...

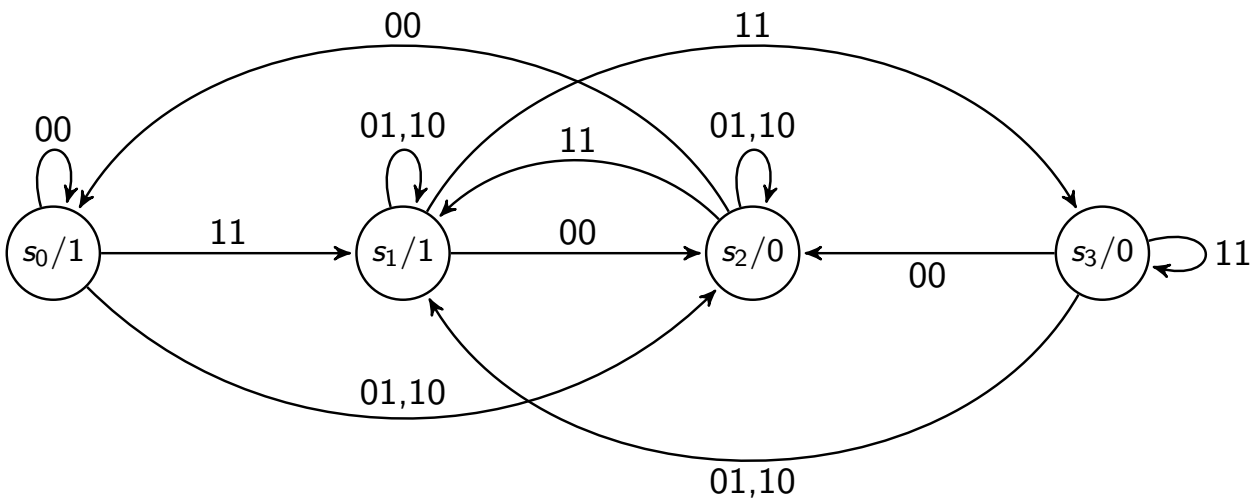


Sequential circuit analysis

- Last step might be to "abstract out" the particular numerical values representing the different states; e.g., you might be able to figure out that these states represent things like "start", "stop", and so forth.
- Let's just call them s_0, s_1, s_2 and s_3 to denote 00, 01, 10 and 11 respectively.

current state	*next state* $(q_1 q_0)$				*outputs* (z)			
$q_1 q_0$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$	$ab = 00$	$ab = 01$	$ab = 11$	$ab = 10$
s_0	s_0	s_2	s_1	s_2	0	0	0	0
s_1	s_2	s_1	s_3	s_1	0	0	0	0
s_2	s_0	s_2	s_1	s_2	1	1	1	1
s_3	s_2	s_1	s_3	s_1	1	1	1	1

- State diagram (from the circuit, it is unclear what the initial state should be)...



Sequential circuit design

- Circuit design implies figuring out from a verbal problem description how to implement a circuit to accomplish what is described.
- The procedure involves a few steps...
  1. Understand the verbal problem description.
  2. Figure out what states are required and how we transition from state to state based on circuit inputs. Figure out what the circuit outputs need to be — this will lead to a state diagram and/or a state table. Note that things might be *symbolic* at this point; the states might have *names* rather than binary numbers to represent them.
  3. Perform *state reduction*. It could be that our state diagram and/or state table, while valid and correct, uses more states than what is required. Often we can find *equivalencies* between states and get a smaller state diagram and/or state table that will also work. This could lead to a smaller circuit.
  4. Perform *state assignment*. This means giving each state a binary number to represent it.
  5. Select a flip flop in order to store the state information. The number of flip flops depends on how many bits are required to represent the states.
  6. Derive output logic equations and next state (flip flop input) equations.
  7. Draw the resulting circuit.
- I would like to comment here that we have seen an example of sequential circuit design when we did generic counter implementation!

