Lab 4 Report

ECE 124
Group 9 Section 201

Chengfeng Deng
Tasviq Hossain

VHDL Design for LogicalStep_Lab4_top

```
1  -- Chengfeng Deng, Tasviq Hossain
2  -- Group 9 Section 201
3
4  LIBRARY ieee;
5  USE ieee.std_logic_1164.ALL;
6  USE ieee.numeric_std.ALL;
7
8  ENTITY LogicalStep_Lab4_top IS
9     PORT
10    (
11    Clk          : in  std_logic;
12    pb_n         : in  std_logic_vector(3 downto 0);
13    sw           : in  std_logic_vector(7 downto 0);
14    leds         : out std_logic_vector(7 downto 0);
15
16 ------------------------------------------------------------------
17    xreg, yreg   : out std_logic_vector(3 downto 0);-- (for SIMULATION only)
18    xPOS, yPOS   : out std_logic_vector(3 downto 0);-- (for SIMULATION only)
19 ------------------------------------------------------------------
20    seg7_data    : out std_logic_vector(6 downto 0); -- 7-bit outputs to a 7-segment display (for LogicalStep only)
21    seg7_char1   : out std_logic;                    -- seg7 digit1 selector (for LogicalStep only)
22    seg7_char2   : out std_logic                     -- seg7 digit2 selector (for LogicalStep only)
23
24    );
25 END LogicalStep_Lab4_top;
26
27 ARCHITECTURE Circuit OF LogicalStep_Lab4_top IS
28
29 -------------------------------------------------------------------------------
30 -- Provided Project Components Used for sevensegment output and clk sources        --
31 ------------------------------------------------------------------ ------------
32 COMPONENT Clock_Source  port (SIM_FLAG: in boolean;clk_input: in std_logic;clock_out: out std_logic);
33 END COMPONENT;
34
35 component SevenSegment
36   port
37    (
38       hex      :  in  std_logic_vector(3 downto 0);    -- The 4 bit data to be displayed
39       sevenseg :  out std_logic_vector(6 downto 0)     -- 7-bit outputs to a 7-segment
40    );
41 end component SevenSegment;
42
43 component segment7_mux
44   port
45    (
46       clk        : in  std_logic := '0';
47       DIN2     : in  std_logic_vector(6 downto 0);
48       DIN1     : in  std_logic_vector(6 downto 0);
49       DOUT       : out std_logic_vector(6 downto 0);
```

Cont.

```
50          DIG2        : out std_logic;
51          DIG1        : out std_logic
52      );
53  end component segment7_mux;
54
55
56  ----------------------------------------------------------------
57  --              Add  Other Components here                    --
58  ----------------------------------------------------------------
59
60  component Bidir_shift_reg is port
61      (
62              CLK             : in std_logic := '0';
63              RESET           : in std_logic := '0';
64              CLK_EN          : in std_logic := '0';
65              LEFT0_RIGHT1    : in std_logic := '0';
66              REG_BITS        : out std_logic_vector(3 downto 0)
67      );
68  end component;
69
70  component U_D_Bin_Counter4bit is port
71      (
72              CLK             : in std_logic := '0';
73              RESET           : in std_logic := '0';
74              CLK_EN          : in std_logic := '0';
75              UP1_DOWN0       : in std_logic := '0';
76              COUNTER_BITS    : out std_logic_vector(3 downto 0)
77      );
78  end component;
79
80  component Compx4 is port
81      (
82              A               : in std_logic_vector(3 downto 0);
83              B               : in std_logic_vector(3 downto 0);
84              AGTB, AEQB, ALTB : out std_logic
85      );
86  end component;
87
88  component Extender is port
89      (
90          CLK         : in std_logic;
91          Reset       : in std_logic;
92          Extender_en : in std_logic;
93          Extender    : in std_logic;
94          ext_pos     : in std_logic_vector(3 downto 0);
95          clk_en      : out std_logic;
96          left0_right1: out std_logic;
97          extender_out: out std_logic;
98          grappler_en : out std_logic
```

Cont.

```
 99      );
100    END component;
101
102    component Grappler is port
103        (
104            CLK         : in std_logic;
105            Reset       : in std_logic;
106            grappler    : in std_logic;
107            grappler_en: in std_logic;
108            grappler_on: out std_logic
109        );
110    END component;
111
112    component Inverter is port
113        (
114           pb_n3        : in std_logic;
115           pb_n2        : in std_logic;
116           pb_n1        : in std_logic;
117           pb_n0        : in std_logic;
118           AH_pb_n3     : out std_logic;
119           AH_pb_n2     : out std_logic;
120           AH_pb_n1     : out std_logic;
121           AH_pb_n0     : out std_logic
122        );
123    end component;
124
125    component Target_Register is port
126        (
127           CLK             : in std_logic;
128           Reset           : in std_logic;
129           Capture         : in std_logic;
130           Target_Value    : in std_logic_vector  (3 downto 0);
131           Reg_Value       : out std_logic_vector (3 downto 0)
132        );
133    END component;
134
135    component XY_Motion is port
136        (
137           CLK              : in std_logic;
138           reset            : in std_logic;
139           X_GT             : in std_logic;
140           X_EQ             : in std_logic;
141           X_LT             : in std_logic;
142           motion           : in std_logic;
143           Y_GT             : in std_logic;
144           Y_EQ             : in std_logic;
145           Y_LT             : in std_logic;
146           extender_out     : in std_logic;
147           clk1_en          : out std_logic;
```

Cont.

```
148         up1_down0_1     : out std_logic;
149         clk2_en         : out std_logic;
150         up1_down0_2     : out std_logic;
151         error           : out std_logic;
152         Capture_XY      : out std_logic;
153         extender_en     : out std_logic
154     );
155 end component;
156
157
158 ------------------------------------------------------------------
159 --                provided signals
160 ------------------------------------------------------------------
161 signal clk_in, clock : std_logic;                                --
162 ------------------------------------------------------------------
163 constant SIM_FLAG : boolean := TRUE; -- set to FALSE when compiling for FPGA download to LogicalStep board
164 ------------------------------------------------------------------
165 ------------------------------------------------------------------
166 --                Declared Signals                              --
167 ------------------------------------------------------------------
168
169 -- signals for inverter
170 signal reset            : std_logic;
171 signal motion           : std_logic;
172 signal extender_in      : std_logic;
173 signal grappler_in      : std_logic;
174
175 -- signals for XY Motion
176 signal clk_enX     : std_logic;
177 signal clk_enY     : std_logic;
178 signal up_downX    : std_logic;
179 signal up_downY    : std_logic;
180 signal error       : std_logic;
181 signal Capture_XY    : std_logic;
182 signal extender_en   : std_logic;
183
184 -- signals for extender
185 signal clk_en_ext       : std_logic;
186 signal left_right       : std_logic;
187 signal extender_out_IN  : std_logic;
188 signal grappler_en      : std_logic;
189
190 -- signals for grappler
191 signal grappler_on   : std_logic;
192
193 -- signals for X COUNTER
194 signal X_pos    : std_logic_vector(3 downto 0);
195
196 -- signals for Y COUNTER
```

