```vhdl
-------------------------------------------------------------------------------
--
-- Title       : spi_mux_digit_driver
-- Design      : prelab11
-- Author      : Dongyun Lee
-- Company     : Stony Brook University
--
-------------------------------------------------------------------------------
--
-- File        : \\Mac\Home\Documents\SBU 2024 Spring\ESE 382\Prelab11\Prelab11\prelab11\src\spi_mux_digit_driver.vhd
-- Generated   : Tue Apr 30 15:39:45 2024
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.22
--
-------------------------------------------------------------------------------
--
-- Description :
--
-------------------------------------------------------------------------------

--------------------slv_spi_rx_shifter------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity slv_spi_rx_shifter is
    port(
        rxd        : in  std_logic;           -- Data received from master
        rst_bar    : in  std_logic;           -- Asynchronous reset
        sel_bar    : in  std_logic;           -- Selects shifter for operation
        clk        : in  std_logic;           -- System clock
        shift_en   : in  std_logic;           -- Enable shift
        rx_data_out: out std_logic_vector(7 downto 0) -- Received data
    );
end entity slv_spi_rx_shifter;


architecture slv_spi_rx_shifter of slv_spi_rx_shifter is
begin
    shift: process (clk, rst_bar)
    -- variable memory : unsigned(7 downto 0);
    begin
        if rst_bar = '0' then
            rx_data_out <= (others => '0');
        elsif rising_edge(clk) then
            if sel_bar = '0' then
                if (shift_en = '1') and (rxd = '1' or rxd = '0') then
                    rx_data_out <= rx_data_out(6 downto 0) & rxd;
                end if;
            end if;

        end if;

    end process;
end slv_spi_rx_shifter;


--------------------edge_det-----------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity edge_det is
    port(
        rst_bar   : in  std_logic; -- Asynchronous system reset
        clk       : in  std_logic; -- System clock
        sig       : in  std_logic; -- Input signal
        pos       : in  std_logic; -- '1' for positive edge, '0' for
negative
        sig_edge  : out std_logic  -- High for one system clk after edge
detected
    );
end entity edge_det;


architecture moore_fsm of edge_det is
type state is (state_a, state_b, state_c);
signal present_state, next_state : state;
begin
    -- first state : detects rst_bar
    state_reg: process (clk, rst_bar)
    begin
        if rst_bar = '0' then
            present_state <= state_a;
        elsif rising_edge(clk) then
            present_state <= next_state;
        end if;
    end process;

    -- process where it outputs
    outputs: process (present_state)
    begin
        case present_state is
            when state_c => sig_edge <= '1';
            when others => sig_edge <= '0';
        end case;
    end process;

    nxt_state: process (present_state, sig)
    begin
        case present_state is
            when state_a =>
            if (pos = '1' and sig = '0') or (pos = '0' and sig = '1') then
                next_state <= state_b;
            else
                next_state <= state_a;
            end if;

            when state_b =>
            if (pos = '1' and sig = '1') or (pos = '0' and sig = '0') then
                next_state <= state_c;
            else
                next_state <= state_b;
            end if;

            when others =>
            if (pos = '1' and sig = '0') or (pos = '0' and sig = '1') then
                next_state <= state_b;
            else
                next_state <= state_a;
            end if;
        end case;
    end process;

end moore_fsm;


--------------------rx_buff_reg--------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rx_buff_reg is
    port (
        rst_bar : in std_logic; -- asynchronous reset
        clk : in std_logic; -- system clock
        load_en : in std_logic; -- enable shift
        rx_buff_in : in std_logic_vector(7 downto 0); -- received data in
        rx_buff_out : out std_logic_vector(7 downto 0) -- received data
out
    );
end rx_buff_reg;

architecture Behavioral of rx_buff_reg is
begin
    double_buffer : process (clk, rst_bar)
    begin
        if rst_bar = '0' then
            rx_buff_out <= (others => '0');
        elsif rising_edge(clk) then
            if load_en = '1' then
                rx_buff_out <= rx_buff_in;
            end if;