Sequential circuit design

- Comment... I'd like to point out that we've seen an example of sequential circuit design already when we considered designing circuits for generic count sequences!
  - We required a number of states equal to the values that our counter needed to produce; (e.g., count $1 \rightarrow 7 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 6$ and repeat requires 6 states). The circuit outputs are the binary values of the count sequence;
  - An active clock edge was all that was required to move from one state to the next state;
  - Each state was given a name; (e.g., we have state "1", "7", "3", and so forth);
  - Each state was "assigned" or represented in binary using the binary representation of the number. Here, we'd need 3 bits and the states would be represented as $1 \rightarrow 001$, $7 \rightarrow 111$, $3 \rightarrow 011$, and so forth.
  - We used all different types of flip flops;
  - The outputs were equal to the state for a counter.
- Anyway... we can consider more complicated sorts of circuits now...

Sequential circuit design — sequence detector example

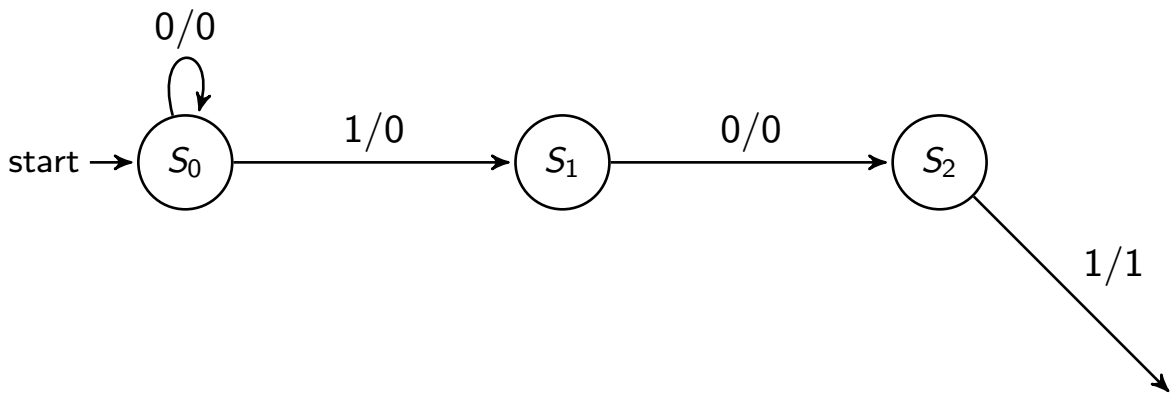
- Sequence detectors are a good sort of circuit to consider when trying to learn sequential circuit design. We will do a few examples.
- The problem...

  > *Design a circuit that has one input x and one output z. The input x is assumed to change only between active clock edges. When any input sequence* 101 *is detected, the circuit should output a* 1*, otherwise* 0*. A sample input and output sequence might be*
  > x : 001101010001000100100...
  > z : 000001010001000001000...

- A useful way to design such circuits is to start with an initial state where no input has been received. Then, we can start adding states to follow the sequence which requires a 1 as output. Finally, we can fill in any remaining edges (and add additional states if required).
- Note that at this point, we don't know how many states are required, how many flip flops we will need, and so forth.

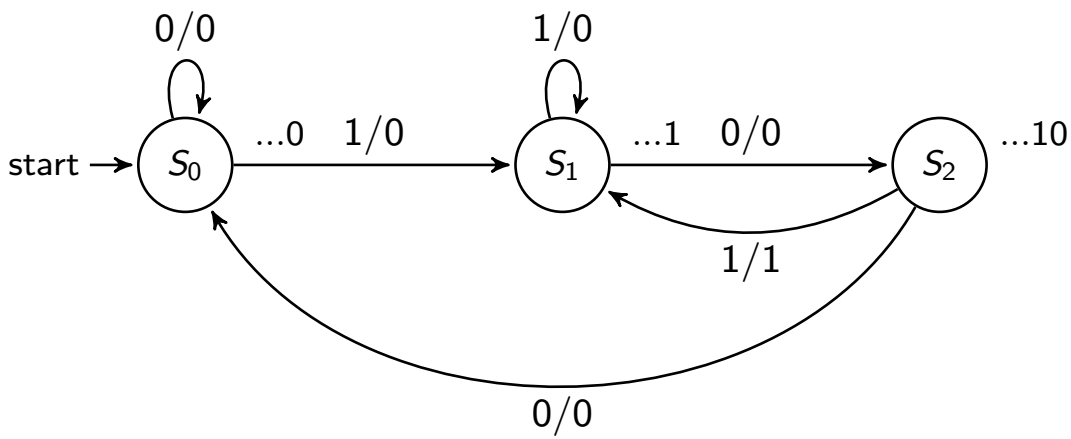
Sequential circuit design — sequence detector example