Cont.

```
197   signal Y_pos    : std_logic_vector(3 downto 0);
198
199   -- signals for X Register
200   signal X_reg    : std_logic_vector(3 downto 0);
201
202   -- signals for Y Register
203   signal Y_reg    : std_logic_vector(3 downto 0);
204
205   -- signals for extender reg4
206   signal ext_pos     : std_logic_vector(3 downto 0);
207
208   -- signals for X Compx4
209   signal X_GT        : std_logic;
210   signal X_EQ        : std_logic;
211   signal X_LT        : std_logic;
212
213   -- signals for Y Compx4
214   signal Y_GT        : std_logic;
215   signal Y_EQ        : std_logic;
216   signal Y_LT        : std_logic;
217
218   -- signal for X7Segdec
219   signal X_7seg      : std_logic_vector(6 downto 0);
220
221   -- signal for Y7Segdec
222   signal Y_7seg      : std_logic_vector(6 downto 0);
223
224   -- no need for extra clearification as all signals
225   -- are defined in top entity part
226
227   -
228   ------------------------------------------------------------------
229   --                Declared signal ends                          --
230   ------------------------------------------------------------------
231   -
232   BEGIN -- here the circuit begins
233   clk_in <= clk;
234
235   ------------------------------------------------------------------
236   --           Clock_Selector Declared                            --
237   ------------------------------------------------------------------
238   Clock_Selector: Clock_source port map(SIM_FLAG, clk_in, clock); --
239   ------------------------------------------------------------------
240   --                 Instances Begin                              --
241   ------------------------------------------------------------------
242   -- Instance for the Inverter
243   inst0: Inverter port map(
244          pb_n(3),pb_n(2),pb_n(1),pb_n(0),
245          reset,motion,extender_in,grappler_in
```

Cont.

```
246         );
247 
248     -- Instance for the XY Motion Component
249     inst1: XY_Motion port map(
250         clock,
251         reset,
252         X_GT,
253         X_EQ,
254         X_LT,
255         motion,
256         Y_GT,
257         Y_EQ,
258         Y_LT,
259         extender_out_IN,
260         clk_enX,
261         up_downX,
262         clk_enY,
263         up_downY,
264         error,
265         Capture_XY,
266         extender_en
267         );
268 
269         leds(0) <= error;
270 
271     -- Instance for the extender component
272     inst2: Extender port map(
273         clock,
274         reset,
275         extender_en,
276         extender_in,
277         ext_pos,
278         clk_en_ext,
279         left_right,
280         extender_out_IN,
281         grappler_en
282         );
283     -- Instance for the grappler component
284     inst3: Grappler port map(
285         clock,
286         reset,
287         grappler_in,
288         grappler_en,
289         grappler_on
290         );
291         leds(1) <= grappler_on;
```

Cont.

```
292     -- binary counter for x position
293     inst4: U_D_Bin_Counter4bit port map(
294             clock,
295             reset,
296             clk_enX,
297             up_downX,
298             X_pos-- (3 downto 0);
299             );
300     
301             Xpos(3 downto 0) <= X_pos;-- (3 downto 0);
302     
303     -- Register for X position
304     inst5: Target_Register port map(
305             clock,
306             reset,
307             Capture_XY,
308             sw(7 downto 4),
309             X_reg-- (3 downto 0)
310             );
311     
312             Xreg(3 downto 0) <= X_reg;--(3 downto 0);
313     
314     -- binary counter for y position
315     inst6: U_D_Bin_Counter4bit port map(
316             clock,
317             reset,
318             clk_enY,
319             up_downY,
320             Y_pos-- (3 downto 0);
321             );
322     
323             Ypos(3 downto 0) <= Y_pos;-- (3 downto 0);
324     
325     -- Register for Y position
326     inst7: Target_Register port map(
327             clock,
328             reset,
329             Capture_XY,
330             sw(3 downto 0),
331             Y_reg-- (3 downto 0)
332             );
333     
334             Yreg(3 downto 0) <= Y_reg;--(3 downto 0);
335     -- Reg4 for extender
336     inst8: Bidir_shift_reg port map(
337             clock,
338             reset,
339             clk_en_ext,
340             left_right,
```

Cont.

```
341         ext_pos
342         );
343 
344         leds(5 downto 2) <= ext_pos;
345 
346 -- Compx4 for X position
347 inst9: Compx4 port map(
348         X_pos,
349         X_reg,
350         X_GT,
351         X_EQ,
352         X_LT
353         );
354 
355 -- Compx4 for y position
356 inst10: Compx4 port map(
357         Y_pos,
358         Y_reg,
359         Y_GT,
360         Y_EQ,
361         Y_LT
362         );
363 
364 -- Hex to 7seg for X
365 inst11: SevenSegment port map(
366         X_pos,
367         X_7seg
368         );
369 
370 -- Hex to 7seg for Y
371 inst12: SevenSegment port map(
372         Y_pos,
373         Y_7seg
374         );
375 
376 inst13: segment7_mux port map(
377         clk_in,
378         X_7seg,
379         Y_7seg,
380         seg7_data (6 downto 0),
381         seg7_char1,
382         seg7_char2
383         );
384 
385 
386 END Circuit;
```

VHDL Design for State Machine (Moore): Extender

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

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

ENTITY Extender is port
   (
       CLK         : in std_logic;
       Reset       : in std_logic;
       Extender_en : in std_logic;
       Extender    : in std_logic;
       ext_pos     : in std_logic_vector(3 downto 0);
       clk_en      : out std_logic;    -- enables the usage of extender when is 1 and disables it when is 0
       left0_right1: out std_logic;    -- when 0 shift left, when 1 shift right
       extender_out: out std_logic;    -- identify the out of extender
       grappler_en : out std_logic
   );
END ENTITY;

ARCHITECTURE E_Logic of Extender is

 TYPE STATE_NAMES IS (S0, extender_pressed_S0, extending_state, extended_state,extender_pressed_extended, retracting_state);
 SIGNAL current_state, next_state   :  STATE_NAMES;


--             here the circuit begins             --
BEGIN

REGISTER_SECTION : process(CLK, Reset, next_state)

   BEGIN

       if (Reset = '1') then
          current_state <= S0;
       elsif(rising_edge(CLK)) then
          current_state <= next_state;
       end if;

   END process;
```

Cont.

```
42  TRANSITION_SECTION : process(Extender_en, Extender, ext_pos, current_state)
43      BEGIN
44      -- seperate cases by extender_en value
45              CASE current_state is
46
47              -- In S0 state the extender should be fully retracted
48              -- If the extender input is 1, the state go to extender_pressed
49              -- otherwise, the extender stays at fully retracted(S0) state
50
51                 when S0 =>
52                     if (Extender = '1' AND Extender_en = '1') then
53
54                       next_state <= extender_pressed_S0;
55
56                     else
57
58                       next_state <= S0;
59
60                     end if;
61
62              -- In extender_pressed state we mainly detect whether the extender button has been pressed & released
63              -- If the button is released then we take ext_pos into account
64              -- if ext_pos is not 1111, then the ext pos needs to be increased, otherwise it needs to be decreased
65                 when extender_pressed_S0 =>
66                     if (Extender = '1') then
67
68                       next_state <= extender_pressed_S0;
69
70                     else
71
72                       next_state <= extending_state;
73
74                     end if;
75
```

Cont.

