Lab 8: Simple SPI Transmitter

Ishabul Haque 111598085
Ken Ejinkonye 112011486

Section 2
04/10/20

```
1   ------------------------------------------------------------------------
    ----
2   --
3   -- Title       : send_pos_edge_det
4   -- Design      : task1_2
5   -- Author      : Ishabul Haque and Ken Ejinkonye
6   -- Company     : Stony Brook
7   --
8   -- Description : A positive edge detector is used to detect the
9   -- positive edge of the send and to generate a narrow pulse that will
10  -- be used by the spi_tx_shifter to determine when it should start to send
11  -- a byte.
12  -- List of Circuit Features To Be Verified:
13  -- Inputs: rst_bar, clk, send
14  -- Outputs: send_en
15
16
17  ------------------------------------------------------------------------
    ----
18
19  library IEEE;
20  use IEEE.std_logic_1164.all;
21
22  entity send_pos_edge_det is
23      port(
24      rst_bar : in std_logic;     -- asynnchronous system reset
25      clk : in std_logic;         -- system clock
26      send : in std_logic;        -- debounced send input
27      send_en : out std_logic     -- send enable output pulse
28      );
29  end send_pos_edge_det;
30
31
32  architecture moore_fsm of send_pos_edge_det is
33      type state is (state_a, state_b, state_c);
34      signal present_state, next_state : state;
35
36      begin
37          state_reg: process (clk,rst_bar)
38          begin
39              if rst_bar = '0' then
40              -- If rst_bar is enabled, then present state will switch
41              -- to state a
42                  present_state <= state_a;
43              elsif rising_edge(clk) then
44              -- If rst_bar is not enabled, the present state will switch
45              -- to the next state on the rising edge of clock
46                  present_state <= next_state;
47              end if;
48          end process;
49
50          outputs: process (present_state)
51          begin
52              case present_state is
53                  -- Output send_en will only be high when the present state
54                  -- is in state c, otherwise it will output low
55                  when state_c => send_en <= '1';
```

