

```

1  -----
2  --
3  -- Title       : spi_mux_digit_driver
4  -- Design      : prelab11
5  -- Author      : Dongyun Lee
6  -- Company     : Stony Brook University
7  --
8  -----
9  --
10 -- File        : \\Mac\Home\Documents\SBU 2024 Spring\ESE
11 382\Prelab11\Prelab11\prelab11\src\spi_mux_digit_driver.vhd
12 -- Generated   : Tue Apr 30 15:39:45 2024
13 -- From        : interface description file
14 -- By          : Itf2Vhdl ver. 1.22
15 -----
16 --
17 -- Description :
18 --
19 -----
20
21 -----slv_spi_rx_shifter-----
22 library ieee;
23 use ieee.std_logic_1164.all;
24 use ieee.numeric_std.all;
25
26
27 entity slv_spi_rx_shifter is
28     port(
29         rxd          : in  std_logic;           -- Data received from master
30         rst_bar      : in  std_logic;           -- Asynchronous reset
31         sel_bar       : in  std_logic;           -- Selects shifter for
32         operation
33         clk           : in  std_logic;           -- System clock
34         shift_en      : in  std_logic;           -- Enable shift
35         rx_data_out   : out std_logic_vector(7 downto 0) -- Received data
36     );
37 end entity slv_spi_rx_shifter;
38
39 architecture slv_spi_rx_shifter of slv_spi_rx_shifter is
40 begin
41     shift: process (clk, rst_bar)
42         -- variable memory : unsigned(7 downto 0);
43     begin
44         if rst_bar = '0' then
45             rx_data_out <= (others => '0');
46         elsif rising_edge(clk) then
47             if sel_bar = '0' then
48                 if (shift_en = '1') and (rxd = '1' or rxd = '0') then
49                     rx_data_out <= rx_data_out(6 downto 0) & rxd;
50                 end if;
51             end if;
52         end if;
53     end process;
54 end architecture slv_spi_rx_shifter;

```

```

52
53         end if;
54
55     end process;
56 end slv_spi_rx_shifter;
57
58
59 -----edge_det-----
60 library ieee;
61 use ieee.std_logic_1164.all;
62 use ieee.numeric_std.all;
63
64
65 entity edge_det is
66     port(
67         rst_bar    : in  std_logic; -- Asynchronous system reset
68         clk        : in  std_logic; -- System clock
69         sig        : in  std_logic; -- Input signal
70         pos        : in  std_logic; -- '1' for positive edge, '0' for
        negative
71         sig_edge   : out std_logic  -- High for one system clk after edge
        detected
72     );
73 end entity edge_det;
74
75
76 architecture moore_fsm of edge_det is
77     type state is (state_a, state_b, state_c);
78     signal present_state, next_state : state;
79 begin
80     -- first state : detects rst_bar
81     state_reg: process (clk, rst_bar)
82     begin
83         if rst_bar = '0' then
84             present_state <= state_a;
85         elsif rising_edge(clk) then
86             present_state <= next_state;
87         end if;
88     end process;
89
90     -- process where it outputs
91     outputs: process (present_state)
92     begin
93         case present_state is
94             when state_c => sig_edge <= '1';
95             when others => sig_edge <= '0';
96         end case;
97     end process;
98
99     nxt_state: process (present_state, sig)
100     begin
101         case present_state is
102             when state_a =>
103                 if (pos = '1' and sig = '0') or (pos = '0' and sig = '1') then
104                     next_state <= state_b;
105                 else
106                     next_state <= state_a;