```
76          -- in extending state the extender is still extending
77          -- referring to the ext_pos, when it is not 1111, the extender is still extending
78          -- if ext_pos = 1111 then it is fully extended, next_state goes to extended state
79
80             when extending_state =>
81                 if ( NOT(ext_pos = "1111")) then
82
83                    next_state <= extending_state;
84
85                 else
86
87                    next_state <= extended_state;
88
89                 end if;
90
91          -- in extended state the extended has fully extended
92          -- we only take the Extender input into consideration, when releasing, the extended would be retracting
93          -- otherwise, the extender stays at this state forever
94
95             when extended_state =>
96                 if (Extender = '1') then
97
98                   next_state <= extender_pressed_extended;
99
100                else
101
102                  next_state <= extended_state;
103
104               end if;
105
106         -- in this state we pressed the button when the extender is done extending
107         -- if we hold the button we stay in this state
108         -- otherwise we go to retraction state
109            when extender_pressed_extended =>
110                if( Extender = '1') then
111
112                  next_state <= extender_pressed_extended;
113
114               else
115
116                  next_state <= retracting_state;
117
118               end if;
119
```

Cont.

```
120             -- in retracting state the extender is retracting
121             -- retracting would pursue until the FULL RETRACTION, that is, ext_pos = 0000
122             -- when ext_pos = 0000, state gets redirected to S0
123             -- otherwise, the state stays at retracting state
124
125                when retracting_state =>
126                    if (ext_pos = "0000") then
127
128                      next_state <= S0;
129
130                    else
131
132                      next_state <= retracting_state;
133
134                    end if;
135             end case;
136    END process;
137
138
139 -- Moore Machine, the output only depends on its current state --
140 DECODER_SECTION : process(current_state)
141    BEGIN
142       CASE current_state is
143
144          -- In S0 state, it is not moving and thus clk_en is 0
145          -- not shifting
146          -- grappler not allowed to move
147          when S0 =>
148            CLK_en       <= '0';
149            left0_right1 <= '0';
150            grappler_en  <= '0';
151            extender_out <= '0';
152
153         -- transition state
154         -- extender not moving, grappler not able to move
155          when extender_pressed_S0 =>
156            CLK_en       <= '0';
157            left0_right1 <= '0';
158            grappler_en  <= '0';
159            extender_out <= '0';
160
161          -- In extending state, it is moving and clk_en is 1
162          -- shifting right
163          -- grappler not allowed to move
164          when extending_state =>
165            CLK_en       <= '1';
166            left0_right1 <= '1';
167            grappler_en  <= '0';
```

Cont.

```
168             extender_out <= '1';
169
170           -- not moving, clk_en is 0
171           -- grappler is allowed to move
172           when extended_state =>
173             CLK_en       <= '0';
174             left0_right1 <= '0';
175             grappler_en  <= '1';
176             extender_out <= '1';
177
178           -- Transition state
179           -- extender not moving, grappler able to move
180           when extender_pressed_extended =>
181             CLK_en       <= '0';
182             left0_right1 <= '0';
183             grappler_en  <= '1';
184             extender_out <= '1';
185
186           -- moving, clk_en is 1
187           -- shifting to the left
188           -- grappler is not allowed to move
189           when retracting_state =>
190             CLK_en       <= '1';
191             left0_right1 <= '0';
192             grappler_en  <= '0';
193             extender_out <= '1';
194
195       END CASE;
196     END process;
197   END E_Logic;
```

VHDL Design for State Machine (Moore): Grappler

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

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

ENTITY Grappler is port
    (
        CLK         : in std_logic;
        Reset       : in std_logic;
        grappler    : in std_logic;    -- Grappler Toggle
        grappler_en: in std_logic;    -- Coming from the extender, acting as a condition for toggling grappler
        grappler_on: out std_logic
    );

 END ENTITY;

ARCHITECTURE G_Logic of Grappler is

  TYPE STATE_NAMES is (S0, grappler_pressed_S0, open_state, grappler_pressed_open, closed_state);

  SIGNAL current_state, next_state   :  STATE_NAMES;

--       HERE THE CIRCUIT BEGINS       --
BEGIN


REGISTER_SECTION: process (CLK, Reset, next_state)

    BEGIN

         if (Reset = '1') then
            current_state <= S0;
         elsif(rising_edge(CLK)) then
            current_state <= next_state;
         end if;

    END PROCESS;

TRANSITION_SECTION: process (grappler, grappler_en, current_state)
```

Cont.

```
42      BEGIN
43          CASE current_state is
44
45          -- the grappler is in its INITIAL STATE, which is OBVIOUSLY going to be closed!
46          -- grappler_en = 1 AND fallingedge(grappler) sends it into open state (allows it to open
47          -- and input says open)
48          -- otherwise, the grappler stays at S0, which is initial state
49
50              WHEN S0 =>
51                  if (grappler_en = '1' AND grappler = '1') then
52                     next_state <= grappler_pressed_S0;
53                  else
54                     next_state <= S0;
55                  end if;
56
57          -- this state is used to detect the press of button during S0 state,
58          -- if the button is being hold, stay, if not, go to open_state.
59
60              WHEN grappler_pressed_S0 =>
61                  if (grappler = '1') then
62                    next_state <= grappler_pressed_S0;
63
64                  else
65                    next_state <= open_state;
66
67                  end if;
68
69          -- the grappler is now in its OPEN STATE, which is again, OBVIOUSLY going to be opened!
70          -- in this case, the grappler is only changing when grappler_en = 1 AND grappler = 1
71          -- otherwise, the grappler stays open
72
73             WHEN open_state =>
74                  if (grappler_en = '1' AND grappler = '1') then
75                     next_state <= grappler_pressed_open;
76                  else
77                     next_state <= open_state;
78                  end if;
79
```

Cont.

```
80          -- this state is used to detect the press of button when the grappler is open
81          -- if the button is being hold, stay, if not, go to closed_state
82
83             WHEN grappler_pressed_open =>
84                  if ( grappler = '1') then
85                    next_state <= grappler_pressed_open;
86
87                  else
88                    next_state <= closed_state;
89
90                  end if;
91
92           -- just a rediretion to S0 because no state machine diagram is being shown
93           -- (when there are only two states)
94             WHEN closed_state =>
95                  next_state <= S0;
96
97
98          END CASE;
99
100     END PROCESS;
101
```

Cont.

```
102   -- Moore Machine, the output only depends on its current state --
103   DECODER_SECTION: process(current_state)
104   
105      BEGIN
106           -- when it is in its open state, the output is ZERO
107           -- when it is in its close state and S0, the output is ONE
108           CASE current_state is
109                WHEN S0 =>
110   
111                     grappler_on <= '1';
112   
113                WHEN grappler_pressed_S0 =>
114   
115                     grappler_on <= '1';
116   
117                WHEN open_state =>
118   
119                     grappler_on <= '0';
120   
121                WHEN grappler_pressed_open =>
122   
123                     grappler_on <= '0';
124   
125                WHEN closed_state =>
126   
127                     grappler_on <= '1';
128   
129           END CASE;
130   
131      END PROCESS;
132   END G_Logic;
```

VHDL Design for State Machine (Mealy): XY Motion

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity XY_Motion is port
    (
        CLK             : in std_logic;
        reset           : in std_logic;
        X_GT            : in std_logic;
        X_EQ            : in std_logic;
        X_LT            : in std_logic;
        motion          : in std_logic;
        Y_GT            : in std_logic;
        Y_EQ            : in std_logic;
        Y_LT            : in std_logic;
        extender_out    : in std_logic;
        clk1_en         : out std_logic;
        up1_down0_1     : out std_logic;
        clk2_en         : out std_logic;
        up1_down0_2     : out std_logic;
        error           : out std_logic;
        Capture_XY      : out std_logic;
        extender_en     : out std_logic
    );
end XY_Motion;

--          HERE THE LOGIC BEGINS      --
Architecture XY_Logic of XY_Motion is


-- four stages--
  TYPE STATE_NAMES IS (S0, motion_pressed, moving, system_error);
  SIGNAL current_state, next_state  :  STATE_NAMES;
  SIGNAL motion_done, motion_not_done, s_error: std_logic;

BEGIN
  motion_done <= Y_EQ AND X_EQ;
  motion_not_done <= not(Y_EQ AND X_EQ);
  s_error <= motion and extender_out;

Register_Section: PROCESS (CLK, reset, next_state)  -- this process synchronizes the activity to a clock
   BEGIN
      IF (reset = '1') THEN
         current_state <= S0;
      ELSIF(rising_edge(CLK)) THEN
```

Cont.