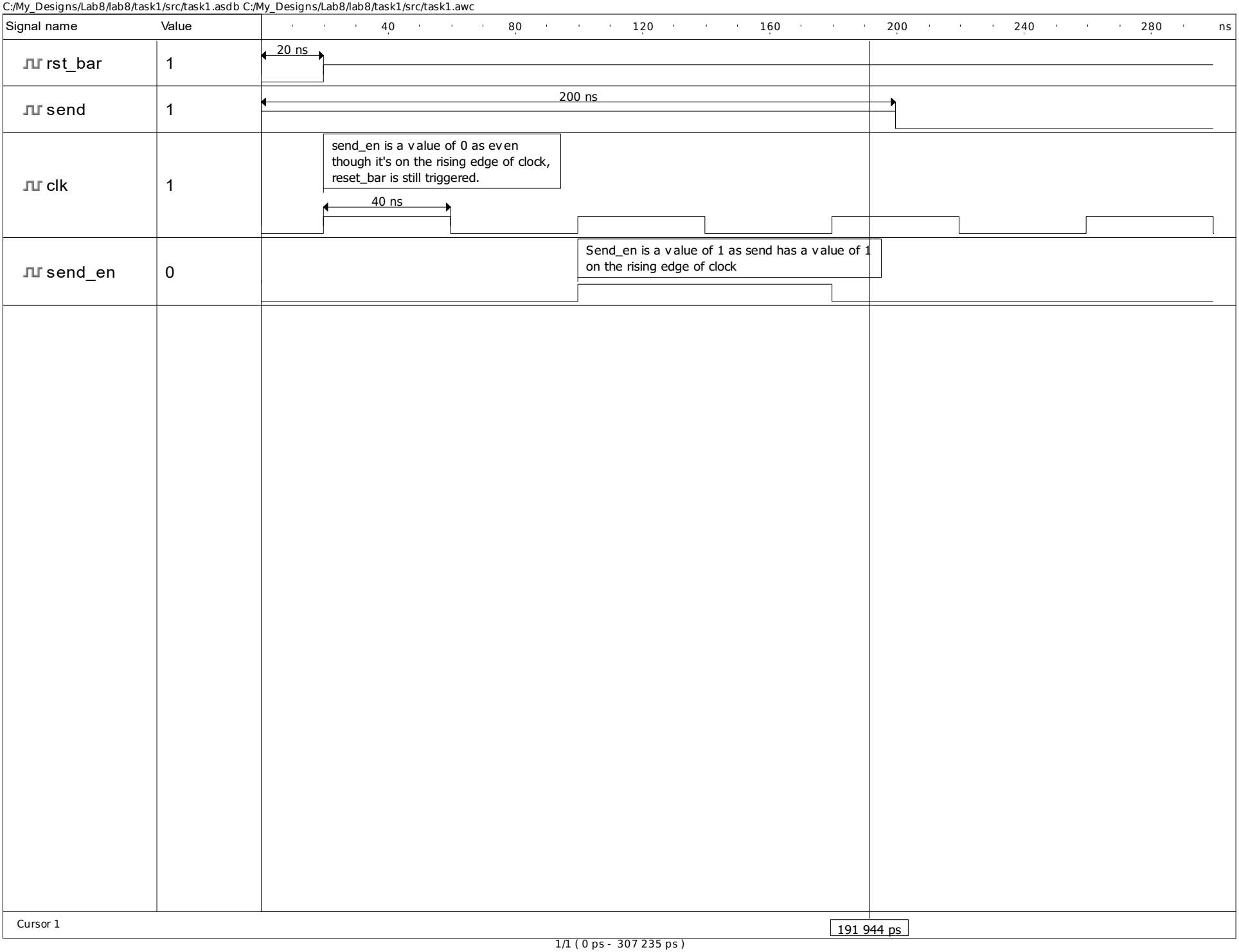
```
56                  when others => send_en <= '0';
57              end case;
58          end process;
59
60          nxt_state: process (present_state, send)
61          begin
62              -- Moore FSM, state values are determined by
63              -- state diagram
64              case present_state is
65                  when state_a =>
66                  if send = '0' then
67                      next_state <= state_b;
68                  else
69                      next_state <= state_b;
70                  end if;
71
72                  when state_b =>
73                  if send = '1' then
74                      next_state <= state_c;
75                  else
76                      next_state <= state_b;
77                  end if;
78
79                  when others =>
80                  if send = '0' then
81                      next_state <= state_b;
82                  else
83                      next_state <= state_a;
84                  end if;
85              end case;
86          end process;
87      end moore_fsm;
88
89
90
```

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

```vhdl
1   ----------------------------------------------------------------------------
    ----
2   --
3   -- Title       : send_pos_edge_det_tb
4   -- Design      : task1_2
5   -- Author      : Ishabul Haque and Ken Ejinkonye
6   -- Company     : Stony Brook
7   --
8   -- Description : Non self checking testbench for send_pos_edge_det design.
9   --
10  ----------------------------------------------------------------------------
    ----
11
12
13  library ieee;
14  use ieee.std_logic_1164.all;
15  use ieee.numeric_std.all;
16  library work;
17  use work.all;
18
19
20
21  entity send_pos_edge_det_TB is
22
23
24  end send_pos_edge_det_TB;
25
26  architecture tb_architecture of send_pos_edge_det_TB is
27
28
29         --stimulus signals
30      signal rst_bar : std_logic;
31      signal send : std_logic;
32      signal clk : std_logic;
33          --observed signals
34      signal send_en :  std_logic;
35
36      constant period : time := 20ns; --need a much longer period
37                                      -- in actual hardware
38
39  begin
40      -- Unit Under Test port map
41      UUT: entity send_pos_edge_det
42      port map (
43          clk => clk,
44          rst_bar => rst_bar,
45          send => send,
46          send_en => send_en
47          );
48
49      rst_bar <= '0', '1' after period;   -- reset
50
51      clock: process                      -- system clock
52      begin
53          -- clock starts at 0 for 0.5 clock periods
54          for i in 0 to 28 loop
55              wait for period;
```

```
56              clk <= not clk;             -- 28 rising edges
57              wait for period;
58          end loop;                       -- stop clock
59          std.env.finish;
60      end process;
61
62      -- Duration of signal send is 200ns at a high value
63      snd: process
64      begin
65          for i in 0 to 28 loop      --28 rising edges
66              wait for 200ns;        -- Send pulse duration of 200ns
67              send <= not send;
68          end loop;                  --end pulse
69
70          std.env.finish;
71          end process;
72
73  end tb_architecture;
74
```

