Lab 4 Report

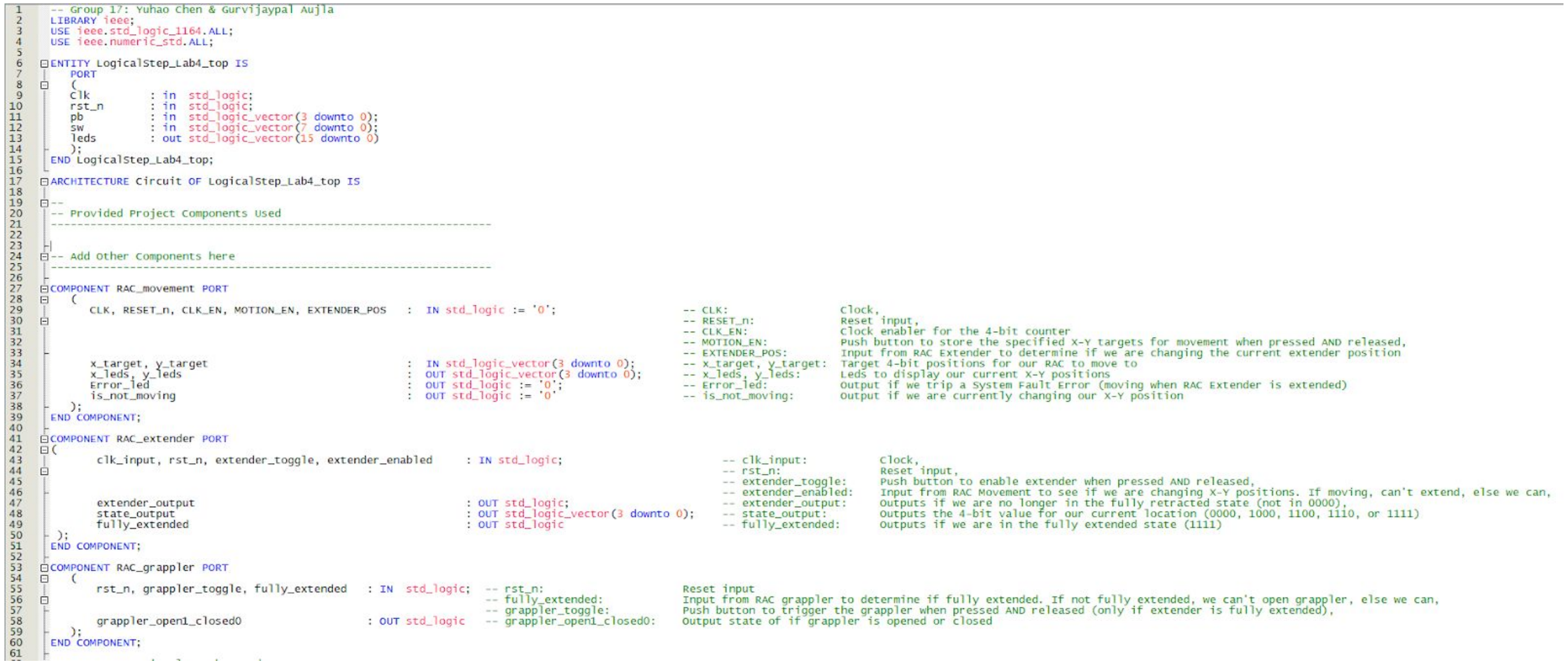
ECE 124

Group 17 Session 204

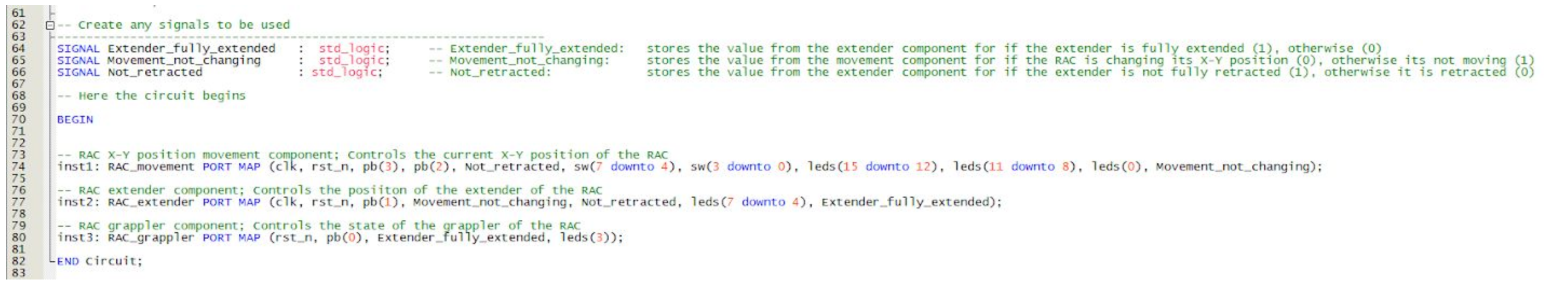
Gurvijaypal Aujla

Yuhao Chen

Top-level Design

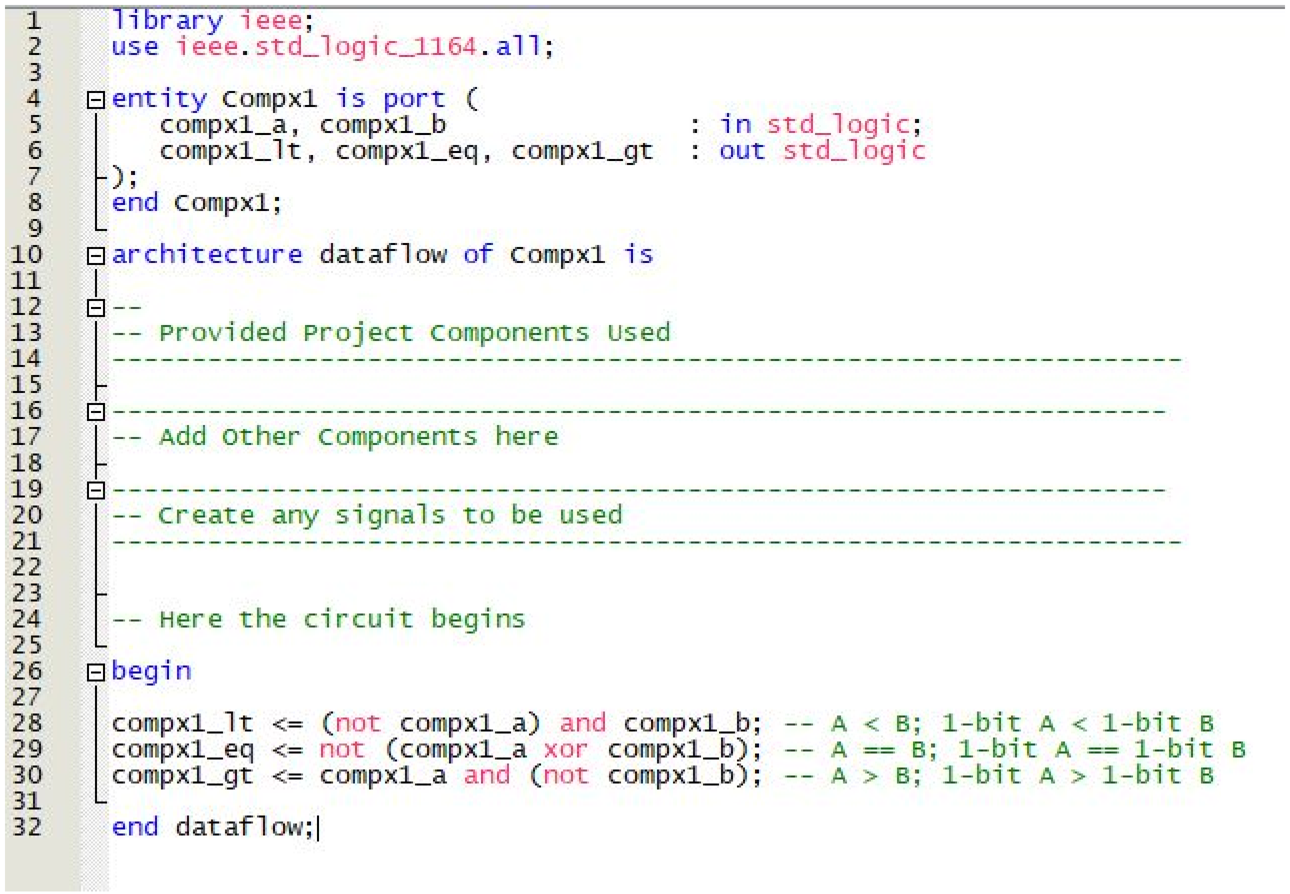


```vhdl
-- Group 17: Yuhao Chen & Gurvijaypal Aujla
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY LogicalStep_Lab4_top IS
   PORT
   (
   Clk          : in  std_logic;
   rst_n        : in  std_logic;
   pb           : in  std_logic_vector(3 downto 0);
   sw           : in  std_logic_vector(7 downto 0);
   leds         : out std_logic_vector(15 downto 0)
   );
END LogicalStep_Lab4_top;

ARCHITECTURE Circuit OF LogicalStep_Lab4_top IS

--
-- Provided Project Components Used
------------------------------------------------------------------


-- Add Other Components here
------------------------------------------------------------------

COMPONENT RAC_movement PORT
   (
      CLK, RESET_n, CLK_EN, MOTION_EN, EXTENDER_POS   :  IN std_logic := '0';                    -- CLK:                Clock,
                                                                                                  -- RESET_n:            Reset input,
                                                                                                  -- CLK_EN:             Clock enabler for the 4-bit counter
                                                                                                  -- MOTION_EN:          Push button to store the specified X-Y targets for movement when pressed AND released,