```
50          current_state <= next_state;
51      END IF;
52   END PROCESS;
53
54 TRANSITION_SECTION: Process (motion, motion_done, motion_not_done, extender_out)
55   BEGIN
56        CASE current_state IS
57
58         -- S0 is the state where the x and y coordinate is not changing
59         -- in this case, we only detect if the motion button is being pressed
60         -- if motion is one, we go to motion pressed state, otherwise we no change
61
62            when S0 =>
63
64                if ( motion = '1') then
65                  next_state <= motion_pressed;
66
67                else
68                  next_state <= S0;
69
70                end if;
71
72         -- Motion_pressed state is used to detect whether go to s_error or moving state
73         -- if extender out the s_error state occurs and everything's to be determined there
74         -- if not we check for the release of motion, when motion = 0 we go moving
75            when motion_pressed =>
76
77                if (motion = '1') then
78                  next_state <= motion_pressed;
79
80                elsif (s_error = '1') then
81                  next_state <= system_error;
82
83                elsif (motion = '0') then
84                  next_state <= moving;
85
86                end if;
87
88         -- moving state is when the RAC is moving, in this circumstances the extender would not be on
89         -- we only care about whether the xy value matches the targeted xy
90         -- if matches, go to S0 and system stop. If not, stay in this state
91            when moving =>
92
93                if (motion_not_done = '1') then
94                  next_state <= moving;
95
96                elsif( motion_done = '1' ) then
97                  next_state <= S0;
98
```

Cont.

```
 99                  end if;
100
101          -- system_error state is being used prevent xy from being changing
102          -- to exit this state the extender_out must equal to zero, and it would be redirected back to S0
103          -- if it is not zero we stay in this state no matter what
104             when system_error =>
105
106                 if ( extender_out = '1') then
107                   next_state <= system_error;
108
109                 elsif ( extender_out = '0') then
110                   next_state <= S0;
111
112                 end if;
113         END CASE;
114     END PROCESS;
115
116
117  -- It is a mealy machine
118 DECODER_SECTION: Process (current_state, X_GT, X_EQ, X_LT, Y_GT, Y_EQ, Y_LT)
119     BEGIN
120         CASE current_state is
121
122         -- In S0 state nothing would change at all, the machine is at rest
123             when S0 =>
124                 clk1_en     <= '0';
125                 up1_down0_1 <= '0';
126                 clk2_en     <= '0';
127                 up1_down0_2 <= '0';
128                 error       <= '0';
129                 Capture_XY  <= '0';
130                 extender_en <= '1';
131
132         -- purpose of this state is to check for s_error and capture xy value
133             when motion_pressed =>
134                 clk1_en     <= '0';
135                 up1_down0_1 <= '0';
136                 clk2_en     <= '0';
137                 up1_down0_2 <= '0';
138                 error       <= '0';
139                 Capture_XY  <= '1';
140                 extender_en <= '1';
141
142
143         -- this is when machine goes mealy, x and y counters get individual inputs from
144         -- xy motion block according to the compx4 value
145             when moving =>
146                 clk1_en     <=  NOT X_EQ;
147                 up1_down0_1 <=  X_LT;
```

Cont.

```
148                 clk2_en     <=  NOT Y_EQ;
149                 up1_down0_2 <=  Y_LT;
150                 error       <= '0';
151                 Capture_XY  <= '0';
152                 extender_en <= '0';
153
154
155         -- s_error state essentially gives nothing to the system but the error message, therefore
156         -- everything else is being stopped
157             when system_error =>
158                 clk1_en     <= '0';
159                 up1_down0_1 <= '0';
160                 clk2_en     <= '0';
161                 up1_down0_2 <= '0';
162                 error       <= CLK;
163                 Capture_XY  <= '0';
164                 extender_en <= '0';
165
166         END CASE;
167
168     END PROCESS;
169 END XY_Logic;
```

VHDL Design: Bidirectional Shift Register

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

Entity Bidir_shift_reg is port
   (
         CLK            : in std_logic := '0';
         RESET          : in std_logic := '0';
         CLK_EN         : in std_logic := '0';
         LEFT0_RIGHT1   : in std_logic := '0';
         REG_BITS       : out std_logic_vector(3 downto 0)
   );
end Entity;

-- here the circuit begins          --
ARCHITECTURE one OF Bidir_shift_reg is

Signal sreg          : std_logic_vector(3 downto 0);

BEGIN

process (CLK, RESET) is
begin
   if (RESET = '1') then
            sreg <= "0000";

   elsif (rising_edge(CLK) AND (CLK_EN = '1')) then

         if (LEFT0_RIGHT1 = '1') then

            sreg (3 downto 0) <= '1' & sreg(3 downto 1);

         elsif (LEFT0_RIGHT1 = '0') then

            sreg (3 downto 0) <= sreg(2 downto 0) & '0';

         end if;

   end if;
   REG_BITS <= sreg;

end process;

END one;
```

VHDL Design: Target Register

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

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

Entity Target_Register is port
   (
      CLK            : in std_logic;
      Reset          : in std_logic;
      Capture        : in std_logic;
      Target_Value   : in std_logic_vector  (3 downto 0);
      Reg_Value      : out std_logic_vector (3 downto 0)
   );
END ENTITY;

Architecture R_Logic of Target_Register is


--          Here the Circuit Begins        --
BEGIN

   process (CLK, Reset, Capture) is
   BEGIN

      if (reset = '1') then
         Reg_Value <= "0000";
      elsif (Capture = '1' AND rising_edge(CLK)) then
         Reg_Value <= Target_Value;
      end if;

   end process;

END R_Logic;
--          Here everything ends           --
```

VHDL Design: Up/Down Binary Counter

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

Entity U_D_Bin_Counter4bit is port
    (
          CLK             : in std_logic := '0';
          RESET           : in std_logic := '0';
          CLK_EN          : in std_logic := '0';
          UP1_DOWN0       : in std_logic := '0';
          COUNTER_BITS    : out std_logic_vector(3 downto 0)
    );
end Entity;

--       here the circuit begins       --
ARCHITECTURE one OF U_D_Bin_Counter4bit IS

Signal ud_bin_counter   : UNSIGNED(3 downto 0);

BEGIN

--process (CLK, CLK_EN, RESET_n, UP1_DOWN0) is

process (CLK, RESET) is
begin
    if (RESET = '1') then
              ud_bin_counter <= "0000";

    elsif (rising_edge(CLK)) then

       if(( UP1_DOWN0 ='1') AND (CLK_EN ='1'))then
           ud_bin_counter <= (ud_bin_counter + 1);

       elsif (( UP1_DOWN0 ='0') AND (CLK_EN ='1')) then
           ud_bin_counter <= (ud_bin_counter - 1);

       end if;

    end if;

    COUNTER_BITS <= std_logic_vector(ud_bin_counter);

end process;

end;
```