```

```

107         end if;
108
109         when state_b =>
110             if (pos = '1' and sig = '1') or (pos = '0' and sig = '0') then
111                 next_state <= state_c;
112             else
113                 next_state <= state_b;
114             end if;
115
116         when others =>
117             if (pos = '1' and sig = '0') or (pos = '0' and sig = '1') then
118                 next_state <= state_b;
119             else
120                 next_state <= state_a;
121             end if;
122         end case;
123     end process;
124
125 end moore_fsm;
126
127
128 -----rx_buff_reg-----
129 library ieee;
130 use ieee.std_logic_1164.all;
131 use ieee.numeric_std.all;
132
133 entity rx_buff_reg is
134     port (
135         rst_bar : in std_logic; -- asynchronous reset
136         clk : in std_logic; -- system clock
137         load_en : in std_logic; -- enable shift
138         rx_buff_in : in std_logic_vector(7 downto 0); -- received data in
139         rx_buff_out : out std_logic_vector(7 downto 0) -- received data
140     );
141 end rx_buff_reg;
142
143 architecture Behavioral of rx_buff_reg is
144     begin
145         double_buffer : process (clk, rst_bar)
146             begin
147                 if rst_bar = '0' then
148                     rx_buff_out <= (others => '0');
149                 elsif rising_edge(clk) then
150                     if load_en = '1' then
151                         rx_buff_out <= rx_buff_in;
152                     end if;
153                 end if;
154             end process;
155         end Behavioral;
156
157
158 -----hex_digit_reg-----
159 library ieee;
160 use ieee.std_logic_1164.all;
161 use ieee.numeric_std.all;
162

```

```

163 entity hex_digit_reg is
164     port (
165         rst_bar : in std_logic; -- asynchronous reset
166         clk : in std_logic; -- system clock
167         load_en1 : in std_logic; -- enable load
168         load_en2 : in std_logic; -- enable load
169         hex_dig_in : in std_logic_vector(3 downto 0); -- received data in
170         hex_dig_out : out std_logic_vector(3 downto 0) -- received data
171     out
172 );
173 end hex_digit_reg;
174 architecture behavioral of hex_digit_reg is
175 begin
176     reg : process (clk, rst_bar)
177     begin
178         if rst_bar = '0' then
179             hex_dig_out <= (others => '0');
180         elsif rising_edge(clk) then
181             if (load_en1 = '1') and (load_en2 = '1') then
182                 hex_dig_out <= hex_dig_in;
183             end if;
184         end if;
185     end process;
186 end behavioral;
187
188 -----decoder_2to4-----
189
190 library ieee;
191 use ieee.std_logic_1164.all;
192 use ieee.numeric_std.all;
193
194 entity decoder_2to4 is
195     port (
196         b : in std_logic; -- most significant address bit
197         a : in std_logic; -- least significant address bit
198         Y : out std_logic_vector(3 downto 0) -- selected output asserted
199     high
200 );
201 end decoder_2to4;
202
203 architecture dataflow of decoder_2to4 is
204 begin
205     Y <= "0001" when (a & b = "00") else
206         "0010" when (a & b = "01") else
207         "0100" when (a & b = "10") else
208         "1000" when (a & b = "11") else
209         (others => '0'); -- default case to handle undefined statesend
210 dataflow;
211 end dataflow;
212
213 -----load_digit_fsm-----
214 library ieee;
215 use ieee.std_logic_1164.all;
216 use ieee.numeric_std.all;

```

```

217 entity load_digit_fsm is
218     port (
219         rst_bar : in std_logic; -- asynchronous system reset
220         clk : in std_logic; -- system clock
221         ss_bar_pe : in std_logic; -- positive edge of ss_bar detected
222         ld_cmd : in std_logic; -- bit 7 is '1' for load command
223         load_dig : out std_logic -- enable a hex_digit to be loaded
224     );
225 end load_digit_fsm;
226
227 architecture moore_fsm of load_digit_fsm is
228     type state is (wait_for_sb_0, wait_for_sb_1, wait_ldc_1, output_state);
229     signal present_state, next_state : state;
230 begin
231     -- first state : detects rst_bar
232     state_reg : process (clk, rst_bar)
233     begin
234         if rst_bar = '0' then
235             present_state <= wait_for_sb_0;
236         elsif rising_edge(clk) then
237             present_state <= next_state;
238         end if;
239     end process;
240
241     -- process where it outputs
242     outputs: process (present_state)
243     begin
244         case present_state is
245             when output_state => load_dig <= '1';
246             when others => load_dig <= '0';
247         end case;
248     end process;
249
250     nxt_state: process (present_state, ss_bar_pe, ld_cmd)
251     begin
252         case present_state is
253             when wait_for_sb_0 =>
254                 if ss_bar_pe = '0' then
255                     next_state <= wait_for_sb_1;
256                 else
257                     next_state <= wait_for_sb_0;
258                 end if;
259
260             when wait_for_sb_1 =>
261                 if ss_bar_pe = '1' then
262                     next_state <= wait_ldc_1;
263                 else
264                     next_state <= wait_for_sb_1;
265                 end if;
266
267             when wait_ldc_1 =>
268                 if ld_cmd = '1' then
269                     next_state <= output_state;
270                 else
271                     next_state <= wait_ldc_1;
272                 end if;
273

