Management and analysis of physics datasets, Part. 1

Sixth Laboratory

Stefano Pavinato
19/12/2018

Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali di Legnaro

Outline

Outline

Goals

- Introduction to FSM (Finite State Machine).
- FSM in VHDL.

VHDL naming convention

Signals/components	Name
Clock	*clk*
Reset	*rst*
Input Port	*port_in*
Output Port	*port_out*
VHDL file name	*entityname.vhd*
Test bench file name	*tb_entityname.vhd*
Signal between 2 comps	*sign_cmp1_cmp2*
Process name	*p_name*
state name	*s_name*
...	...

Outline

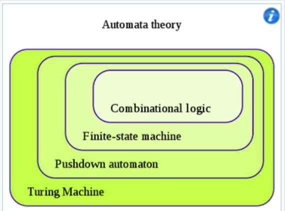
FSM (1)

Finite-state machine

From Wikipedia, the free encyclopedia

"State machine" redirects here. For infinite state machines, see State trans methodology, see State machine replication.
"SFSM" redirects here. For the Italian railway company, see Circumvesuvia
"Finite Automata" redirects here. For the electro-industrial group, see Finit

A **finite-state machine** (**FSM**) or **finite-state automaton** (**FSA**, plural: *automata*), **finite automaton**, or simply a **state machine**, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of *states* at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a *transition*. An FSM is defined by a list of its states, its initial state, and the conditions for each transition.



FSM (2)

1. A FSM can be implemented in hardware, firmware (VHDL) or software.
2. It is an abstract machine that can be in only one of a finite number of states defined by a user.
3. It is in one state at any time.
4. It changes from one state to another when there is a trigger event or a given condition, that is named transition.

FSM (3)

Essentially you formalize these concepts:

- when you have done these things wait for a certain time;
- when you have done these things wait for an event;
- when you have done these things wait for a sequence of events;
- when something happens do something:
  - move to another state;
  - stay in that state;
  - come back to one or few states.

In order to simplify, in this laboratory, we consider that for each state is associated an action or an output.

Basic example (1)

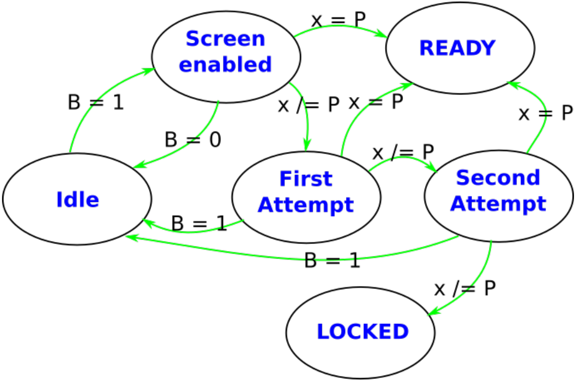
Unlocking of the smartphone.

The prevalent sequence of events for unlock it:

1. the smarthphone is in idle state;
2. press the button (B) to enable the screen. If B is re-pressed the screen is deactivated.
3. insert the code (P) once. If it is right the smartphone is unlocked and in a state ready for a new event. Else ..
4. insert the code (P) twice. If it is right the smartphone is unlocked and in a state ready for a new event. Else ..
5. insert the code (P) for the third time. If it is right the smartphone is unlocked and in a state ready for a new event. Else it goes in a permanent locked state.

Basic example (2)

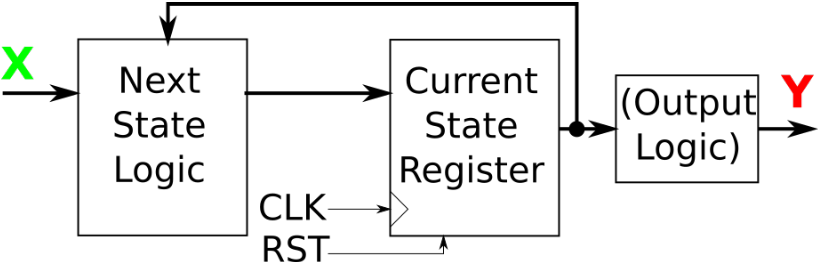
In order to simplify the state diagram, when the FSM is in idle state no attempts to insert the password were tried previously.