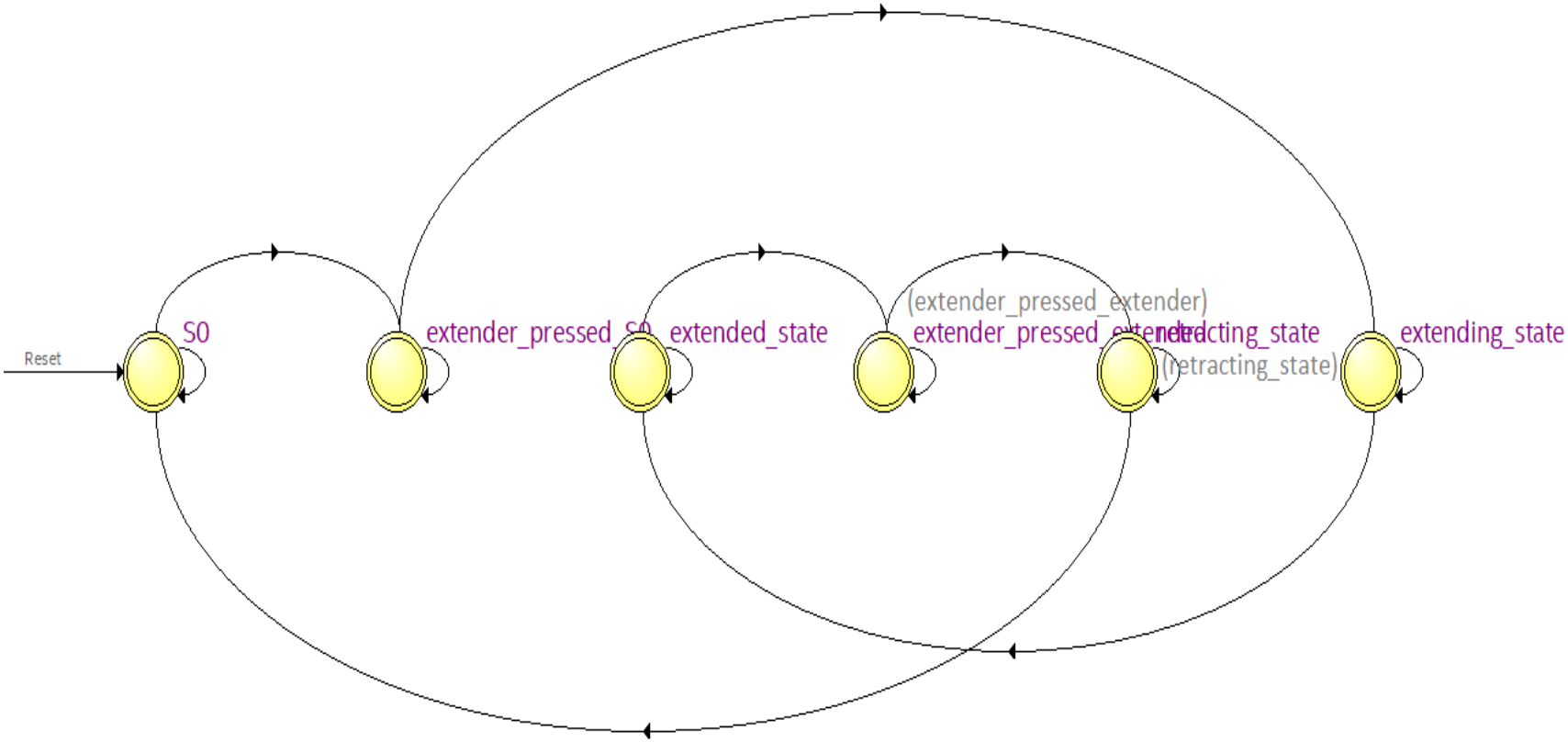
VHDL Design: Clock Source

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-- clk input is the LogicalStep 50MHz clock input
entity Clock_Source is

    port
    (
        SIM_FLAG            : in boolean;
        clk_input           : in std_logic;
        clock_out           : out std_logic
    );
end entity;
architecture rtl of Clock_Source is

signal clk_2hz          : std_logic;
signal sim_clock        : std_logic;
signal digital_counter  : std_logic_vector(23 downto 0);

begin

-- clk_divider process generates a 2Hz Clck from the 50 Mhz clk

clk_divider: process (clk_input)
    variable   counter   :  unsigned(23 downto 0);

    begin
-- Synchronously update counter
        if (rising_edge(clk_input))  then
                counter :=  counter + 1;
        end if;
        digital_counter <= std_logic_vector(counter);

    end process;

clk_2hz <= digital_counter(23);

clk_mux: process (SIM_FLAG)
    begin
        if (SIM_FLAG)  then
                clock_out<=  clk_input;      -- non-divided clock used for simulations
        else
                clock_out<=  clk_2hz;  -- 2Hz clock used for LogicalStep Board downloads
        end if;
    end process;
end rtl;
```

VHDL Design: Seven Segment

```vhdl
-- Chengfeng Deng, Tasviq Hossain
-- Group 9 Section 201

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

----------------------------------------------------------------------------

entity SevenSegment is port (

   hex      :  in  std_logic_vector(3 downto 0);   -- The 4 bit data to be displayed

   sevenseg :  out std_logic_vector(6 downto 0)    -- 7-bit outputs to a 7-segment
);
end SevenSegment;

architecture decode_function1 of SevenSegment is

--
-- The following statements convert a 4-bit input, called dataIn to a pattern of 7 bits.
-- The segment turns on when it is '1' otherwise '0'.
-- Correct the appropriate assignments for the seveseg outputs below for each of the various hex values.
--
begin
   with hex select            --GFEDCBA        3210      -- data in
   sevenseg                <= "0111111" when "0000",     -- [0]
                              "0000110" when "0001",     -- [1]
                              "1011011" when "0010",     -- [2]      +---- a -----+
                              "1001111" when "0011",     -- [3]      |            |
                              "1100110" when "0100",     -- [4]      |            |
                              "1101101" when "0101",     -- [5]      f            b
                              "1111101" when "0110",     -- [6]      |            |
                              "0000111" when "0111",     -- [7]      |            |
                              "1111111" when "1000",     -- [8]      +---- g -----+
                              "1101111" when "1001",     -- [9]      |            |
                              "1110111" when "1010",     -- [A]      |            |
                              "1111100" when "1011",     -- [b]      e            c
                              "0111001" when "1100",     -- [c]      |            |
                              "1011110" when "1101",     -- [d]      |            |
                              "1111001" when "1110",     -- [E]      +---- d -----+
                              "1110001" when "1111",     -- [F]
                              "0000000" when others;     -- [ ]

