HE VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

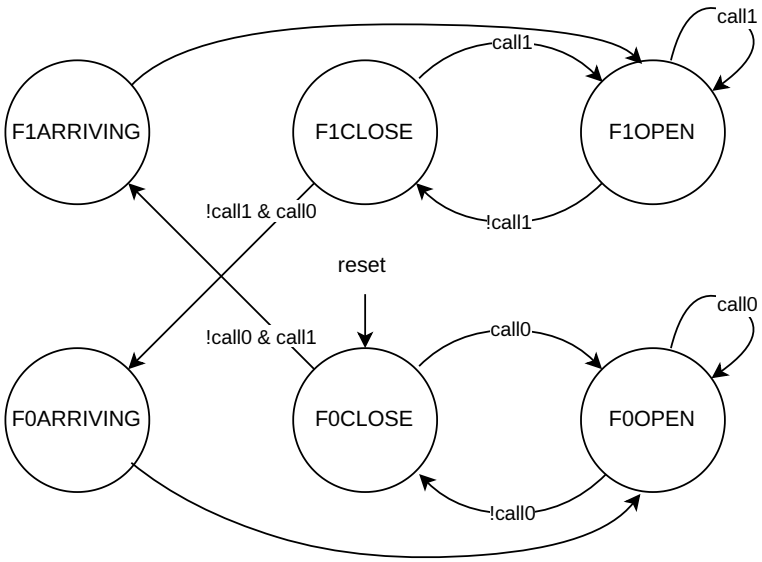
Formal verification
Verification of an elevator model
CERN training - July 2025

Context

We are interested in verifying the correctness of a finite state machine modeling the behavior of an elevator.
Its entity is the following :

```
entity elevator_fsm is
    port(
        clk_i    : in  std_logic;
        rst_i    : in  std_logic;
        call0_i  : in  std_logic;
        call1_i  : in  std_logic;
        open_o   : out std_logic;
        floor0_o : out std_logic;
        floor1_o : out std_logic
        );
end elevator_fsm;
```

Two inputs allow to call the elevator on floor 0 and floor 1. The call is only effective if the elevator has its door closed and is waiting on a floor. Calling the elevator from the floor it is on just opens the door. The door then stays open while the button of the floor is pressed.
When the elevator travels to another floor, it passes through a state where it is just arriving to the corresponding floor. At this moment both floor outputs are off, as the elevator is not really on a floor.
The asynchronous reset puts the elevator on floor 0, door closed.
The state machine corresponds to the following graph :



`door_o` is `'1'` in states `F0OPEN` and `F1OPEN`, else shall be `'0'`.
`floor0_o` is `'1'` in states `F0CLOSE` and `F0OPEN`, else shall be `'0'`.
`floor1_o` is `'1'` in states `F1CLOSE` and `F1OPEN`, else shall be `'0'`.
Add code in the `.psl` file in order to verify the correct FSM behavior. First do this using only the input/output ports. Second, as you have access to the architecture, and so, the state of the system, add assertions checking the FSM transitions.

A generic parameter `ERRNO` allows to inject errors in the design. Its behavior is the following :

1. When 0 the result is valid ;
2. When in the $[1, 6]$ interval, the result is unvalid.

This generic parameter allows to test your assertions by trying various `ERRNO` values
You can modify its value in the `.sby` file.
Use the code in folder `v1`.

Second version

In order to make our elevator FSM more realistic, we modified its behavior such as to let the door open 10 clock cycles. So, when the door opens, it stays open for 10 cycles. If the button of the current floor is pressed while the door is open, then the door stays open for 10 clock cycles from the moment the button is pressed.
The corresponding code is in folder `v2`. You can start with a copy-paste of your previous assertions, as some of them should still hold, and then modify what has to be.
A generic parameter `ERRNO` allows to inject errors in the design. Its behavior is the following :

1. When 0 the result is valid ;
2. When in the $[1, 6]$ interval, the result is unvalid.

Third version

The design team decided that having a counter inside the FSM was not OK, and so they decided to implement a counter outside of the FSM. Therefore they did add two ports to the FSM :

— `counter_init_o`, as output, to initialize the counter to 0
— `counter_done_i`, indicating that the counter reached the final value (after 10 clock cycles for instance)

Therefore, the behavior of the FSM has to act according to the `counter_done_i` signal, and act correctly on the `counter_init_o`.
We expect then the `counter_init_o` to be high just before entering into one of the *OPEN* states, and low elsewhere.
Also, the door should close when `counter_done_i` gets high.
The corresponding code is in folder `v3`. You can start with a copy-paste of your previous assertions, as some of them should still hold, and then modify what has to be.
One thing to keep in mind is that the FSM, as it is currently implemented does not allow to stay open only one clock cycle.
A generic parameter `ERRNO` allows to inject errors in the design. Its behavior is the following :

1. When 0 the result is valid ;
2. When in the $[1, 6]$ interval, the result is unvalid.