```

```

274         when output_state =>
275             next_state <= wait_for_sb_0;
276
277         when others =>
278             next_state <= wait_for_sb_0;
279
280     end case;
281 end process;
282
283 end moore_fsm;
284
285 -----hex_dig_mux -----
286
287 library ieee;
288 use ieee.std_logic_1164.all;
289 use ieee.numeric_std.all;
290
291 entity hex_dig_mux is
292     port (
293         hex_dig_0 : in std_logic_vector(3 downto 0); -- mux input vectors
294         hex_dig_1 : in std_logic_vector(3 downto 0); -- mux input vectors
295         hex_dig_2 : in std_logic_vector(3 downto 0); -- mux input vectors
296         hex_dig_3 : in std_logic_vector(3 downto 0); -- mux input vectors
297         sel : in std_logic_vector(1 downto 0); -- multiplexer select
298
299         inputs
300         hex_dig_out : out std_logic_vector(3 downto 0) -- multiplexer
301         output
302     );
303 end hex_dig_mux;
304
305 architecture behavioral of hex_dig_mux is
306 begin
307     with sel select
308         hex_dig_out <= hex_dig_0 when "00",
309         hex_dig_1 when "01",
310         hex_dig_2 when "10",
311         hex_dig_3 when "11",
312         "----" when others;
313
314 end behavioral;
315
316 -----hex_seven-----
317
318 library ieee;
319 use ieee.std_logic_1164.all;
320 use ieee.numeric_std.all;
321
322 entity hex_seven is
323     port(
324         hex : in std_logic_vector(3 downto 0); -- hexadecimal input
325         -- segs. a..g right justified
326         seg_drive : out std_logic_vector(7 downto 0)
327     );
328 end hex_seven;

```

```

329 architecture behavioral of hex_seven is
330 begin
331     with hex select
332     seg_drive <=
333         "01111110" when "0000", -- 0
334         "00110000" when "0001", -- 1
335         "01101101" when "0010", -- 2
336         "01111001" when "0011", -- 3
337         "00110011" when "0100", -- 4
338         "01011011" when "0101", -- 5
339         "01011111" when "0110", -- 6
340         "01110000" when "0111", -- 7
341         "01111111" when "1000", -- 8
342         "01111011" when "1001", -- 9
343         "01110111" when "1010", -- A
344         "00011111" when "1011", -- b
345         "01001110" when "1100", -- C
346         "00111101" when "1101", -- d
347         "01001111" when "1110", -- E
348         "01000111" when others; -- F
349 end architecture behavioral;
350
351
352 -----top level-----
353 library ieee;
354 use ieee.std_logic_1164.all;
355 use ieee.numeric_std.all;
356 use work.all;
357
358 entity spi_mux_digit_driver is
359 port(
360     rst_bar : in std_logic; -- asynchronous system reset
361     clk : in std_logic; -- system clock
362     mosi : in std_logic; -- master out slave in SPI serial data
363     sck : in std_logic; -- SPI shift clock to slave
364     ss_bar : in std_logic; -- slave select signal
365     sel : in std_logic_vector(1 downto 0);
366     seg_drive : out std_logic_vector(7 downto 0)
367 );
368 end spi_mux_digit_driver;
369
370 architecture spi_mux_digit_driver of spi_mux_digit_driver is
371 signal sig_edge_to_shift_en : std_logic;
372 signal rx_8bits : std_logic_vector(7 downto 0);
373 signal u3_to_u4 : std_logic;
374 signal u4_output_8bits : std_logic_vector(7 downto 0);
375 signal u5_output : std_logic_vector(3 downto 0);
376 signal u12_output : std_logic;
377 signal hex_reg_0, hex_reg_1, hex_reg_2, hex_reg_3 : std_logic_vector(3
downto 0);
378 signal u10_output : std_logic_vector(3 downto 0);
379 begin
380     u1 : entity edge_det port map (clk => clk, rst_bar => rst_bar, sig =>
sck, pos => '1', sig_edge => sig_edge_to_shift_en);
381     u2 : entity slv_spi_rx_shifter
382         port map (
383         shift_en => sig_edge_to_shift_en,