C:/My_Designs/Lab8/lab8/task1/src/task1.asdb C:/My_Designs/Lab8/lab8/task1/src/task1.awc



1/1 (0 ps - 307 235 ps)

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=>
```

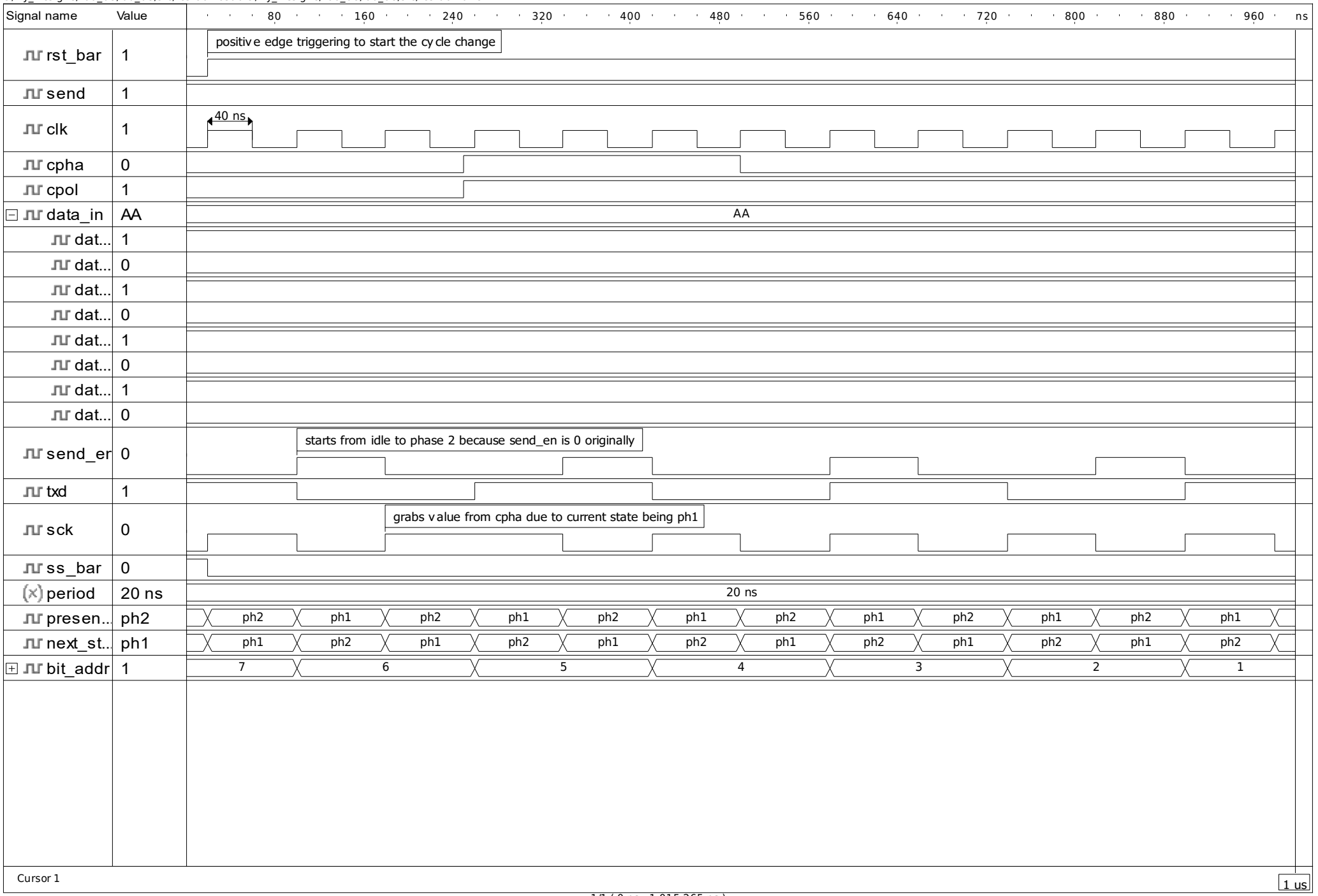
```
108             sck <= cpol;
109             ss_bar <= '1';
110             txd <= data_in(to_integer(bit_addr));
111             when ph1=>
112             sck<=cpol;
113             ss_bar<='0';
114             txd<= data_in(to_integer(bit_addr));
115             when ph2=>
116             sck <= 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 
```

File: C:/My_Designs/Lab 8/lab8/task1/src/task2_tb.vhd *

