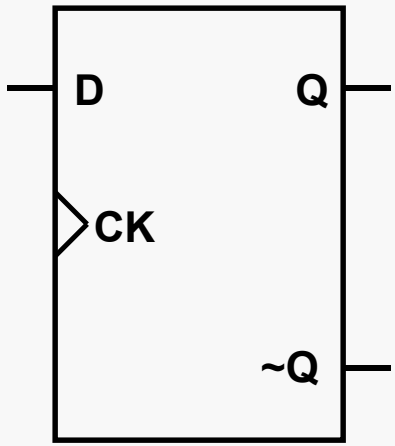
D Flip-flop

The D flip-flop takes one data input and updates its state Q, on a clock tick, according to the table:

D	Q	~Q
0	0	1
1	1	1

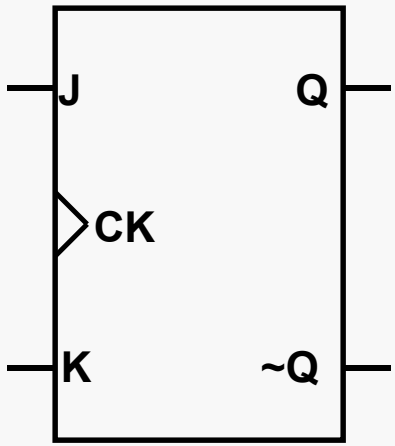


In the following Logisim diagrams, the D flip-flops update state on the falling edge (when the clock goes from high to low).

JK Flip-flop

The JK flip-flop takes two data inputs and updates its state Q, on a clock tick, according to the table:

J	K	Q	~Q
0	0	no change	
0	1	0	1
1	0	1	0
1	1	opposite	



In the following Logisim diagrams, the JK flip-flops update state on the falling edge (when the clock goes from high to low).

Recognizers

A *recognizer* accepts a binary input (typically a sequence of bits) and outputs a 1 if and only if the binary input matches a specific pattern.

You might think of a recognizer as being similar to a combination lock.

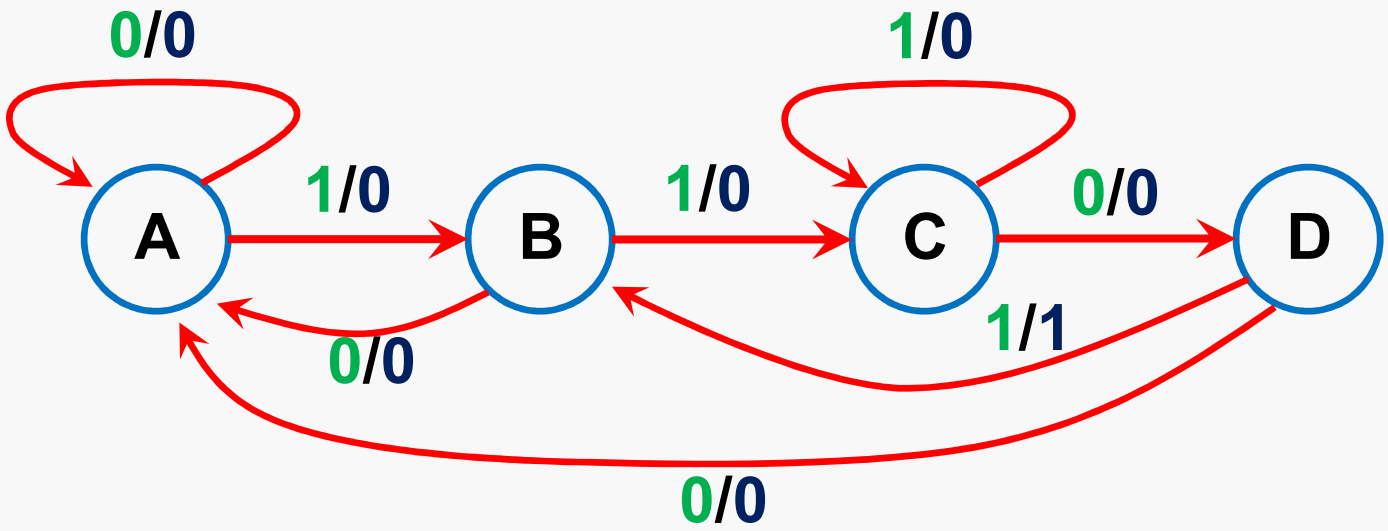
The output signal can be used to control the activation of another circuit.