end architecture decode_function1;
--------------------------------------------------------------------------
```

State Diagram: Extender



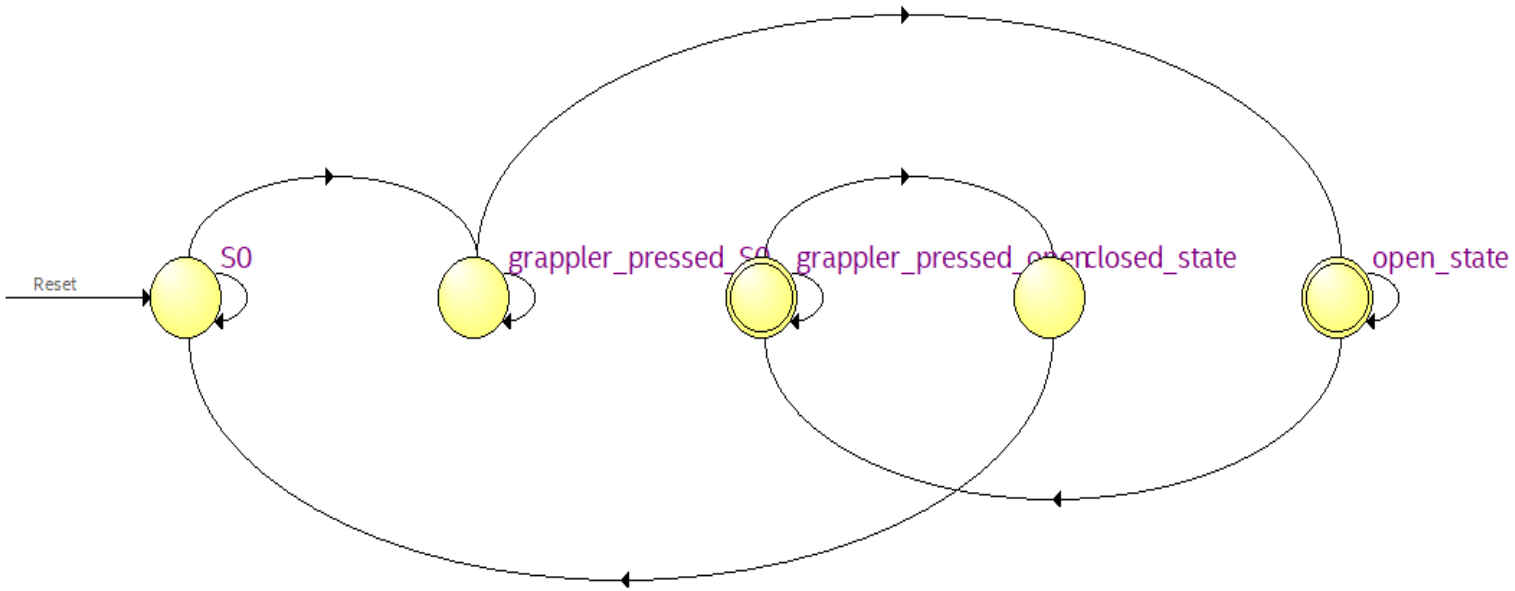
Transition Table: Extender

State Table

	Source State	Destination State	Condition
1	extended_state	extender_pressed_extended	(Extender)
2	extended_state	extended_state	(!Extender)
3	extender_pressed_extended	extender_pressed_extended	(Extender)
4	extender_pressed_extended	retracting_state	(!Extender)
5	extender_pressed_S0	extender_pressed_S0	(Extender)
6	extender_pressed_S0	extending_state	(!Extender)
7	extending_state	extended_state	(ext_pos[...
8	extending_state	extending_state	(!ext_pos...
9	retracting_state	retracting_state	(!ext_pos...
10	retracting_state	S0	(!ext_pos...
11	S0	extender_pressed_S0	(TRANSI...
12	S0	S0	(!TRANSI...