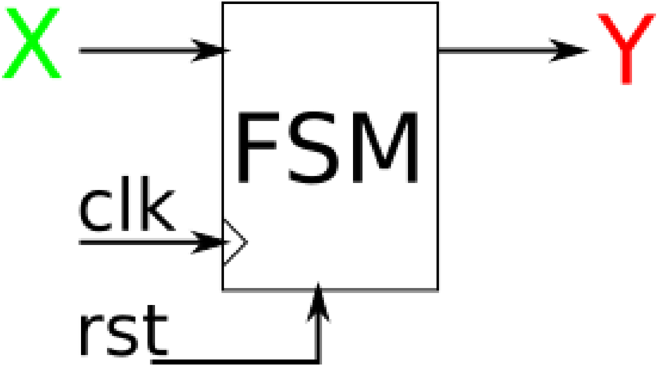
Outline

Moore Machine

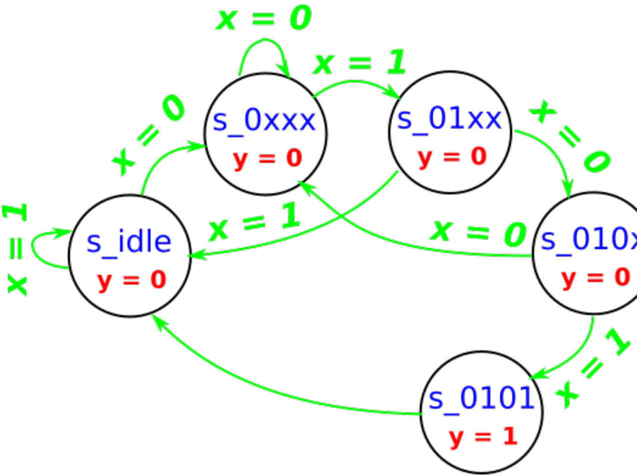
In this laboratory, only the Moore FSM machine is considered. Essentially in this type of FSM the output depends only on the current state of the machine itself.



The VHDL code given behaves like a Moore FSM. It can recognize a certain input sequence. In this case $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$. When this sequence is recognized the output goes high.



Practical example (2)



Notes on the code fsm.vhd (1)

- The code to define and declare the states of the FSM.

```
type state is (s_idle, s_0xxx, s_01xx, s_010x, s_0101);
signal state_curr, state_next: state;
```

- The code that describes the "Current State Register" block.

```
p_reg : process(clk,rst) is
begin
   if rst = '1' then
      state_curr <= s_idle;
   elsif rising_edge(clk) then
      state_curr <= state_next;
   end if;
end process;
```

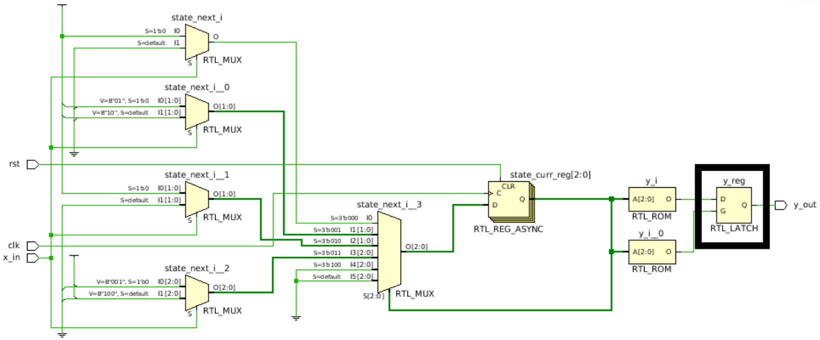
Notes on the code fsm.vhd (2)

- The code that describes the "Next State Logic" block.

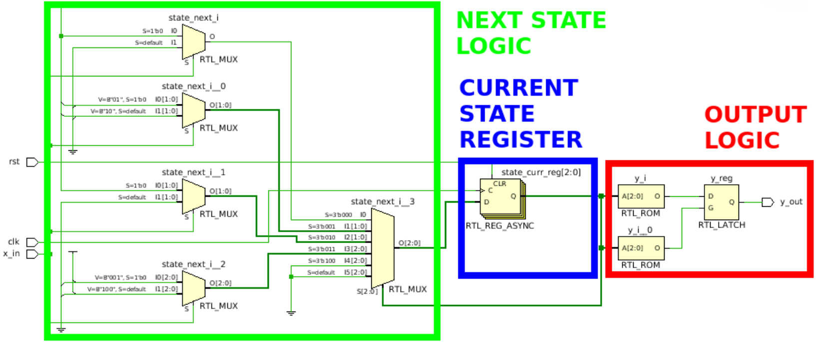
```
p_cmb : process(state_curr, x_in) is
begin
   case state_curr is
   when s_idle =>...
   when s_0xxx =>...
   when s_01xx =>...
   when s_010x =>...
   when s_0101 =>...
   when others =>...
   end case;
end process;
```

- The code that describes the "Output logic" block, is simply $y\_out <= y$.

fsm.vhd RTL Schematic (1)



fsm.vhd RTL Schematic (2)



Notes on the code fsm_1.vhd (1)

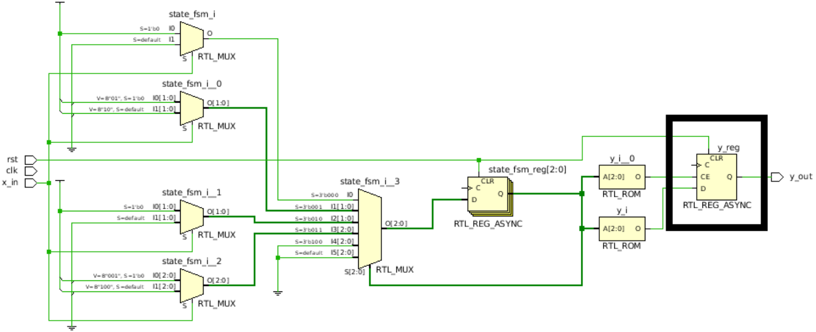
- Basically there is a more practical way to implement a Moore FSM in VHDL.
- Essentially the two processes "Current State Register" and "Next State Logic" are merged together.
- You have less code to write and just one signal state type.
- It is more flexible.