Design: recognizing 1101

We need to recognize a sequence of 4 bits, which suggests the following logical states:

- **A** initial state/have not yet seen a 1
- **B** last bit seen was 1, previous bit was not a 1
- **C** last two bits seen were 1's
- **D** last bit seen was 0; previous two bits were 1's

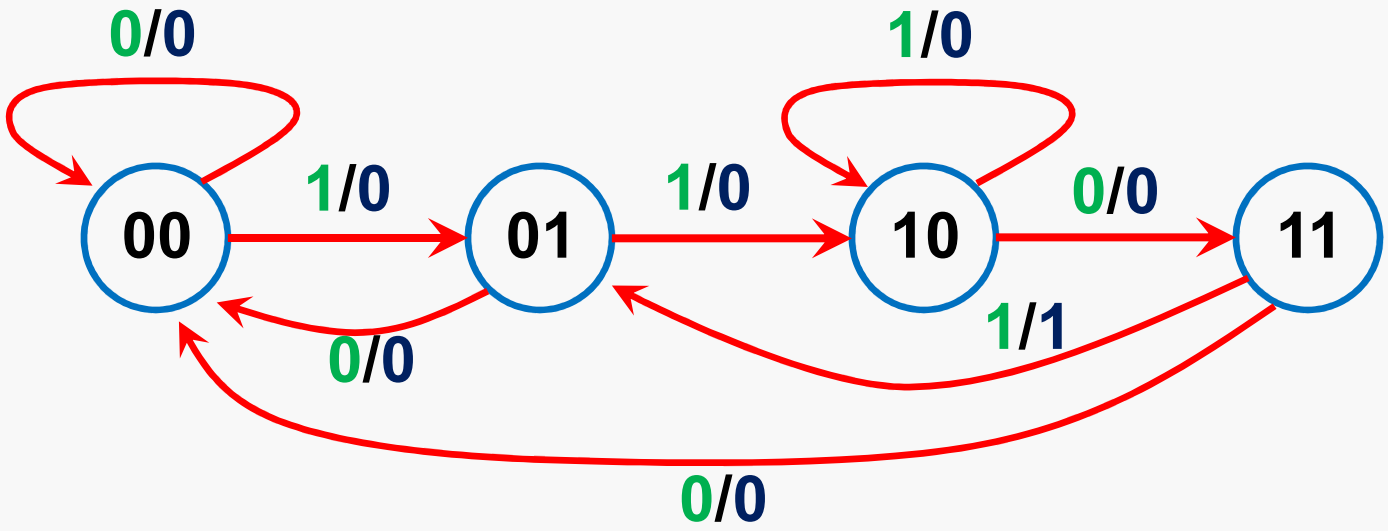
We recognize the pattern **1101** when we make a transition from state **D** back to state **B**.



Design: mapping states to labels

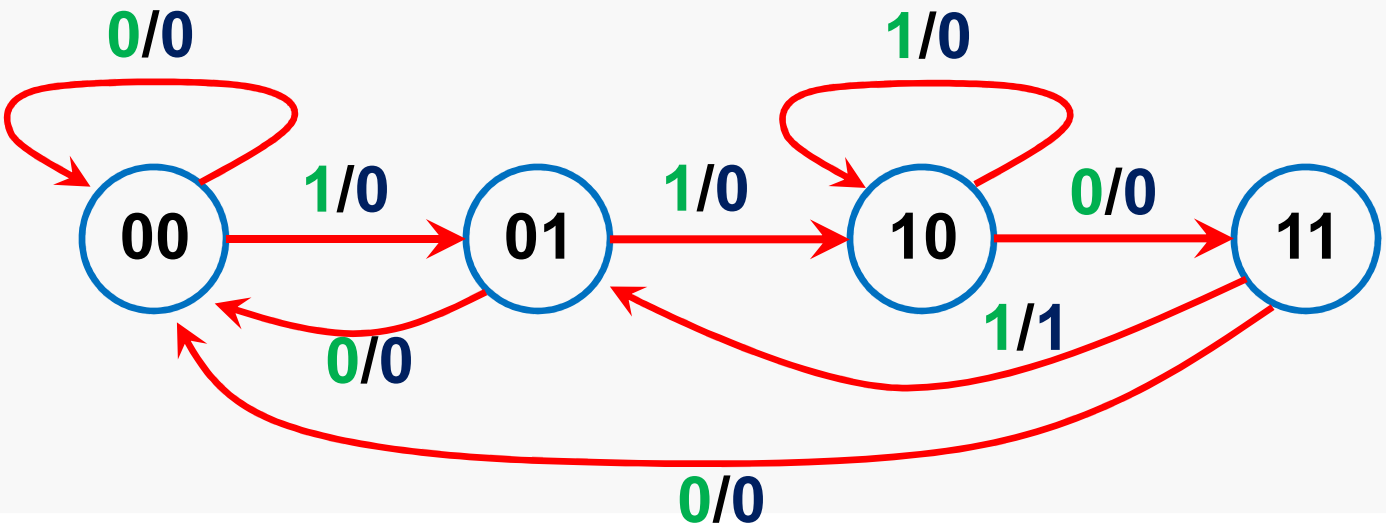
We need to represent each state by a binary string; a trivial mapping will do:

A	00
B	01
C	10
D	11



Design: state transition table

Curr State		Input	Next State		Output
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1



Design: choose a flip-flop

We will choose D flip-flops, since that simplifies the derivation of the next-state equations.