```
1   ----------------------------------------------------------------------------
    ----
2   --
3   -- Title       : task2_tb
4   -- Design      : task1_2
5   -- Author      : Ishabul Haque and Ken Ejinkonye
6   -- Company     : Stony Brook
7   --
8   -- Description : Non-self checking testbench for spi_tx_shifter
9   ----------------------------------------------------------------------------
    ----
10
11
12  library ieee;
13  use ieee.std_logic_1164.all;
14  use ieee.numeric_std.all;
15  library work;
16  use work.all;
17
18
19  entity task2_tb is
20      -- Generic declarations of the tested unit
21
22
23
24
25  --}} End of automatically maintained section
26  end task2_tb;
27
28  architecture tb_architecture of task2_tb is
29
30
31         --stimulus signals
32      signal rst_bar : std_logic;
33      signal send : std_logic := '1';
34      signal clk : std_logic := '0';
35      signal cpha: std_logic :='0';
36      signal cpol: std_logic := '0';
37      signal data_in: std_logic_vector(7 downto 0);
38
39      --observed signals
40      signal send_en: std_logic;
41      signal txd: std_logic;
42      signal sck: std_logic;
43      signal ss_bar: std_logic;
44
45
46      constant period : time := 20ns; --need a much longer period
47                                      -- in actual hardware
48
49  begin
50      -- Unit Under Test port map
51      UUT: entity spi_tx_shifter
52      port map (
53          rst_bar => rst_bar,
54          clk => clk,
55          send_en => send_en,
```