                                                                                                  -- EXTENDER_POS:       Input from RAC Extender to determine if we are changing the current extender position
      x_target, y_target                              :  IN std_logic_vector(3 downto 0);        -- x_target, y_target: Target 4-bit positions for our RAC to move to
      x_leds, y_leds                                  :  OUT std_logic_vector(3 downto 0);       -- x_leds, y_leds:     Leds to display our current X-Y positions
      Error_led                                       :  OUT std_logic := '0';                   -- Error_led:          Output if we trip a System Fault Error (moving when RAC Extender is extended)
      is_not_moving                                   :  OUT std_logic := '0'                    -- is_not_moving:      Output if we are currently changing our X-Y position
   );
END COMPONENT;

COMPONENT RAC_extender PORT
(
       clk_input, rst_n, extender_toggle, extender_enabled     : IN std_logic;                       -- clk_input:          Clock,
                                                                                                       -- rst_n:              Reset input,
                                                                                                       -- extender_toggle:    Push button to enable extender when pressed AND released,
                                                                                                       -- extender_enabled:   Input from RAC Movement to see if we are changing X-Y positions. If moving, can't extend, else we can,
       extender_output                                         : OUT std_logic;                      -- extender_output:    Outputs if we are no longer in the fully retracted state (not in 0000),
       state_output                                            : OUT std_logic_vector(3 downto 0);   -- state_output:       Outputs the 4-bit value for our current location (0000, 1000, 1100, 1110, or 1111)
       fully_extended                                          : OUT std_logic                       -- fully_extended:     Outputs if we are in the fully extended state (1111)
 );
END COMPONENT;

COMPONENT RAC_grappler PORT
   (
       rst_n, grappler_toggle, fully_extended   : IN  std_logic;  -- rst_n:                      Reset input
                                                                  -- fully_extended:             Input from RAC grappler to determine if fully extended. If not fully extended, we can't open grappler, else we can,
                                                                  -- grappler_toggle:            Push button to trigger the grappler when pressed AND released (only if extender is fully extended),
       grappler_open1_closed0                   : OUT std_logic   -- grappler_open1_closed0:     Output state of if grappler is opened or closed
   );
END COMPONENT;