```

```

384         clk => clk,
385         sel_bar => ss_bar,
386         rst_bar => rst_bar,
387         rxd => mosi,
388         rx_data_out => rx_8bits
389     );
390     u3 : entity edge_det port map (clk => clk, rst_bar => rst_bar, sig =>
ss_bar, pos => '1', sig_edge => u3_to_u4);
391     u4 : entity rx_buff_reg
392         port map(
393             rst_bar => rst_bar,
394             clk => clk,
395             load_en => u3_to_u4,
396             rx_buff_in => rx_8bits,
397             rx_buff_out => u4_output_8bits
398         );
399     u5 : entity decoder_2to4
400         port map(
401             b => u4_output_8bits(5),
402             a => u4_output_8bits(4),
403             y => u5_output
404         );
405     u6 : entity hex_digit_reg
406         port map(
407             rst_bar => rst_bar,
408             clk => clk,
409             load_en1 => u5_output(0),
410             load_en2 => u12_output,
411             hex_dig_in => u4_output_8bits(3 downto 0),
412             hex_dig_out => hex_reg_0
413         );
414
415     u7 : entity hex_digit_reg
416         port map(
417             rst_bar => rst_bar,
418             clk => clk,
419             load_en1 => u5_output(1),
420             load_en2 => u12_output,
421             hex_dig_in => u4_output_8bits(3 downto 0),
422             hex_dig_out => hex_reg_1
423         );
424
425     u8 : entity hex_digit_reg
426         port map(
427             rst_bar => rst_bar,
428             clk => clk,
429             load_en1 => u5_output(2),
430             load_en2 => u12_output,
431             hex_dig_in => u4_output_8bits(3 downto 0),
432             hex_dig_out => hex_reg_2
433         );
434
435     u9 : entity hex_digit_reg
436         port map(
437             rst_bar => rst_bar,
438             clk => clk,
439             load_en1 => u5_output(3),

```



```
440         load_en2 => u12_output,
441         hex_dig_in => u4_output_8bits(3 downto 0),
442         hex_dig_out => hex_reg_3
443     );
444
445     u10 : entity hex_dig_mux
446     port map(
447         hex_dig_0 => hex_reg_0,
448         hex_dig_1 => hex_reg_1,
449         hex_dig_2 => hex_reg_2,
450         hex_dig_3 => hex_reg_3,
451         sel => sel,
452         hex_dig_out => u10_output
453     );
454
455     u11 : entity hex_seven
456     port map(
457         hex => u10_output,
458         seg_drive => seg_drive
459     );
460
461
462     u12 : entity load_digit_fsm
463     port map(
464         rst_bar => rst_bar,
465         clk => clk,
466         ss_bar_pe => u3_to_u4,
467         ld_cmd => u4_output_8bits(7),
468         load_dig => u12_output
469     );
470
471
472 end spi_mux_digit_driver;
473
```