It is an interesting exercise to design an alternate implementation using JK flip-flops.

Design: next-state equations

C1	C0	I	N1	N0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

$$N0 = \overline{C0} \cdot \overline{C1} \cdot I + \overline{C0} \cdot C1 \cdot \overline{I} + \overline{C0} \cdot C1 \cdot I$$

$$N1 = C0 \cdot \overline{C1} \cdot I + \overline{C0} \cdot C1 \cdot \overline{I} + \overline{C0} \cdot C1 \cdot I = C0 \cdot \overline{C1} \cdot I + \overline{C0} \cdot C1$$

Design: output equation

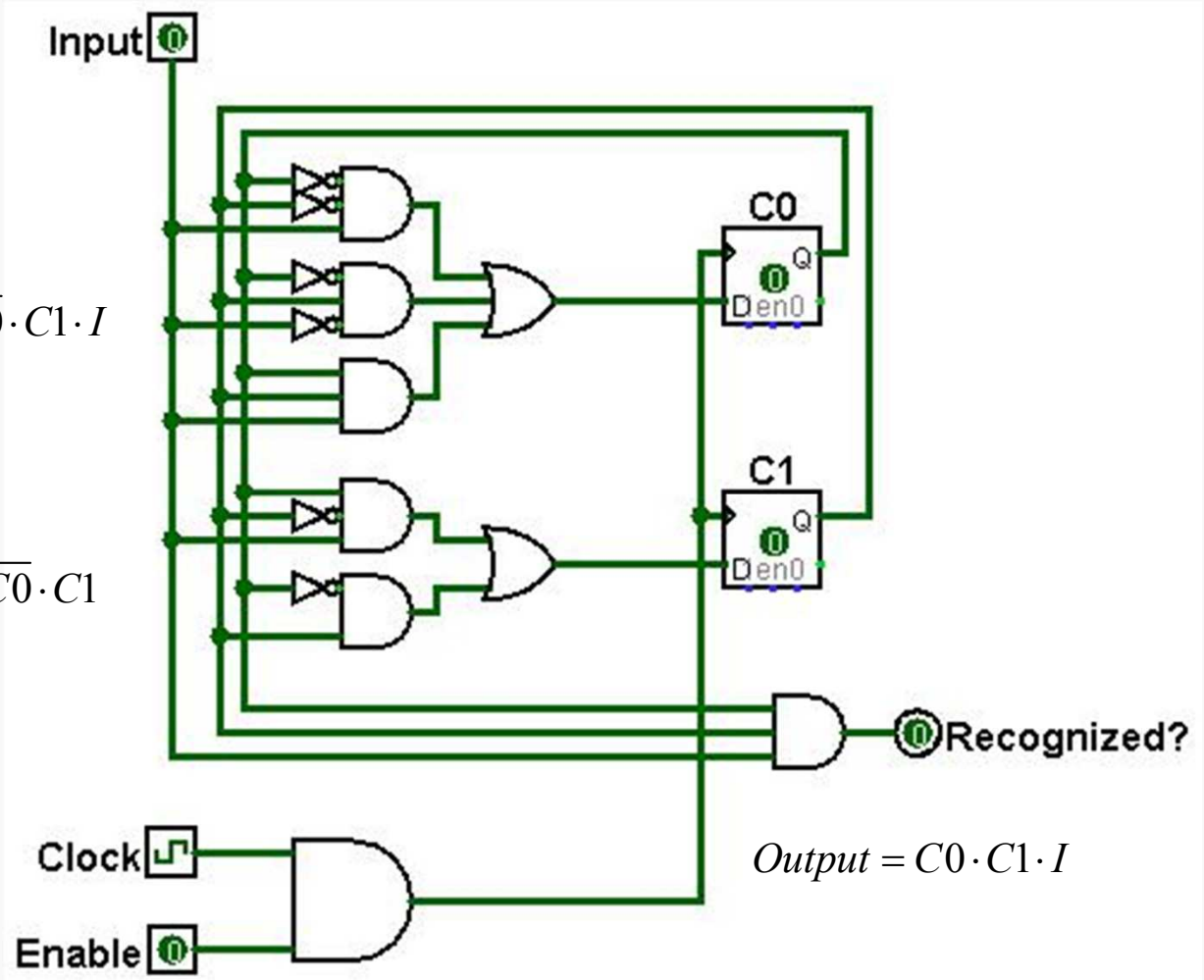
Curr State		Input	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$Output = C0 \cdot C1 \cdot I$$

Implementation

$$N0 = \overline{C0} \cdot \overline{C1} \cdot I + \overline{C0} \cdot C1 \cdot \overline{I} + \overline{C0} \cdot C1 \cdot I$$

$$N1 = C0 \cdot \overline{C1} \cdot I + \overline{C0} \cdot C1$$



$$Output = C0 \cdot C1 \cdot I$$