Le macchine a stati

Di seguito è riportato il *file* VHDL che descrive la macchina a stati *switch_machine*.

```
library ieee;
use ieee.std_logic_1164.all;

entity switch_machine is
 port ( reset, clk, and_out, go, counter: IN std_logic;
       my_prbs_rst, counter_reset, counter0_start, counter1_start, counter2_start, j_start: OUT std_logic;
       prbs_enable0, prbs_enable1, prbs_enable2: OUT std_logic);
end switch_machine;

architecture switch_machine of switch_machine is
type stato is ( s0, s1, s2, s3, s4, s5, s6);
signal state, next_state : stato;
signal my_prbs_rst_next, counter_reset_next, counter0_start_next, counter1_start_next, counter2_start_next,
j_start_next: std_logic;
signal prbs_enable0_next, prbs_enable1_next, prbs_enable2_next: std_logic;

begin
reg: process( clk, reset)
begin
 if reset='0' then
  state<=s0;
  my_prbs_rst<='0';
  counter_reset<='0';
  counter0_start<='0';
  counter1_start<='0';
  counter2_start<='0';
  j_start<='0';
  prbs_enable0<='0';
  prbs_enable1<='0';
  prbs_enable2<='0';
 elsif ( clk'event and clk='1' ) then
   state<=next_state;
   my_prbs_rst<=my_prbs_rst_next;
   counter_reset<=counter_reset_next;
   counter0_start<=counter0_start_next;
   counter1_start<=counter1_start_next;
   counter2_start<=counter2_start_next;
   j_start<=j_start_next;
   prbs_enable0<=prbs_enable0_next;
   prbs_enable1<=prbs_enable1_next;
   prbs_enable2<=prbs_enable2_next;
 end if;
end process reg;

comb: process ( and_out, go, state, counter)
begin
 case state is
  when s0 =>
    next_state<=s1;
    my_prbs_rst_next<='0';
    counter_reset_next<='1';
    counter0_start_next<='1';
    counter1_start_next<='1';
    counter2_start_next<='1';
    j_start_next<='0';
    prbs_enable0_next<='0';
    prbs_enable1_next<='0';
    prbs_enable2_next<='0';
  when s1 =>
   if ( counter='1') then
    next_state<=s2;
    my_prbs_rst_next<='1';
    counter_reset_next<='0';
    counter0_start_next<='0';
    counter1_start_next<='0';
    counter2_start_next<='0';
    j_start_next<='0';
    prbs_enable0_next<='0';
    prbs_enable1_next<='0';
    prbs_enable2_next<='0';
```

```
 else
  next_state<=s1;
  my_prbs_rst_next<='0';
  counter_reset_next<='1';
  counter0_start_next<='1';
  counter1_start_next<='1';
  counter2_start_next<='1';
  j_start_next<='0';
  prbs_enable0_next<='0';
  prbs_enable1_next<='0';
  prbs_enable2_next<='0';
 end if;
when s2 =>
 if ( and_out='1') then
  next_state<=s3;
  my_prbs_rst_next<='1';
  counter_reset_next<='0';
  counter0_start_next<='0';
  counter1_start_next<='0';
  counter2_start_next<='0';
  j_start_next<='1';
  prbs_enable0_next<='0';
  prbs_enable1_next<='0';
  prbs_enable2_next<='0';
 else
  next_state<=s2;
  my_prbs_rst_next<='1';
  counter_reset_next<='0';
  counter0_start_next<='0';
  counter1_start_next<='0';
  counter2_start_next<='0';
  j_start_next<='0';
  prbs_enable0_next<='0';
  prbs_enable1_next<='0';
  prbs_enable2_next<='0';
 end if;
when s3 =>
  next_state<=s4;
  my_prbs_rst_next<='1';
  counter_reset_next<='0';
  counter0_start_next<='0';
  counter1_start_next<='0';
  counter2_start_next<='0';
  j_start_next<='0';
  prbs_enable0_next<='0';
  prbs_enable1_next<='0';
  prbs_enable2_next<='0';
when s4 =>
 if ( go='1') then
  next_state<=s5;
  my_prbs_rst_next<='1';
  counter_reset_next<='0';
  counter0_start_next<='0';
  counter1_start_next<='0';
  counter2_start_next<='0';
  j_start_next<='0';
  prbs_enable0_next<='1';
  prbs_enable1_next<='1';