        end if;
    end process;
end Behavioral;

--------------------hex_digit_reg-------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

```

```vhdl
entity hex_digit_reg is
    port (
        rst_bar : in std_logic; -- asynchronous reset
        clk : in std_logic; -- system clock
        load_en1 : in std_logic; -- enable load
        load_en2 : in std_logic; -- enable load
        hex_dig_in : in std_logic_vector(3 downto 0); -- received data in
        hex_dig_out : out std_logic_vector(3 downto 0) -- received data out
    );
end hex_digit_reg;

architecture behavioral of hex_digit_reg is
begin
    reg : process (clk, rst_bar)
    begin
        if rst_bar = '0' then
            hex_dig_out <= (others => '0');
        elsif rising_edge(clk) then
            if (load_en1 = '1') and (load_en2 = '1') then
                hex_dig_out <= hex_dig_in;
            end if;
        end if;
    end process;
end behavioral;

--------------------decoder_2to4--------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity decoder_2to4 is
    port (
        b : in std_logic; -- most significant address bit
        a : in std_logic; -- least significant address bit
        Y : out std_logic_vector(3 downto 0) -- selected output asserted high
    );
end decoder_2to4;

architecture dataflow of decoder_2to4 is
begin
     Y <= "0001" when (a & b = "00") else
     "0010" when (a & b = "01") else
     "0100" when (a & b = "10") else
     "1000" when (a & b = "11") else
     (others => '0');  -- default case to handle undefined statesend dataflow;
end dataflow;

---------------------load_digit_fsm------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


```

```vhdl
entity load_digit_fsm is
    port (
        rst_bar : in std_logic; -- asynchronous system reset
        clk : in std_logic; -- system clock
        ss_bar_pe : in std_logic; -- positive edge of ss_bar detected
        ld_cmd : in std_logic; -- bit 7 is '1' for load command
        load_dig : out std_logic -- enable a hex_digit to be loaded
    );
end load_digit_fsm;

architecture moore_fsm of load_digit_fsm is
type state is (wait_for_sb_0, wait_for_sb_1, wait_ldc_1, output_state);
signal present_state, next_state : state;
begin
    -- first state : detects rst_bar
    state_reg : process (clk, rst_bar)
    begin
        if rst_bar = '0' then
            present_state <= wait_for_sb_0;
        elsif rising_edge(clk) then
            present_state <= next_state;
        end if;
    end process;

    -- process where it outputs
    outputs: process (present_state)
    begin
        case present_state is
            when output_state => load_dig <= '1';
            when others => load_dig <= '0';
        end case;
    end process;

    nxt_state: process (present_state, ss_bar_pe, ld_cmd)
    begin
        case present_state is
            when wait_for_sb_0 =>
            if ss_bar_pe = '0' then
                next_state <= wait_for_sb_1;
            else
                next_state <= wait_for_sb_0;
            end if;

            when wait_for_sb_1 =>
            if ss_bar_pe = '1' then
                next_state <= wait_ldc_1;
            else
                next_state <= wait_for_sb_1;
            end if;

            when wait_ldc_1 =>
            if ld_cmd = '1' then
                next_state <= output_state;
            else
                next_state <= wait_ldc_1;
            end if;

            when output_state =>
            next_state <= wait_for_sb_0;

            when others =>
            next_state <= wait_for_sb_0;

        end case;
    end process;

end moore_fsm;


--------------------hex_dig_mux --------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity hex_dig_mux is
    port (
        hex_dig_0 : in std_logic_vector(3 downto 0); -- mux input vectors
        hex_dig_1 : in std_logic_vector(3 downto 0); -- mux input vectors
        hex_dig_2 : in std_logic_vector(3 downto 0); -- mux input vectors
        hex_dig_3 : in std_logic_vector(3 downto 0); -- mux input vectors
        sel : in std_logic_vector(1 downto 0); -- multiplexer select
inputs
        hex_dig_out : out std_logic_vector(3 downto 0) -- multiplexer
output
    );
end hex_dig_mux;

architecture behavioral of hex_dig_mux is
begin
    with sel select
    hex_dig_out <= hex_dig_0 when "00",
    hex_dig_1 when "01",
    hex_dig_2 when "10",
    hex_dig_3 when "11",
    "----" when others;

end behavioral;