```
56          cpha => cpha,
57          cpol => cpol,
58          data_in => data_in,
59          txd => txd,
60          sck => sck,
61          ss_bar => ss_bar
62          );
63
64      -- Port mapping for send_en from send_pos_edge_det
65      -- design to spi_tx_shifter design
66      u0 : entity send_pos_edge_det
67      port map(
68      rst_bar => rst_bar, clk => clk,
69      send_en => send_en,
70      send=>send);
71      rst_bar <= '0', '1' after period * 3;   -- reset
72
73      clock: process                      -- system clock
74      begin
75          -- clock starts at 0 for 0.5 clock periods
76          for i in 0 to 28 loop
77              wait for period;
78              clk <= not clk;             -- 28 rising edges
79              wait for period;
80          end loop;                       -- stop clock
81          std.env.finish;
82      end process;
83
84
85
86
87  end tb_architecture;
88
```

c:/My_Designs/Lab_08/lab_08/src/test002.asdb c:/My_Designs/Lab_08/lab_08/src/test002.awc



1/1 (0 ps - 1 015 265 ps)

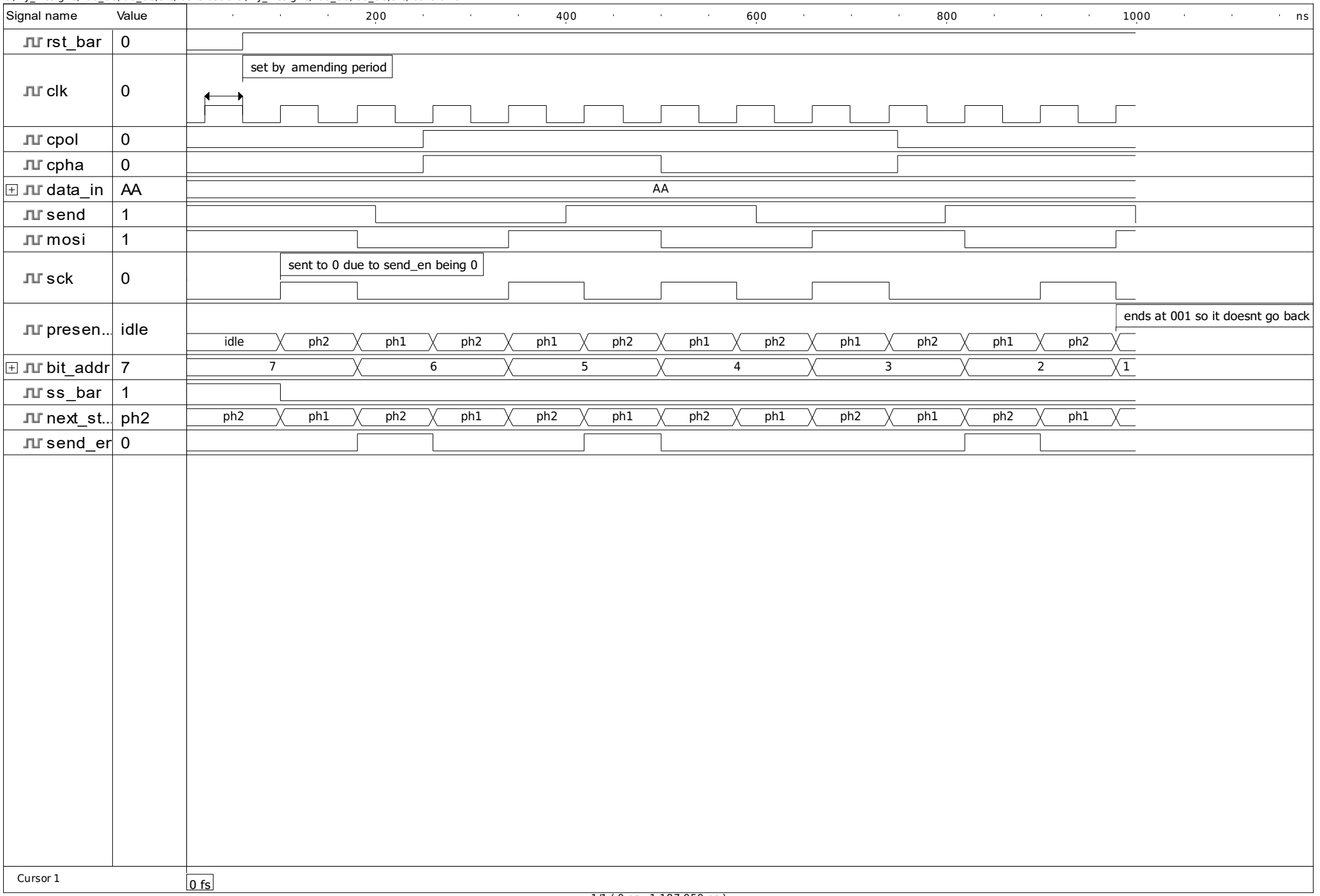
```
1   ------------------------------------------------------------------------
    ----
2   --
3   -- Title       : SPI_test_system
4   -- Design      : task1_2
5   -- Author      : Ishabul Haque and Ken Ejinkonye
6   -- Company     : Stony Brook
7   --
8   -- Description : The top-level structure simply combines the two previous
    entities
9   -- send_pos_edge_det and spi_tx_shifter.
10  -- List of Circuit Features to be verified:
11  -- Inputs: rst_bar, clk, send, cpol, cpha, data_in
12  -- Outputs: mosi, sck, ss_bar
13  -- Observant Stimulus: Presennt State, Next State
14  ------------------------------------------------------------------------
    ----
15
16  library IEEE;
17  use IEEE.std_logic_1164.all;
18  library work;
19  use work.all;
20
21  entity SPI_test_system_I is
22      port(
23      rst_bar : in std_logic;     -- asynnchronous system reset
24      clk : in std_logic;         -- system clock
25      send : in std_logic;        -- debounced send input
26      cpol : in std_logic;        -- clock poolarity settinng
27      cpha : in std_logic;        -- clock phase setting
28      data_in : in std_logic_vector(7 downto 0);  -- parallel input data
29      mosi: out std_logic;        -- master out slave in SPI serial data
30      sck: out std_logic;         -- SPI shift clock to slave
31      ss_bar : out std_logic      -- SPI shift clock to slave
32      );
33  end SPI_test_system_I;
34
35  architecture structural of SPI_test_system_I is
36
37  signal temp_send_en : std_logic;
38
39  begin
40
41
42  -- Port Map for first two designs to structural design SPI_test_system
43  u0 : entity send_pos_edge_det port map(rst_bar => rst_bar, clk => clk,
44      send => send, send_en => temp_send_en);
45
46  u1: entity spi_tx_shifter port map(rst_bar => rst_bar, clk => clk,
47      cpol => cpol, cpha => cpha, data_in => data_in, sck => sck,
48      ss_bar => ss_bar, send_en => temp_send_en, txd => mosi);
49
50
51
52
53  end structural;
```

File: C:\My_Designs\Lab 8\lab8\task1\src\task3_tb.vhd *

