File: C:/My_Designs/Lab 8/lab8/task1/src/spi_tx_shifter.vhd

```
1   ----------------------------------------------------------------------------
    -----
2   --
3   -- Title       : spi_tx_shifter
4   -- Design      : task1_2
5   -- Author      : Ishabul Haque and Ken Ejinkonye
6   -- Company     : Stony Brook
7   --
8   -- Description : The transmitter shifter spi_tx_shifter converts the
    parallel
9   -- data byte data_in to serial and transmits its data bits along with a
10  -- synchronous clock sck. An SPI slave uses sck to determine when to
    sample
11  -- each serial data bit. We will be transmitting the data's most
    significant
12  -- bit.
13
14
15
16  -- List of Circuit Features To Be Verified:
17  -- Inputs: rst_bar, clk, send_en, cpha,cpool,data_in
18  -- Inputs cpol and cpha determine the clock polarity and clock phase of
    the
19  -- transmitted dat.cpol determines whether the clock idles at 0 or 1. cpha

20  -- determines whether the data must be sampled on the leading or trailing
21  -- edge of the shift clock.The values of cpol and cpha are set to be
22  -- compatible with the SPI slave that receives the data.
23
24  -- Outputs: txd,sck, ss_bar
25  -- Output sck is used as the sample clock by the slave.
26
27  ----------------------------------------------------------------------------
    -----
28
29  library ieee;
30  use ieee.numeric_std.all;
31  use ieee.std_logic_1164.all;
32  library work;
33  use work.all;
34
35  entity spi_tx_shifter is
36      port(
37      rst_bar : in std_logic;     -- asynnchronous system reset
38      clk : in std_logic;         -- system clock
39      send_en : in std_logic;         -- enable data transmission
40      cpha : in std_logic;        -- clock phase
41      cpol : in std_logic;        -- clock polarity
42      data_in : in std_logic_vector(7 downto 0); -- data to send
43      txd: out std_logic;         -- serial output data
44      sck: out std_logic;         -- synchronous shift clock
45      ss_bar: out std_logic       -- slave select
46      );
47
48  end spi_tx_shifter;
49
50  architecture fsm of spi_tx_shifter is
```

```
51
52
53   type state is (idle, ph1, ph2);
54   signal present_state, next_state: state;
55   signal bit_addr : unsigned(2 downto 0);
56
57   begin
58
59
60       state_reg: process(clk, rst_bar)
61       begin
62           -- Present State will be idle when
63           -- reset bar is triggered
64           if rst_bar = '0' then
65               present_state <= idle;
66           -- Present state will change to the
67           -- next state only on the rising
68           -- edge of clock
69           elsif rising_edge(clk) then
70               present_state <= next_state;
71           end if;
72       end process;
73
74       -- Only when send_en is high is when next state goes
75       -- to phase 1, otherwise it goes to phase 2 from idle
76       -- When bit adder is 000, it goes back to idle.
77       nxt_state: process (present_state, send_en, bit_addr)
78       begin
79           case present_state is
80               when idle =>
81               --
82               if send_en = '1' then
83                   next_state <= ph1;
84               else
85                   next_state <= ph2;
86               end if;
87
88               when ph1 =>
89               next_state <= ph2;
90
91               when ph2 =>
92               if bit_addr = "000" then
93                   next_state <= idle;
94               else
95                   next_state <= ph1;
96               end if;
97           end case;
98       end process;
99
100      -- Values to be assigned to the output signals
101      -- based on the current state and value of
102      -- bit adder
103      output: process (present_state, data_in, bit_addr)
104      begin
105          case present_state is
106
107              when idle=>
```

```
108             sck <= cpol;
109             ss_bar <= '1';
110             txd <= data_in(to_integer(bit_addr));
111             when ph1=>
112             sck<=cpol;
113             ss_bar<='0';
114             txd<= data_in(to_integer(bit_addr));
115             when ph2=>
116             sck <= not cpol;
117             ss_bar <= '0';
118             txd <= data_in(to_integer(bit_addr));
119         end case;
120     end process;
121 
122     -- Bit counter is used to implement shifting by using count to
123     -- select which bit from the parallel input data is output
124     bit_counter: process (rst_bar, clk, present_state)
125     begin
126         if rst_bar = '0' or present_state = idle then
127             bit_addr <= "111";
128         elsif rising_edge(clk) then
129             if present_state = ph2 then
130                 if bit_addr /= "000" then
131                     bit_addr <= bit_addr - 1;
132                 end if;
133             end if;
134         end if;
135     end process;
136 
137 
138 end fsm;
139 
140 
141 
```