--------------------hex_seven-----------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity hex_seven is
    port(
        hex : in std_logic_vector(3 downto 0); -- hexadecimal input
        -- segs. a..g right justified
        seg_drive : out std_logic_vector(7 downto 0)
    );
end hex_seven;


```

```vhdl
architecture behavioral of hex_seven is
begin
    with hex select
    seg_drive <=
            "01111110" when "0000", -- 0
            "00110000" when "0001", -- 1
            "01101101" when "0010", -- 2
            "01111001" when "0011", -- 3
            "00110011" when "0100", -- 4
            "01011011" when "0101", -- 5
            "01011111" when "0110", -- 6
            "01110000" when "0111", -- 7
            "01111111" when "1000", -- 8
            "01111011" when "1001", -- 9
            "01110111" when "1010", -- A
            "00011111" when "1011", -- b
            "01001110" when "1100", -- C
            "00111101" when "1101", -- d
            "01001111" when "1110", -- E
            "01000111" when others; -- F
end architecture behavioral;


--------------------top level-----------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity spi_mux_digit_driver is
    port(
    rst_bar : in std_logic; -- asynchronous system reset
    clk : in std_logic; -- system clock
    mosi : in std_logic; -- master out slave in SPI serial data
    sck : in std_logic; -- SPI shift clock to slave
    ss_bar : in std_logic; -- slave select signal
    sel : in std_logic_vector(1 downto 0);
    seg_drive : out std_logic_vector(7 downto 0)
    );
end spi_mux_digit_driver;

architecture spi_mux_digit_driver of spi_mux_digit_driver is
signal sig_edge_to_shift_en : std_logic;
signal rx_8bits : std_logic_vector(7 downto 0);
signal u3_to_u4 : std_logic;
signal u4_output_8bits : std_logic_vector(7 downto 0);
signal u5_output : std_logic_vector(3 downto 0);
signal u12_output : std_logic;
signal hex_reg_0, hex_reg_1, hex_reg_2, hex_reg_3 : std_logic_vector(3
downto 0);
signal u10_output : std_logic_vector(3 downto 0);
begin
    u1 : entity edge_det port map (clk => clk, rst_bar => rst_bar, sig =>
sck, pos => '1', sig_edge => sig_edge_to_shift_en);
    u2 : entity slv_spi_rx_shifter
        port map (
        shift_en => sig_edge_to_shift_en,
        clk => clk,
        sel_bar => ss_bar,
        rst_bar => rst_bar,
        rxd => mosi,
        rx_data_out => rx_8bits
        );
    u3 : entity edge_det port map (clk => clk, rst_bar => rst_bar, sig =>
ss_bar, pos => '1', sig_edge => u3_to_u4);
    u4 : entity rx_buff_reg
        port map(
        rst_bar => rst_bar,
        clk => clk,
        load_en => u3_to_u4,
        rx_buff_in => rx_8bits,
        rx_buff_out => u4_output_8bits
        );
    u5 : entity decoder_2to4
        port map(
        b => u4_output_8bits(5),
        a => u4_output_8bits(4),
        y => u5_output
        );
    u6 : entity hex_digit_reg
        port map(
        rst_bar => rst_bar,
        clk => clk,
        load_en1 => u5_output(0),
        load_en2 => u12_output,
        hex_dig_in => u4_output_8bits(3 downto 0),
        hex_dig_out => hex_reg_0
        );

    u7 : entity hex_digit_reg
        port map(
        rst_bar => rst_bar,
        clk => clk,
        load_en1 => u5_output(1),
        load_en2 => u12_output,
        hex_dig_in => u4_output_8bits(3 downto 0),
        hex_dig_out => hex_reg_1
        );

    u8 : entity hex_digit_reg
        port map(
        rst_bar => rst_bar,
        clk => clk,
        load_en1 => u5_output(2),
        load_en2 => u12_output,
        hex_dig_in => u4_output_8bits(3 downto 0),
        hex_dig_out => hex_reg_2
        );

    u9 : entity hex_digit_reg
        port map(
        rst_bar => rst_bar,
        clk => clk,
        load_en1 => u5_output(3),
```

```vhdl
        load_en2 => u12_output,
        hex_dig_in => u4_output_8bits(3 downto 0),
        hex_dig_out => hex_reg_3
        );

    u10 : entity hex_dig_mux
        port map(
        hex_dig_0 => hex_reg_0,
        hex_dig_1 => hex_reg_1,
        hex_dig_2 => hex_reg_2,
        hex_dig_3 => hex_reg_3,
        sel => sel,
        hex_dig_out => u10_output
        );

    u11 : entity hex_seven
        port map(
        hex => u10_output,
        seg_drive => seg_drive
        );


    u12 : entity load_digit_fsm
        port map(
        rst_bar => rst_bar,
        clk => clk,
        ss_bar_pe => u3_to_u4,
        ld_cmd => u4_output_8bits(7),
        load_dig => u12_output
        );


end spi_mux_digit_driver;

```