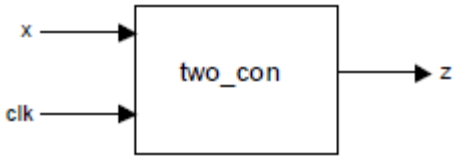
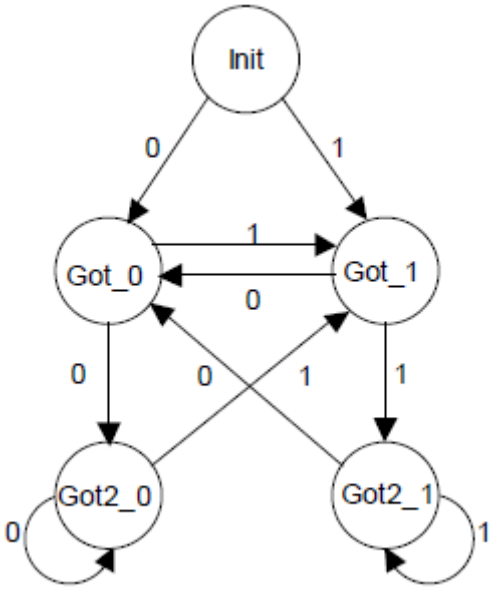
2 A simple example

In this lecture, we will look at a simple example. This is a finite state machine called two_con. Its behaviour is that the output z goes high when the input x has been at the same value for two consecutive clock cycles.



Here is a state diagram for the system.



The system starts in the system *Init*. If the input x is 1 on the next clock edge then the system moves into state *Got_1*. Similarly, if x is 0 on the next clock edge then the system moves into state *Got_0*.

Thereafter, the system will move between the states according to the value of x at the clock edge. The states have the following interpretation and outputs.

State	**Meaning**	**Output z**
Init	The initial state	0
Got_0	The input was zero at the last clock edge, but non-zero at the previous clock edge	0
Got_1	The input was one at the last clock edge but not one at the previous clock edge	0
Got2_0	The input was zero at the two previous clock edges	1
Got2_1	The input was one at the two previous clock edges	1

3 Enumerated types

Sometimes we come across types of signal that have a list of possible values. These are called enumerated types. These are very useful in the description of finite state machines in VHDL. So we could invent a new type, with five possible values corresponding to the name of the state:

```
TYPE state_type IS ( init, got_0, got_1, got2_0, got2_1 );
```

4 The finite state machine description

Here is the description of the two_con finite state machine:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY twocon IS
      PORT ( x, clk: IN STD_LOGIC; z: OUT STD_LOGIC);
END ENTITY twocon;
ARCHITECTURE simple OF twocon IS
TYPE state_type IS ( init, got_0, got_1, got2_0, got2_1 );
SIGNAL state: state_type;
BEGIN
      PROCESS (clk)
      BEGIN
      IF (rising_edge(clk) ) THEN
            CASE state IS
            WHEN init =>
                  IF x='0' THEN state <= got_0;
                  ELSIF x='1' THEN state <= got_1;
                  END IF;
            WHEN got_0 =>
                  IF x='0' THEN state <= got2_0;
                  ELSIF x='1' THEN state <= got_1;
                  END IF;
            WHEN got_1 =>
                  IF x='0' THEN state <= got_0;
                  ELSIF x='1' THEN state <= got2_1;
                  END IF;
            WHEN got2_0 =>
                  IF x='0' THEN state <= got2_0;
                  ELSIF x='1' THEN state <= got_1;
                  END IF;
            WHEN got2_1 =>
                  IF x='0' THEN state <= got_0;
                  ELSIF x='1' THEN state <= got2_1;
                  END IF;
            WHEN OTHERS => state <= init;
      END CASE;
END IF;
END PROCESS;
-- Output logic
z <= '1' WHEN state = got2_0 OR state = got2_1 ELSE '0';
END ARCHITECTURE simple;
```

Each time there is a rising edge of the clock, the process runs and assigns a new value to the *state* variable in response to the value of the input *x*. The *Others* clause is there to catch any invalid states.

4.1 Explicit assignment of the states

One of the key design challenges in implementing a finite state machine is to decide on how to represent the states. In the previous example, we deliberately avoided giving any detail about how the states were to be encoded (by using an enumerated type) so that the synthesis tool could decide how best to achieve this.
If we do have a particular state encoding in mind, then we can insert this into our VHDL. So, for example, if we want to use a particular ones-hot encoding we could change the declarations of the architecture to this:

```
ARCHITECTURE simple OF twocon IS
      SIGNAL state: STD_LOGIC_VECTOR(4 DOWNTO 0);
      CONSTANT init: STD_LOGIC_VECTOR:="00001";
      CONSTANT got_a_0: STD_LOGIC_VECTOR:="00010";
      CONSTANT got_a_1: STD_LOGIC_VECTOR:="00100";
      CONSTANT got_two_0: STD_LOGIC_VECTOR:="01000";
      CONSTANT got_two_1: STD_LOGIC_VECTOR:="10000";
BEGIN
```

The rest of the code is unchanged.