Transitions | Encoding

State Diagram: Grappler



Transition Table: Grappler

State Table

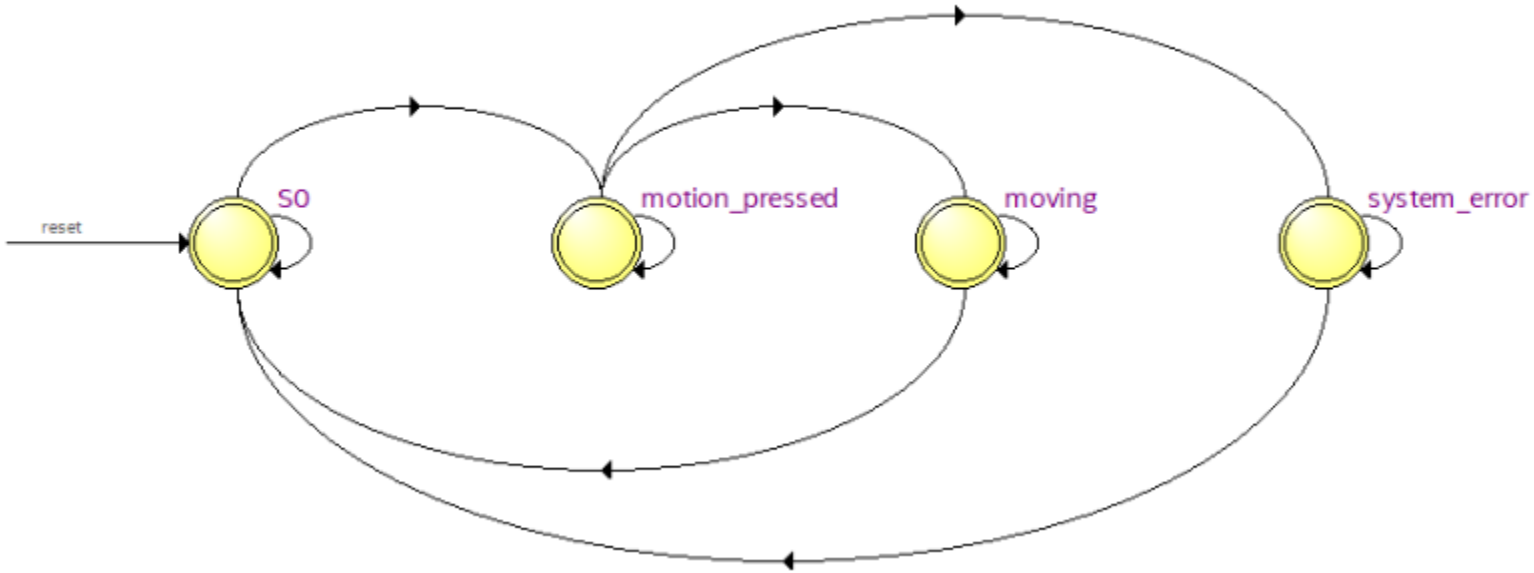
	Source State	Destination State	Condition
1	closed_state	S0	
2	grappler_pressed_open	closed_state	(!grappler)
3	grappler_pressed_open	grappler_pressed_open	(grappler)
4	grappler_pressed_S0	open_state	(!grappler)
5	grappler_pressed_S0	grappler_pressed_S0	(grappler)

Transitions | Encoding

6	open_state	grappler_pressed_open	(TRANSITION_SECTION)
7	open_state	open_state	(!TRANSITION_SECTION)
8	S0	S0	(!TRANSITION_SECTION)
9	S0	grappler_pressed_S0	(TRANSITION_SECTION)

Transitions | Encoding

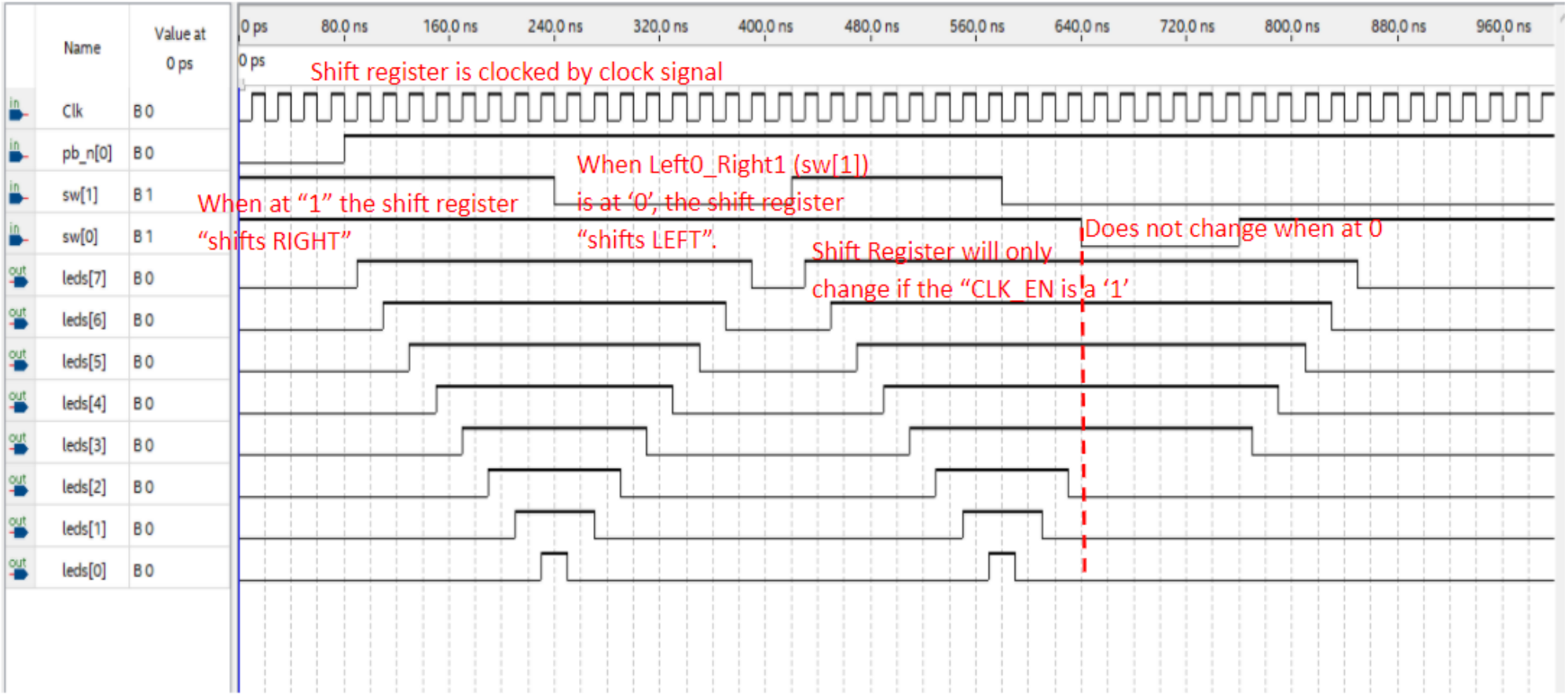
State Diagram: XY Motion



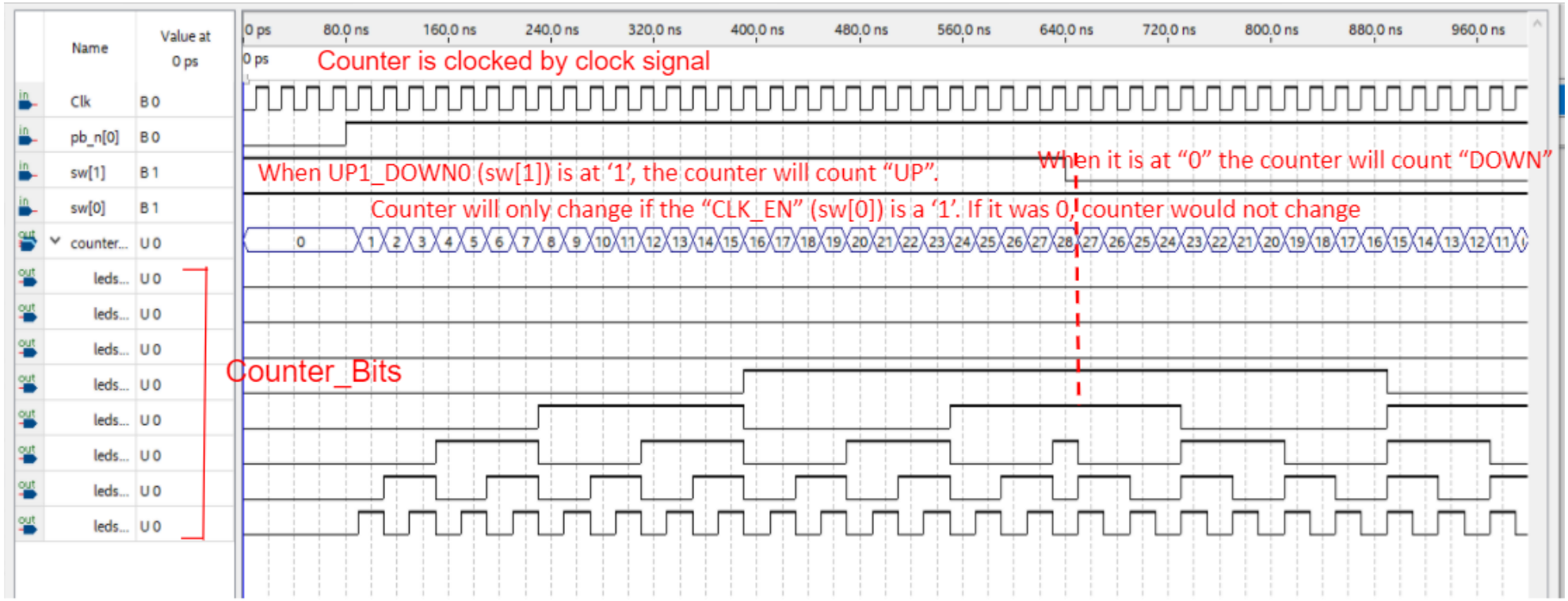
Transition Table: XY Motion

	Source State	Destination State	Condition
1	motion_pressed	system_error	(s_error).(!motion)