```



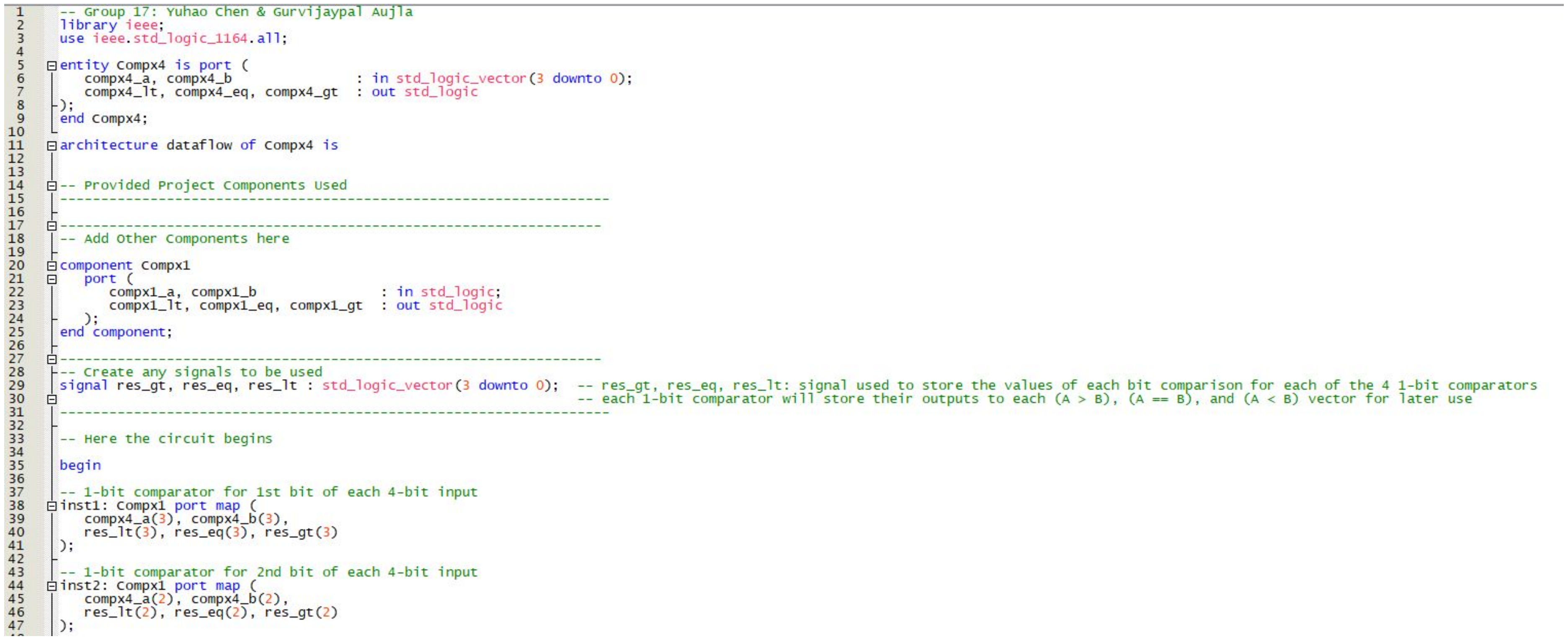
```vhdl

-- Create any signals to be used
------------------------------------------------------------------
SIGNAL Extender_fully_extended   :  std_logic;      -- Extender_fully_extended:   stores the value from the extender component for if the extender is fully extended (1), otherwise (0)
SIGNAL Movement_not_changing     :  std_logic;      -- Movement_not_changing:     stores the value from the movement component for if the RAC is changing its X-Y position (0), otherwise its not moving (1)
SIGNAL Not_retracted             : std_logic;       -- Not_retracted:             stores the value from the extender component for if the extender is not fully retracted (1), otherwise it is retracted (0)

-- Here the circuit begins

BEGIN


-- RAC X-Y position movement component; Controls the current X-Y position of the RAC
inst1: RAC_movement PORT MAP (clk, rst_n, pb(3), pb(2), Not_retracted, sw(7 downto 4), sw(3 downto 0), leds(15 downto 12), leds(11 downto 8), leds(0), Movement_not_changing);

-- RAC extender component; Controls the posiiton of the extender of the RAC
inst2: RAC_extender PORT MAP (clk, rst_n, pb(1), Movement_not_changing, Not_retracted, leds(7 downto 4), Extender_fully_extended);

-- RAC grappler component; Controls the state of the grappler of the RAC
inst3: RAC_grappler PORT MAP (rst_n, pb(0), Extender_fully_extended, leds(3));

END Circuit;