- We will start our circuit in state S_0. Since we are looking for the pattern 101, we can remain in this state until we at least see a 1. Therefore, this state represents seeing a sequence of either nothing or a sequence of 0s which we can denote as ...0. When an input of 1 is received, we move to a new state S_1 which represents string of at least one 1 or ...1. This state also represents the start of a new pattern. If in state S_1 and we receive a 0, we have seen the first 2 bits of our pattern which we can denote as ...10 and we can move to another state s_2 to represent the fact we have reached ...01. Now, once in state S_2 if we see a 1 we can output a 1. This is the first part of the state diagram (the detection of the sequence).



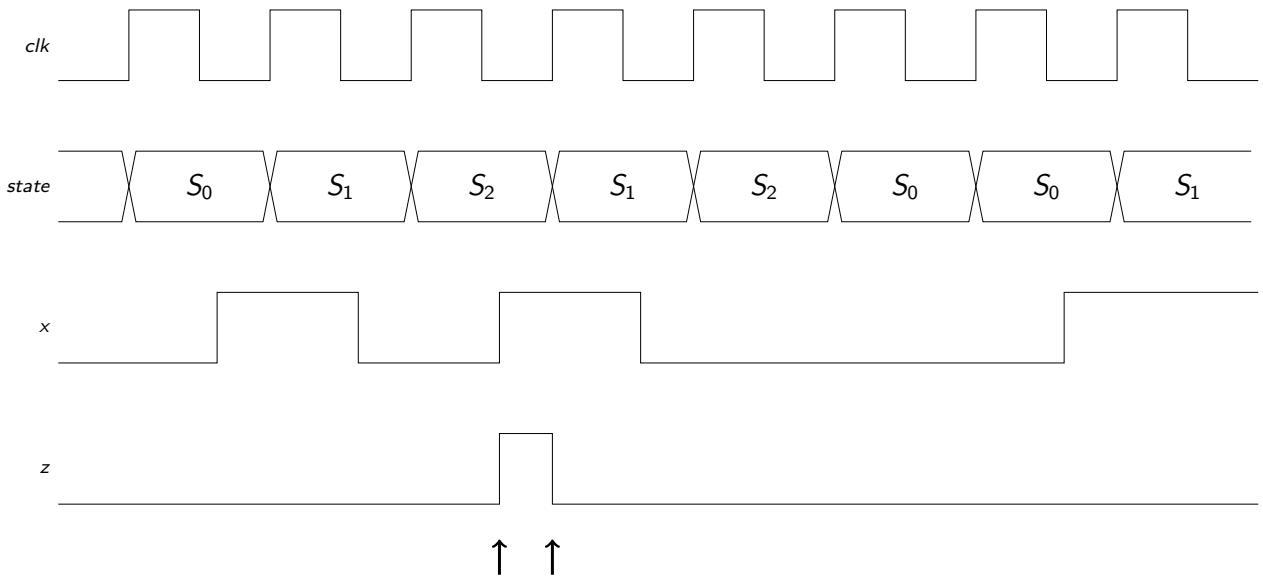
Sequential circuit design — sequence detector example

- We can now fill in the missing edges as well as figure out if we need any more states. If in S_1, we have already ...1. If we see additional 1s we can remain in S_1. If in S_2 and we receive a 0, we have seen ...100 and this breaks our pattern. We can return to S_0 and record that we have seen ...0. Finally, if in S_2 and we receive a 1, we output a 1 and can return to S_1 since this 1 not only represents the end of our pattern, but the potential start of the next occurance of the pattern.



Sequential circuit design — sequence detector example

- As an aside, we should be careful when we read the output value to avoid reading false outputs. The output is valid when we are in S_2 and after the output has changed to a 1, but the next clock edge has not yet arrived. This is illustrated below assuming the positive edge of the clock is the active edge.



Sequential circuit design — sequence detector example

- The state table...

Current State	**Next State**		**Output (z)**	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
S_0	S_0	S_1	0	0
S_1	S_2	S_1	0	0
S_2	S_0	S_1	0	1

Sequential circuit design — sequence detector example

- The next step is *state reduction*...
- This problem is pretty simple and it's our first example, so we will skip state reduction for now (and come back to it later).
- This problem doesn't actually allow for any state reduction. The state diagram and state table are already as small as they are going to get.