  prbs_enable2_next<='1';
 else
  next_state<=s0;
  my_prbs_rst_next<='0';
  counter_reset_next<='0';
  counter0_start_next<='0';
  counter1_start_next<='0';
  counter2_start_next<='0';
  j_start_next<='0';
  prbs_enable0_next<='0';
  prbs_enable1_next<='0';
  prbs_enable2_next<='0';
 end if;
when s5 =>
 if ( and_out='1') then
  next_state<=s6;
  my_prbs_rst_next<='1';
  counter_reset_next<='0';
  counter0_start_next<='0';
  counter1_start_next<='0';
  counter2_start_next<='0';
  j_start_next<='1';
  prbs_enable0_next<='0';
  prbs_enable1_next<='0';
```

```
    prbs_enable2_next<='0';
   else
    next_state<=s5;
    my_prbs_rst_next<='1';
    counter_reset_next<='0';
    counter0_start_next<='0';
    counter1_start_next<='0';
    counter2_start_next<='0';
    j_start_next<='0';
    prbs_enable0_next<='1';
    prbs_enable1_next<='1';
    prbs_enable2_next<='1';
   end if;
  when s6 =>
    next_state<=s0;
    my_prbs_rst_next<='0';
    counter_reset_next<='0';
    counter0_start_next<='0';
    counter1_start_next<='0';
    counter2_start_next<='0';
    j_start_next<='0';
    prbs_enable0_next<='0';
    prbs_enable1_next<='0';
    prbs_enable2_next<='0';
  end case;
 end process comb;
end switch_machine;
```

Di seguito è riportato il *file* VHDL che descrive la macchina a stati *j_machine*.

```
library ieee;
use ieee.std_logic_1164.all;

entity j_machine is
  port ( reset, clk, j_start: IN std_logic;
         j: IN std_logic_vector(2 downto 0);
        go: OUT std_logic;
        out1: OUT std_logic_vector(2 downto 0));
 end j_machine;

 architecture j_machine of j_machine is
 constant s0: std_logic_vector (2 downto 0) :="000";
 constant s1: std_logic_vector (2 downto 0) :="001";
 constant s2: std_logic_vector (2 downto 0) :="010";
 constant s3: std_logic_vector (2 downto 0) :="011";
 constant s4: std_logic_vector (2 downto 0) :="100";
 constant s5: std_logic_vector (2 downto 0) :="101";
 constant s6: std_logic_vector (2 downto 0) :="110";
 constant s7: std_logic_vector (2 downto 0) :="111";
 signal state: std_logic_vector (2 downto 0);

 begin
 reg: process( clk, reset)
 begin
  if reset='0' then
   state<=s0;
   go<='0';
  elsif ( clk'event and clk='1' ) then

  case state is
   when s0 =>
    if ( j_start='0') then
     state<=s0;
     go<='0';
    else
     if(j="111") then
       state<=s0;
       go<='0';
     elsif(j="110") then
       state<=s1;
       go<='1';
     elsif(j="101") then
       state<=s2;
       go<='1';
     elsif(j="100") then
       state<=s3;
       go<='1';
```

```
 elsif(j="011") then
   state<=s4;
   go<='1';
 elsif(j="010") then
   state<=s5;
   go<='1';
 elsif(j="001") then
   state<=s6;
   go<='1';
 elsif(j="000") then
   state<=s7;
   go<='1';
 end if;
end if;

when s1 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s1;
  go<='1';
 end if;

when s2 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s2;
  go<='1';
 end if;

when s3 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s3;
  go<='1';
 end if;

when s4 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s4;
  go<='1';
 end if;

when s5 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s5;
  go<='1';
 end if;

when s6 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s6;
  go<='1';
 end if;

when s7 =>
 if ( j_start='1') then
  state<=s0;
  go<='0';
 else
  state<=s7;
  go<='1';
 end if;

when others=>null;

end case;
end if;
```

```
end process reg;

 out1<=state;
end j_machine;
```