```

1-bit Comparator Design

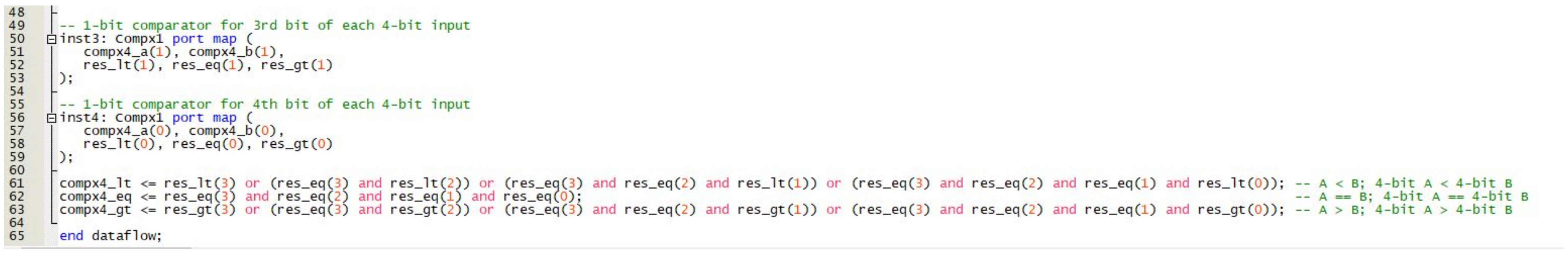


```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity Compx1 is port (
    compx1_a, compx1_b                  : in std_logic;
    compx1_lt, compx1_eq, compx1_gt   : out std_logic
);
end Compx1;

architecture dataflow of Compx1 is

--
-- Provided Project Components Used
--------------------------------------------------------------------

-------------------------------------------------------------------
-- Add Other Components here

-------------------------------------------------------------------
-- Create any signals to be used
--------------------------------------------------------------------


-- Here the circuit begins

begin

compx1_lt <= (not compx1_a) and compx1_b; -- A < B; 1-bit A < 1-bit B
compx1_eq <= not (compx1_a xor compx1_b); -- A == B; 1-bit A == 1-bit B
compx1_gt <= compx1_a and (not compx1_b); -- A > B; 1-bit A > 1-bit B

end dataflow;
```