Sequential circuit design — sequence detector example

- We need to do *state assignment*. This means assigning a binary pattern to represent each state. In this example, we have three states and therefore need *at least* enough bits to represent three different values. Of course, we can do this with 2 bits. We can choose S_0 is 00, S_1 is 01 and S_2 is 10. Note that 11 is unused.
- The state table with binary values to represent states. Recall the current state is stored at the output of flip flops. Therefore, we will require 2 flip flops since we are using 2 bits for the state representation. Call these bits q_1 and q_0...

Current State ($q_1 q_0$)	**Next State ($q_1 q_0$)**		**Output (z)**	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
00	00	01	0	0
01	10	01	0	0
10	00	01	0	1

Sequential circuit design — sequence detector example

- We are at the point where we need to select a flip flop type and figure out the circuit implementation. We can use *DFF*, *TFF* or *JKFF* or even some combination of different types of flip flops.
- We will consider implementing all potential circuits... We therefore need to figure out the necessary flip flop inputs in order to get the desired changes in state as a function of the current state and the circuit input.
- If there are *unused current states* (potential binary patterns which we did not need to use), then these result in don't cares for the flip flop inputs. In this example, there is no state for pattern 11 and therefore if the current state was somehow 11, the flip flop inputs are all don't cares.

Sequential circuit design — sequence detector example

- If using *DFF*... The necessary *DFF* inputs to obtain the desired state transitions...

Current State ($q_1 q_0$)	**Next State** ($q_1 q_0$)		**DFF inputs** ($d_1 d_0$)	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
00	00	01	00	01
01	10	01	10	01
10	00	01	00	01

- This leads to the flip flop input equations $d_1 = \overline{x} q_0$ and $d_0 = x$ (these equations are available using a Karnaugh map).
- We can also find (using a Karnaugh map) that the output $z = q_1 x$.

Sequential circuit design — sequence detector example

- If using *TFF*... The necessary *TFF* inputs to obtain the desired state transitions...

Current State ($q_1 q_0$)	**Next State** ($q_1 q_0$)		**TFF inputs** ($t_1 t_0$)	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
00	00	01	00	01
01	10	01	11	00
10	00	01	10	11

- This leads to the flip flop input equations $t_1 = q_1 + \overline{x} q_0$ and $t_0 = \overline{x} q_0 + x \overline{q}_0 = x \oplus q_0$ (these equations are available using a Karnaugh map).
- We can also find (using a Karnaugh map) that the output $z = q_1 x$.
- Compared to the *DFF*, both t_1 and t_0 are more complicated than d_1 and d_0 so, thus far, *DFF*s are a better choice.