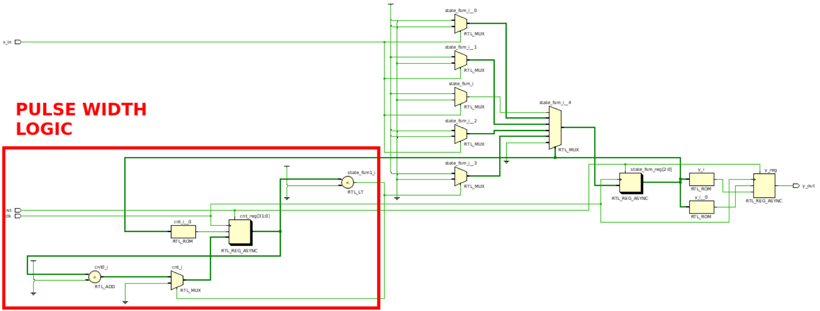
Notes on the code fsm_1.vhd (2)

This is the typical way to slow down the clock or in this case to produce a pulse with a width greater than one clock cycle.

```
p_fsm : process(clk,rst, x_in) is
variable cnt : integer;
begin
   if rst = '1' then
      state_fsm <= s_idle;
      y <= '0';
      cnt := 0;
   elsif rising_edge(clk) then
      case state_fsm is
      when s_idle =>...
      when s_0xxx =>...
      when s_01xx =>...
      when s_010x =>...
      when s_0101 =>
         if cnt < WTIME then
            cnt := cnt + 1;
            state_fsm <= s_0101;
         else
            cnt := 0;
            state_fsm <= s_idle;
         end if;
         y <= '1';
      when others =>
         state_fsm <= s_idle;
      end case;
   end if;
end process;
```

fsm_1.vhd RTL Schematic (1)



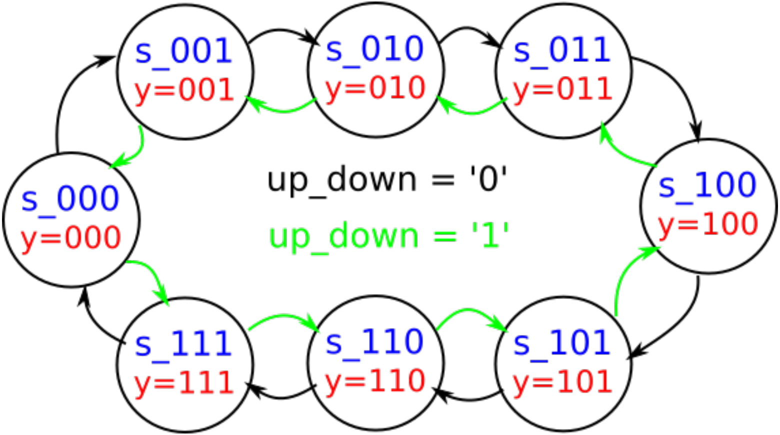

PULSE WIDTH
LOGIC
x_in
rst
clk
state_fsm_i__0
RTL_MUX
state_fsm_i__1
RTL_MUX
state_fsm_i
RTL_MUX
state_fsm_i__2
RTL_MUX
state_fsm_i__3
RTL_MUX
state_fsm_i__4
RTL_MUX
state_fsm1_i
RTL_LT
cnt_i__0
RTL_ROM
cnt_reg[31:0]
RTL_REG_ASYNC
cnt0_i
RTL_ADD
cnt_i
RTL_MUX
state_fsm_reg[2:0]
RTL_REG_ASYNC
y_i
RTL_ROM
y_i__0
RTL_ROM
y_reg
RTL_REG_ASYNC
y_out

Hints for your FSMs

- In the VHDL code that you write, use the more practical way to write a Moore FSM (i.e. fsm_1.vhd).
- Label the state in a consistent way. For example, **no** S_1, S_2, S_3 ...
- In each state is very good practice define the next state for each combination of the inputs.
- Use always **when others** $=>$.

Outline

A counter can be seen as a FSM, where each state represents a value of the counter.



Outline

Suggested exercises

- Implement the counter as a FSM, described by the state diagram of the slide 26. Then the counting has to be visible in the evaluation board leds.
- Write the code that describes a FSM that recognizes a sequence $0 \rightarrow 1 \rightarrow X \rightarrow 1 \rightarrow 1$, where X is a don't care condition.