```
1   ------------------------------------------------------------------------
    ----
2   --
3   -- Title       : task3_tb
4   -- Design      : task1_2
5   -- Author      : Ishabul Haque and Ken Ejinkonye
6   -- Company     : Stony Brook
7   --
8   -- Description : Non self checking test bench for SPI_test_system
9   ------------------------------------------------------------------------
    ----
10
11
12  library ieee;
13  use ieee.std_logic_1164.all;
14  use ieee.numeric_std.all;
15  library work;
16  use work.all;
17
18
19
20  entity task3_tb is
21      -- Generic declarations of the tested unit
22
23
24
25
26  --}} End of automatically maintained section
27  end task3_tb;
28
29  architecture tb_architecture of task3_tb is
30
31
32         --stimulus signals
33      signal rst_bar : std_logic;
34      signal send : std_logic := '1';
35      signal cpha : std_logic;
36      signal cpol: std_logic;
37      signal clk : std_logic := '0';
38
39      signal data_in: std_logic_vector(7 downto 0);
40
41      --observed signals
42      signal send_en: std_logic;
43      signal txd: std_logic;
44      signal sck: std_logic;
45      signal ss_bar: std_logic;
46      signal present_state:std_logic;
47      signal next_state:std_logic;
48      signal mosi : std_logic;
49
50      constant period : time := 20ns; --need a much longer period
51                                      -- in actual hardware
52
53  begin
54      -- Unit Under Test port map
55
```

```
56      u0 : entity send_pos_edge_det port map(rst_bar => rst_bar, clk => clk,
57      send => send, send_en => send_en);
58
59      u1: entity spi_tx_shifter port map(rst_bar => rst_bar, clk => clk,
60      cpol => cpol, cpha => cpha, data_in => data_in, sck => sck,
61      ss_bar => ss_bar, send_en => send_en, txd => mosi);
62
63      rst_bar <= '0', '1' after period * 3;   -- reset
64
65      clock: process              -- system clock
66      begin
67                      -- clock starts at 0 for 0.5 clock periods
68          for i in 0 to 28 loop
69              wait for period;
70              clk <= not clk;     -- 15 rising edges
71              wait for period;
72          end loop;                   -- stop clock
73          std.env.finish;
74      end process;
75
76      snd: process
77      begin
78          for i in 0 to 28 loop      --28 rising edges
79              wait for 200ns;        -- Send pulse duration of 200ns
80              send <= not send;
81          end loop;                  --end pulse
82
83          std.env.finish;
84          end process;
85
86
87
88  end tb_architecture;
89
```

C:/My_Designs/Lab_08/lab_08/src/task3.asdb C:/My_Designs/Lab_08/lab_08/src/task3.awc



1/1 (0 ps - 1 187 959 ps)

1. How long is the send_en pulse generated by send_pos_edge_det? State this time duration as both number of clock pulses and as ns.
    a. The send_en pulse generated by the designs task is 80ns long and is the duration of two clock pulses
2. What is the frequency relationship between the SPI clock sck and the system clock clk?
    a. The frequency of SPI clock sck is half of system clock clk.
3. From the log file determine the default state assignment encoding used by the synthesizer for your FSMs.