Sequential circuit design — sequence detector example

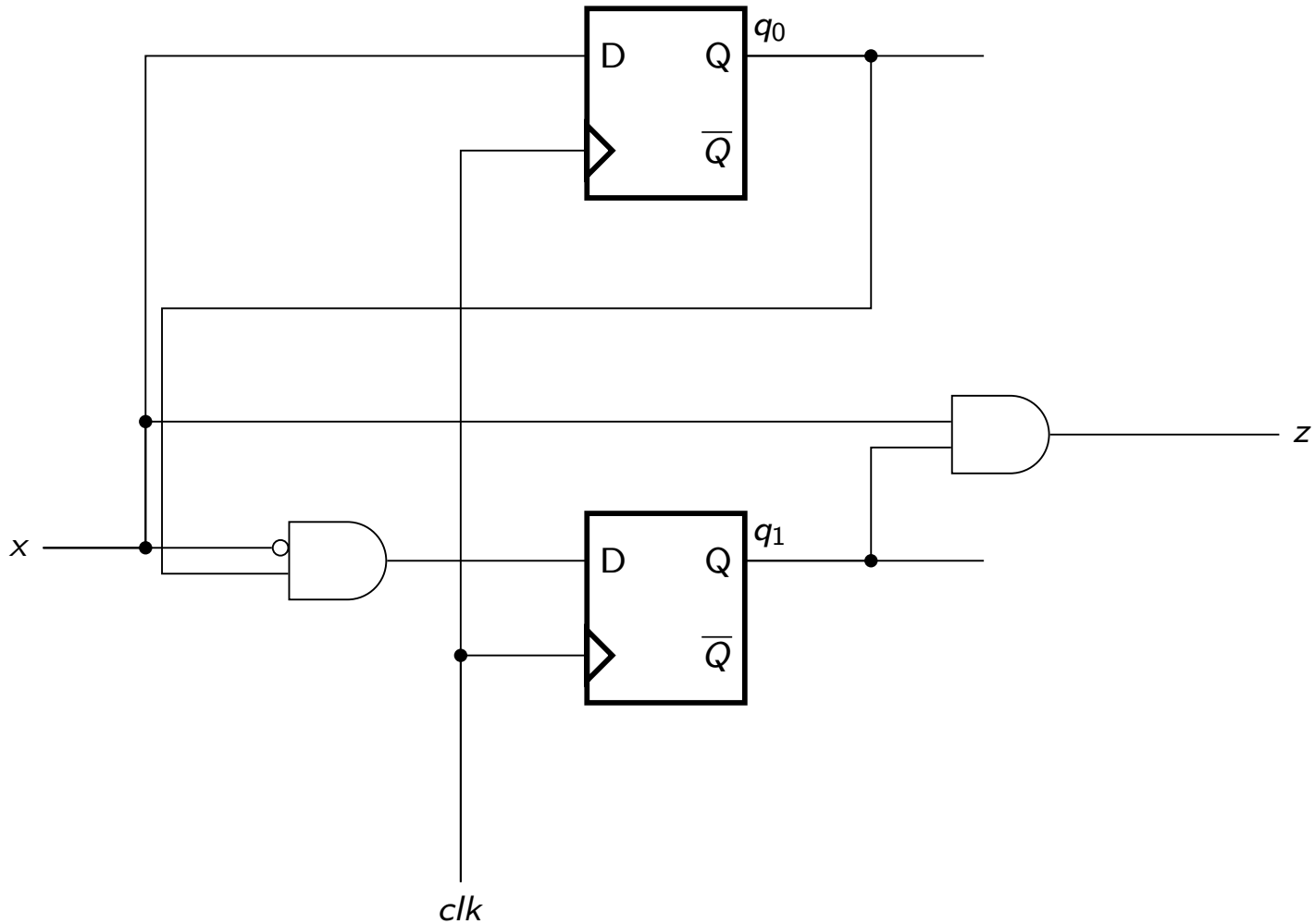
- If using *JKFF*... The necessary *JKFF* inputs to obtain the desired state transitions...

Current State ($q_1 q_0$)	**Next State** ($q_1 q_0$)		**JKFF inputs** ($j_1 k_1$)		**JKFF inputs** ($j_0 k_0$)	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$	$x = 0$	$x = 1$
00	00	01	$0X$	$0X$	$0X$	$1X$
01	10	01	$1X$	$0X$	$X1$	$X0$
10	00	01	$X1$	$X1$	$0X$	$1X$

- This leads to the flip flop input equations $j_1 = \overline{x} q_0$, $k_1 = 1$, $k_0 = x$ and $j_0 = \overline{x}$ (equations from Karnaugh maps).
- We can also find (using a Karnaugh map) that the output $z = q_1 x$.
- Compared to the *DFF*, the circuit using *JKFF* also looks more complicated so, thus far, *DFF* are a better choice.

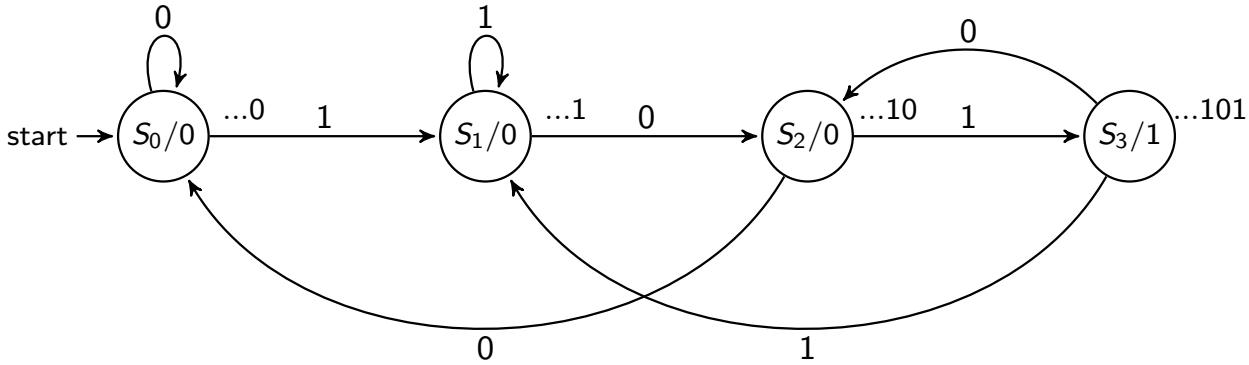
Sequential circuit design — sequence detector example

- The circuit using *DFF*s...