2	motion_pressed	moving	(!s_error).(!motion)
3	motion_pressed	motion_pressed	(motion)
4	moving	S0	(motion_done)
5	moving	moving	(!motion_done)
6	S0	S0	(!motion)
7	S0	motion_pressed	(motion)
8	system_error	system_error	(extender_out)

Functional Simulation of the Bidirectional Shift Register

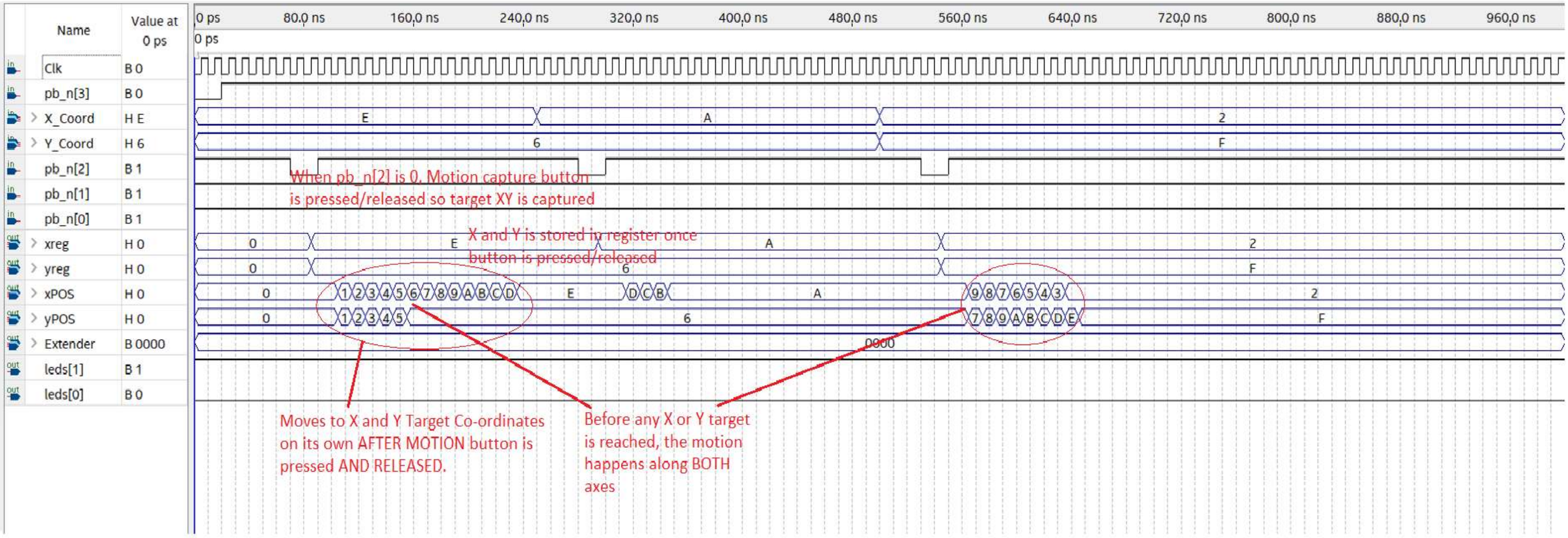


Functional Simulation of the Up/Down Binary Counter



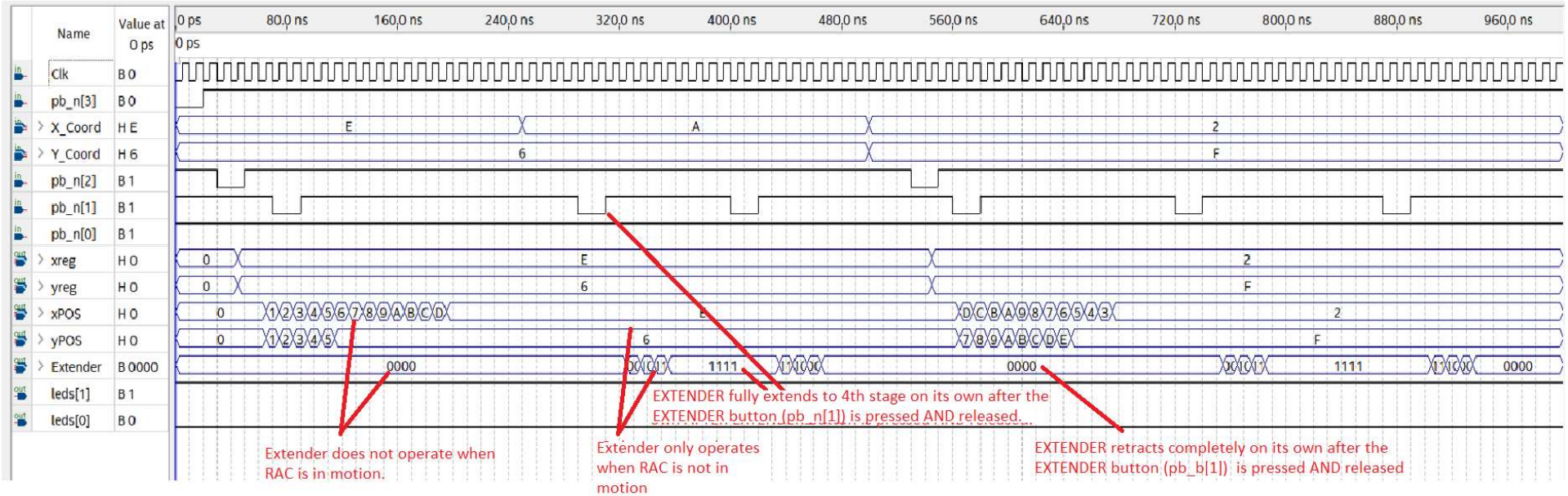
Simulation 1: Functionality of XY Motion

Chengfeng Deng,
Tasviq Hossain
Group 9 Section 201



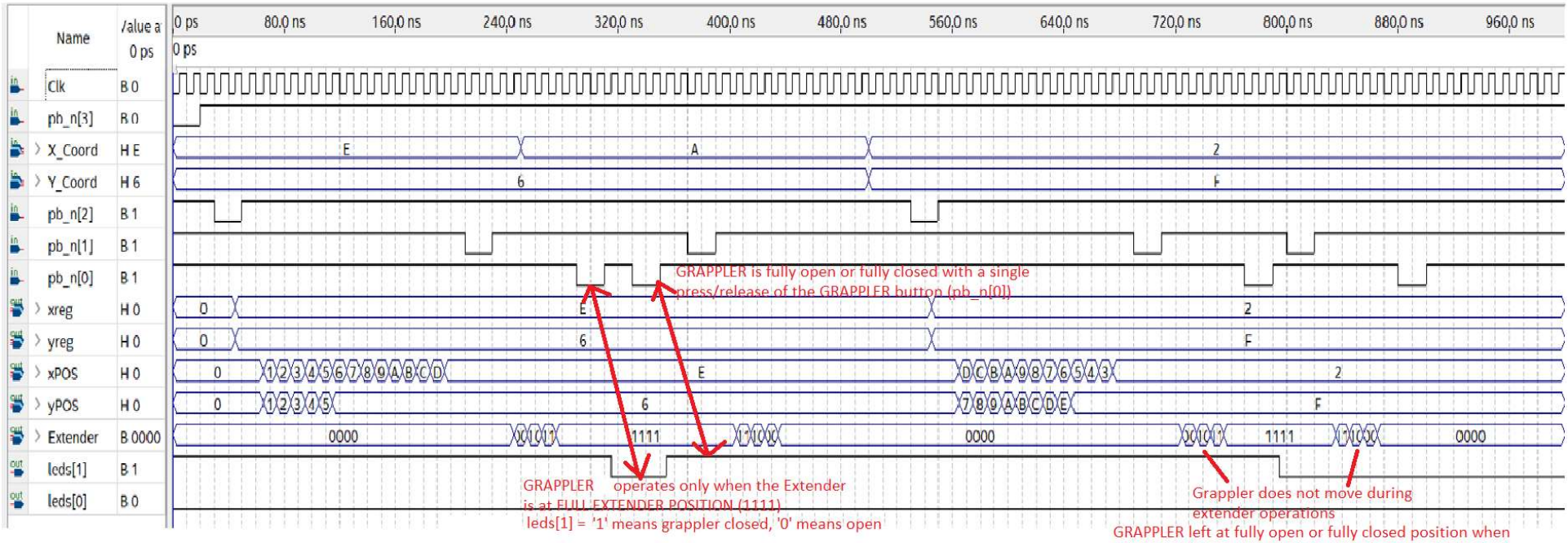
Chengfeng Deng, Tasviq Hossain
Group 9 Section 201

Simulation 2: Extender Operations



Simulation 3: Grappler Operations

Chengfeng Deng, Tasviq Hossain
Group 9 Section 201



Chengfeng Deng, Tasviq Hossain
Group 9 Section 201

Simulation 4