```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 11/09/2021 08:25:48 PM
-- Design Name:
-- Module Name: fsm - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 fsm is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           x : in STD_LOGIC;
           z : out STD_LOGIC
           );
end fsm;

architecture Behavioral of fsm is
    type statename is (stateA, stateB, stateC, stateD, stateE, stateF, stateG);
    signal currentstate, nextstate : statename;

begin

    process(clk, reset)
    begin
        if reset = '1' then
            currentstate <= stateA;
        elsif rising_edge(clk) then
            currentstate <= nextstate;
        end if;
    end process;

    process(currentstate,x)
    begin
        case currentstate is
```

```
        when stateA =>
            if x = '1' then
                nextstate <= stateB;
            else
                nextstate <= stateA;
            end if;
        when stateB =>
            if x = '1' then
                nextstate <= stateB;
            else
                nextstate <= stateC;
            end if;
        when stateC =>
            if x = '1' then
                nextstate <= stateD;
            else
                nextstate <= stateA;
            end if;
        when stateD =>
            if x = '1' then
                nextstate <= stateE;
            else
                nextstate <= stateG;
            end if;
          when stateE =>
            if x = '1' then
                nextstate <= stateE;
            else
                nextstate <= stateF;
            end if;
          when stateF =>
            if x = '1' then
                nextstate <= stateA;
            else
                nextstate <= stateD;
           end if;
         when stateG =>
            if x = '1' then
                nextstate <= stateD;
            else
                nextstate <= stateG;
             end if;
        when others =>
            nextstate <= stateA;
    end case;
end process;

 process(currentstate)
 begin
     case currentstate is
         when stateA =>
             z <= '0';
         when stateB =>
             z <= '0';
         when stateC =>
             z <= '0';
         when stateD =>
             z <= '1';
         when stateE =>
```

```
                z <= '1';
            when stateF =>
                z <= '1';
            when stateG =>
                z <= '1';
            when others =>
                z <= '0';
        end case;
    end process;

end Behavioral;
```