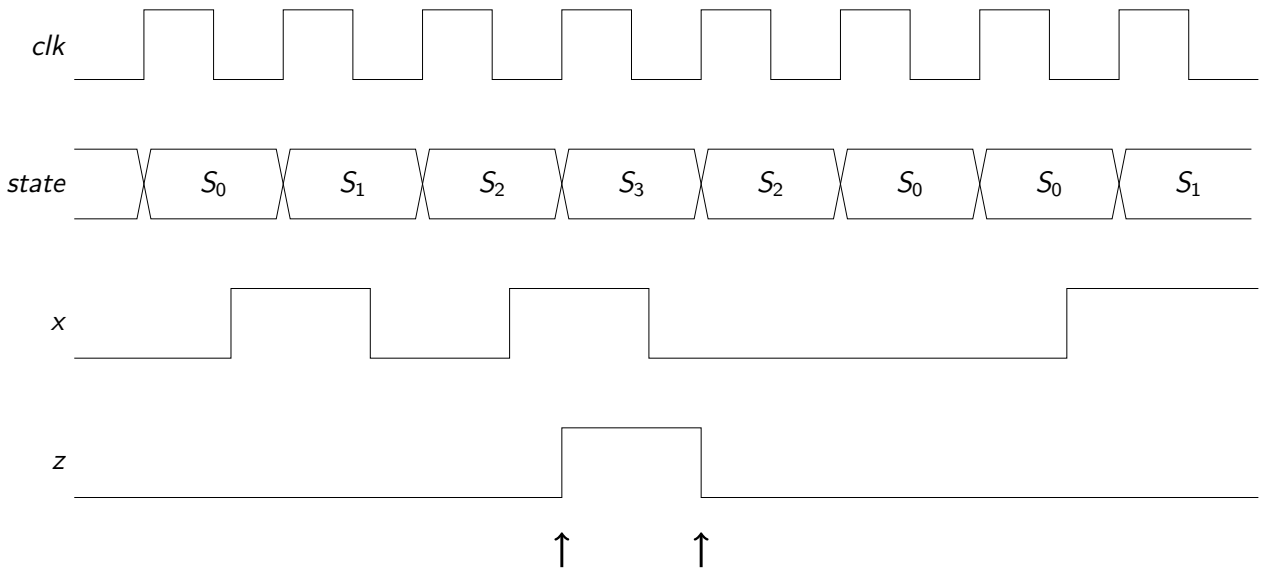


Sequential circuit design — sequence detector example

- What if implemented as a Moore state diagram... The state diagram would have looked like this...



- The timing diagram...



- Moore machine requires an additional state. The output is valid for an entire clock cycle. Timing is less problematic with Moore state diagrams.

