```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TAP_FSM is
    port(
           TMS_I : in STD_LOGIC; -- TMS desde FPGA
           RELOJ_TAP : in STD_LOGIC; -- reloj 25 MHz
           TDO_I : out STD_LOGIC; -- TDO hacia la FPGA
           TDI_I_DR : in STD_LOGIC; -- TDI DR desde la FPGA
           TDI_I_IR : in STD_LOGIC; -- TDI IR desde la FPGA
           TMS_O : out STD_LOGIC;
           TCK_O : out STD_LOGIC;
           TDO_O : in STD_LOGIC;
           TDI_O : out STD_LOGIC
     );
end TAP_FSM;

architecture Behavioral of TAP_FSM is
    -- Estados del TAP controller
     type state_type is (ST_Logic_Reset, ST_Run_Test,
ST_Select_DR, ST_Capture_DR, ST_Shift_DR, ST_Exit1_DR,
ST_Pause_DR, ST_Exit2_DR, ST_Update_DR, ST_Select_IR,
ST_Capture_IR, ST_Shift_IR, ST_Exit1_IR, ST_Pause_IR, ST_Exit2
_IR, ST_Update_IR);
     -- Registro para los estados del programa principal
     signal state   : state_type;

begin

    -- Proceso para cambiar el estado actual acorde a la
entrada
    process (RELOJ_TAP)
    begin
        TCK_O <= RELOJ_TAP;
        if rising_edge(RELOJ_TAP) then
            case state is
                -- Test Logic Reset
                when ST_Logic_Reset=>
                    if TMS_I = '1' then
                        state <= ST_Logic_Reset;
                    else
                        state <= ST_Run_Test;
                    end if;
                -- Run Test / Idle
                when ST_Run_Test=>
                    if TMS_I = '1' then
                        state <= ST_Select_DR;
```

```
    else
        state <= ST_Run_Test;
    end if;
-- Select DR Scan
when ST_Select_DR=>
    if TMS_I = '1' then
        state <= ST_Select_IR;
    else
        state <= ST_Capture_DR;
    end if;
-- Capture DR
when ST_Capture_DR =>
    if TMS_I = '1' then
        state <= ST_Exit1_DR;
    else
        state <= ST_Shift_DR;
    end if;
-- Shift DR
when ST_Shift_DR=>
    if TMS_I = '1' then
        state <= ST_Exit1_DR;
    else
        state <= ST_Shift_DR;
    end if;
-- Exit1 DR
when ST_Exit1_DR=>
    if TMS_I = '1' then
        state <= ST_Update_DR;
    else
        state <= ST_Pause_DR;
    end if;
-- Pause DR
when ST_Pause_DR=>
    if TMS_I = '1' then
        state <= ST_Exit2_DR;
    else
        state <= ST_Pause_DR;
    end if;
-- Exit2 DR
when ST_Exit2_DR =>
    if TMS_I = '1' then
        state <= ST_Update_DR;
    else
        state <= ST_Shift_DR;
    end if;
-- Update DR
when ST_Update_DR=>
    if TMS_I = '1' then
        state <= ST_Select_DR;
    else
        state <= ST_Run_Test;
    end if;
-- Select IR Scan
when ST_Select_IR=>
    if TMS_I = '1' then
        state <= ST_Logic_Reset;
    else
```

```
                        state <= ST_Capture_IR;
                    end if;
                -- Capture IR
                when ST_Capture_IR=>
                    if TMS_I = '1' then
                        state <= ST_Exit1_IR;
                    else
                        state <= ST_Shift_IR;
                    end if;
                -- Shift IR
                when ST_Shift_IR =>
                    if TMS_I = '1' then
                        state <= ST_Exit1_IR;
                    else
                        state <= ST_Shift_IR;
                    end if;
                -- Exit1 IR
                when ST_Exit1_IR=>
                    if TMS_I = '1' then
                        state <= ST_Update_IR;
                    else
                        state <= ST_Pause_IR;
                    end if;
                -- Pause IR
                when ST_Pause_IR=>
                    if TMS_I = '1' then
                        state <= ST_Exit2_IR;
                    else
                        state <= ST_Pause_IR;
                    end if;
                -- Exit2 IR
                when ST_Exit2_IR=>
                    if TMS_I = '1' then
                        state <= ST_Update_IR;
                    else
                        state <= ST_Shift_IR;
                    end if;
                -- Update IR
                when ST_Update_IR =>
                    if TMS_I = '1' then
                        state <= ST_Select_DR;
                    else
                        state <= ST_Run_Test;
                    end if;
            end case;
        end if;
    end process;

     -- Generacion de salidas acorde al estado actual
     process (state, TDI_I_IR, TDI_I_DR, TDO_O, TMS_I,
RELOJ_TAP)
     begin
        TMS_O <= TMS_I;
        TDO_I <= TDO_O;
        TDI_O <= '1';

          case state is
```

```
            -- Test Logic Reset
            when ST_Logic_Reset=>
            -- Run Test / Idle
            when ST_Run_Test=>
            -- Select DR Scan
            when ST_Select_DR=>
            -- Capture DR
            when ST_Capture_DR =>
            -- Shift DR
            when ST_Shift_DR=>
                TDI_O <= TDI_I_DR;
            -- Exit1 DR
            when ST_Exit1_DR=>
            -- Pause DR
            when ST_Pause_DR=>
            -- Exit2 DR
            when ST_Exit2_DR =>
            -- Update DR
            when ST_Update_DR=>
            -- Select IR Scan
            when ST_Select_IR=>
            -- Capture IR
            when ST_Capture_IR=>
            -- Shift IR
            when ST_Shift_IR =>
                TDI_O <= TDI_I_IR;
            -- Exit1 IR
            when ST_Exit1_IR=>
            -- Pause IR
            when ST_Pause_IR=>
            -- Exit2 IR
            when ST_Exit2_IR=>
            -- Update IR
            when ST_Update_IR =>
        end case;
      end process;

end Behavioral;
```