```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 SPI_Controller is
  generic ( freq_SCLK: integer := 5_000_000;
            n: integer := 8);
  Port ( Clk: in std_logic;
         Rst: in std_logic;
         Start: in std_logic;
         TxData: in std_logic_vector (n - 1 downto 0);
         MOSI: out std_logic;
         MISO: in std_logic;
         RxData: out std_logic_vector (n - 1 downto 0);
         SCLK: out std_logic;
         SS: out std_logic;
         TxRdy: out std_logic;
         RxRdy:out std_logic);
end SPI_Controller;

architecture Behavioral of SPI_Controller is

constant freq_Clk: integer := 100_000_000;
constant freq_div: integer := freq_Clk / freq_SCLK;

signal Start1, Start_int, RstStart, LdTxRx, ShTxRx, SCLK_int, SclkEn, Sin, CE_p, CE_n :
std_logic := '0';
signal TxData_reg_out, RxData_reg_out, TxRx_reg_out : std_logic_vector (n - 1 downto 0)
:= (others => '0');

type Stare is (idle, load, tx_rx, bit0, ready);
signal State: Stare;

attribute keep: STRING;
attribute keep of Start1: signal is "TRUE";
attribute keep of Start_int: signal is "TRUE";
attribute keep of CE_p: signal is "TRUE";
attribute keep of CE_n: signal is "TRUE";
attribute keep of State: signal is "TRUE";
```

```
attribute keep of TxData_reg_out: signal is "TRUE";
attribute keep of TxRx_reg_out: signal is "TRUE";

begin

Start1 <= Start;

SCLK_Generator: process(Clk)

variable counter: integer := 0;

begin
   CE_P <= '0';
   CE_n <= '0';
   if rising_edge(Clk) then
      counter := counter + 1;
      if counter = freq_div / 2 then
         SCLK_int <= not SCLK_int;
         CE_p <= '1';
      elsif counter = freq_div then
         SCLK_int <= not SCLK_int;
         CE_n <= '1';
         counter := 0;
      end if;
   end if;
end process;

SCLK <= SCLK_int when SclkEn = '1';

Start_reg: process(Clk, RstStart)
begin
   if rising_edge(Clk) then
      if RstStart = '1' then
         Start_int <= '0';
      else
         Start_int <= Start1;
      end if;
   end if;
end process;

Rx_reg: process(Clk)
begin
   if rising_edge(Clk) then
      if CE_p = '1' then
         if Rst = '1' then
            Sin <= '0';
         else
            Sin <= MISO;
```

```
            end if;
        end if;
    end if;
end process;

TxData_reg: process(Clk)
begin
    if rising_edge(Clk) then
        if Start1 = '1' then
            TxData_reg_out <= TxData;
        end if;
    end if;
end process;

TxRx_reg: process(Clk)
begin
    if rising_edge(Clk) then
        if CE_n = '1' then
            if Rst = '1' then
                TxRx_reg_out <= (others => '0');
            elsif LdTxRx = '1' then
                TxRx_reg_out <= TxData_reg_out;
            elsif ShTxRx = '1' then
                TxRx_reg_out <= TxData_reg_out(n - 2 downto 0) & Sin;
                MOSI <= TxRx_reg_out(n - 1);
            end if;
        end if;
    end if;
end process;

RxData <= TxRx_reg_out;

UC: process (Clk)

variable CntBit: integer := 0;
attribute keep of CntBit: variable is "TRUE";

begin
    if rising_edge(Clk) then
        if CE_n = '1' then
            if Rst = '1' then
                State <= idle;
            else
                case State is
                    when idle =>
                        if Start_int = '1' then
                            State <= load;
                        end if;
```

```
                when load =>
                    CntBit := n - 1;
                    State <= tx_rx;
                when tx_rx =>
                    if CntBit = 0 then
                        State <= bit0;
                    else
                        CntBit := CntBit - 1;
                    end if;
                when bit0 =>
                    State <= ready;
                when ready =>
                    CntBit := n - 1;
                    if Start_int = '0' then
                        State <= idle;
                    else
                        State <= tx_rx;
                    end if;
            end case;
        end if;
    end if;
  end if;
end process;

LdTxRx <= '1' when State = load else '0';
SS <= '1' when State = idle else '0';
RstStart <= '1' when State = load else '0';
ShTxRx <= '1' when State = tx_rx else '0';
SclkEn <= '1' when State = tx_rx else '0';
TxRdy <= '1' when State = bit0 else '0';
RxRdy <= '1' when State = ready else '0';

end Behavioral;
```