Sequential circuit design — another example

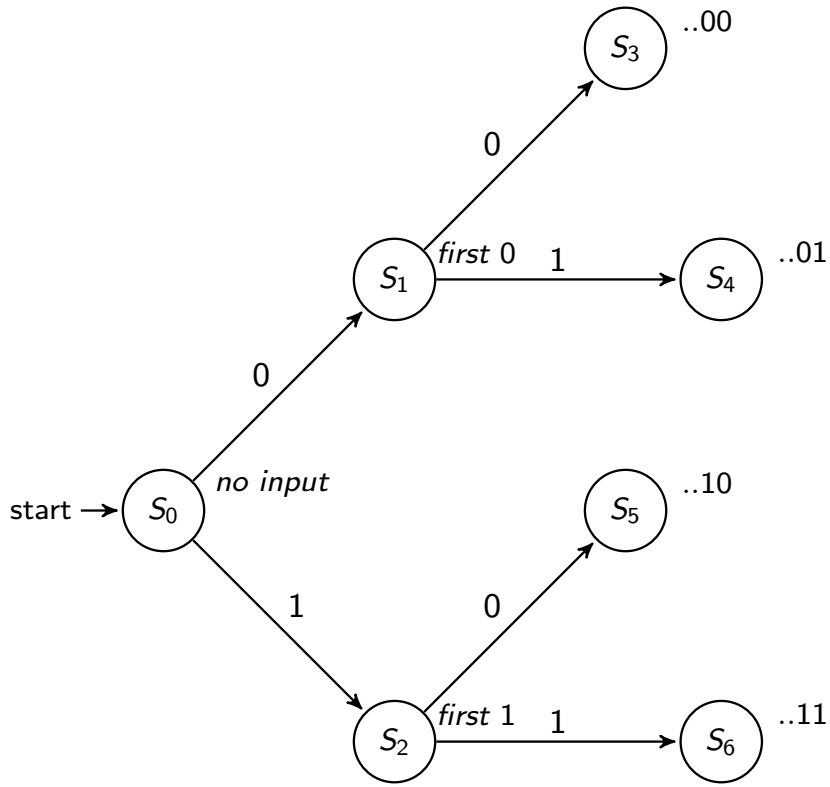
- One more example... (state diagram only)... The problem...

  Design a Mealy state diagram for a circuit that has one input x and one output z. The input x is assumed to change only between active clock edges. The circuit output z should be 1 *when exactly two of the last three inputs are* 1*.*

- In this case, it is useful to think of two things. First, we need to account for the start of the sequence when we have not yet received at least 3 bits. We can have a state "no input", we can have a state "first 0" and a state "first 1". Then, we can have further states, but we will define these states as the "last two bits are...".

Sequential circuit design — another example

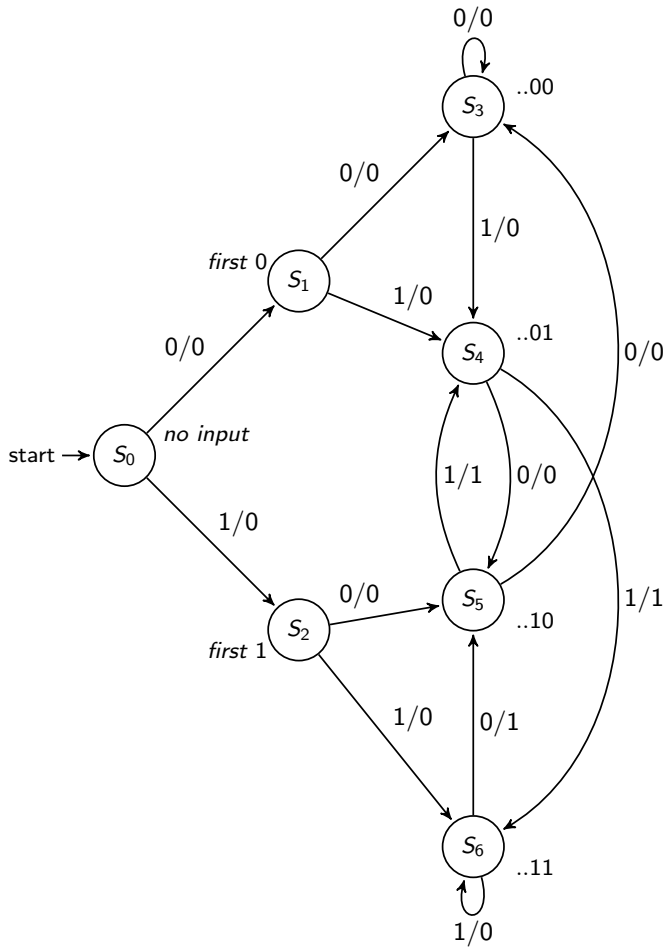
- Partial state diagram...



- It should now be clear that, based on the next input, we will simply move between S_3 to S_6 generating either $z = 1$ or $z = 0$ depending on the next input.

Sequential circuit design — another example

- Completed state diagram...



- It should now be clear that, based on the next input, we will simply move between S_3 to S_6 generating either $z = 1$ or $z = 0$ depending on the next input.

Sequential circuit design — another example

- We could continue with this example if we wanted... Write a state table, perform state reduction, figure out the number of bits required and do state assignment, choose a flip flop type and so on...