4-bit Comparator Design

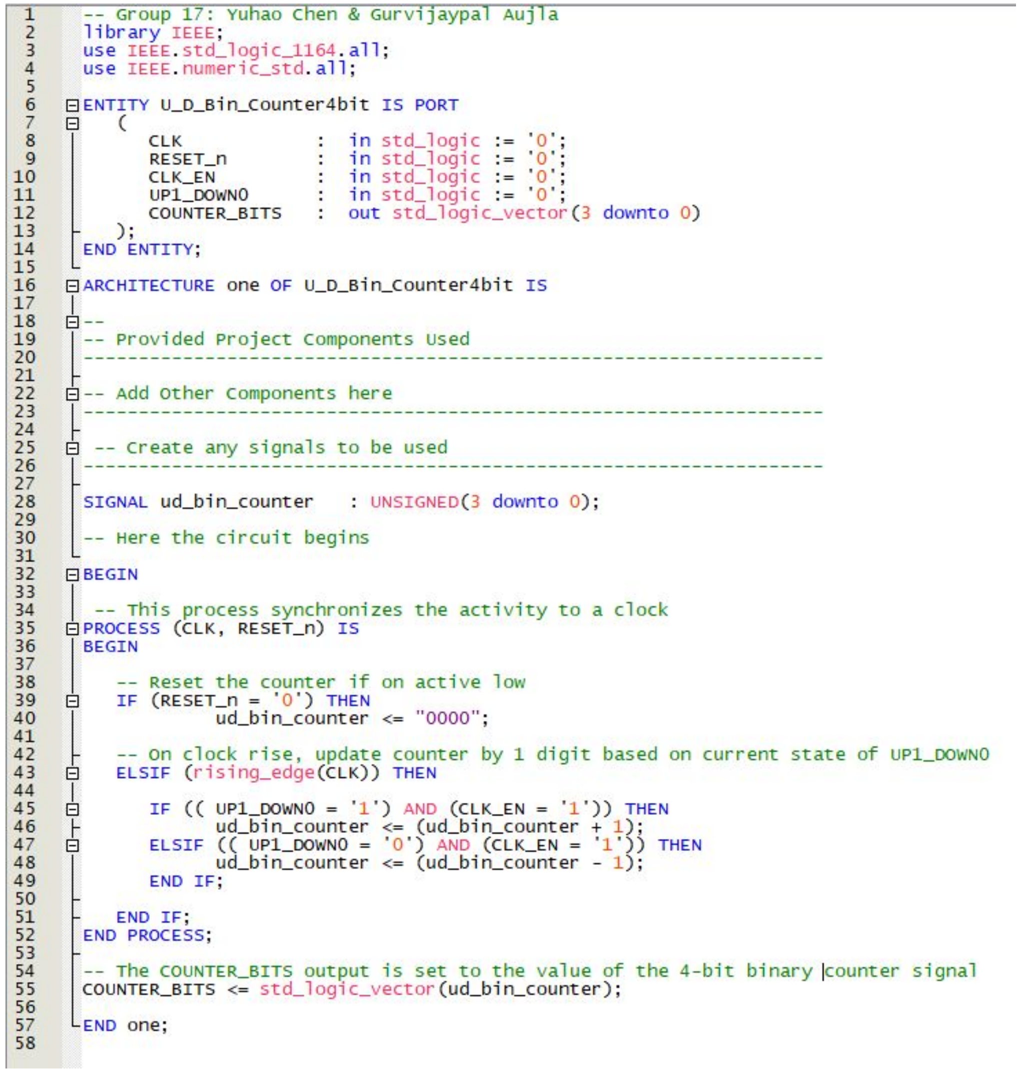


```vhdl
-- Group 17: Yuhao Chen & Gurvijaypal Aujla
library ieee;
use ieee.std_logic_1164.all;

entity Compx4 is port (
    compx4_a, compx4_b              : in std_logic_vector(3 downto 0);
    compx4_lt, compx4_eq, compx4_gt  : out std_logic
);
end Compx4;

architecture dataflow of Compx4 is


-- Provided Project Components Used
------------------------------------------------------------------

------------------------------------------------------------------
-- Add Other Components here

component Compx1
    port (
        compx1_a, compx1_b              : in std_logic;
        compx1_lt, compx1_eq, compx1_gt  : out std_logic
    );
end component;

------------------------------------------------------------------
-- Create any signals to be used
signal res_gt, res_eq, res_lt : std_logic_vector(3 downto 0);  -- res_gt, res_eq, res_lt: signal used to store the values of each bit comparison for each of the 4 1-bit comparators
                                                                -- each 1-bit comparator will store their outputs to each (A > B), (A == B), and (A < B) vector for later use
------------------------------------------------------------------

-- Here the circuit begins

begin

-- 1-bit comparator for 1st bit of each 4-bit input
inst1: Compx1 port map (
    compx4_a(3), compx4_b(3),
    res_lt(3), res_eq(3), res_gt(3)
);

-- 1-bit comparator for 2nd bit of each 4-bit input
inst2: Compx1 port map (
    compx4_a(2), compx4_b(2),
    res_lt(2), res_eq(2), res_gt(2)
);
```



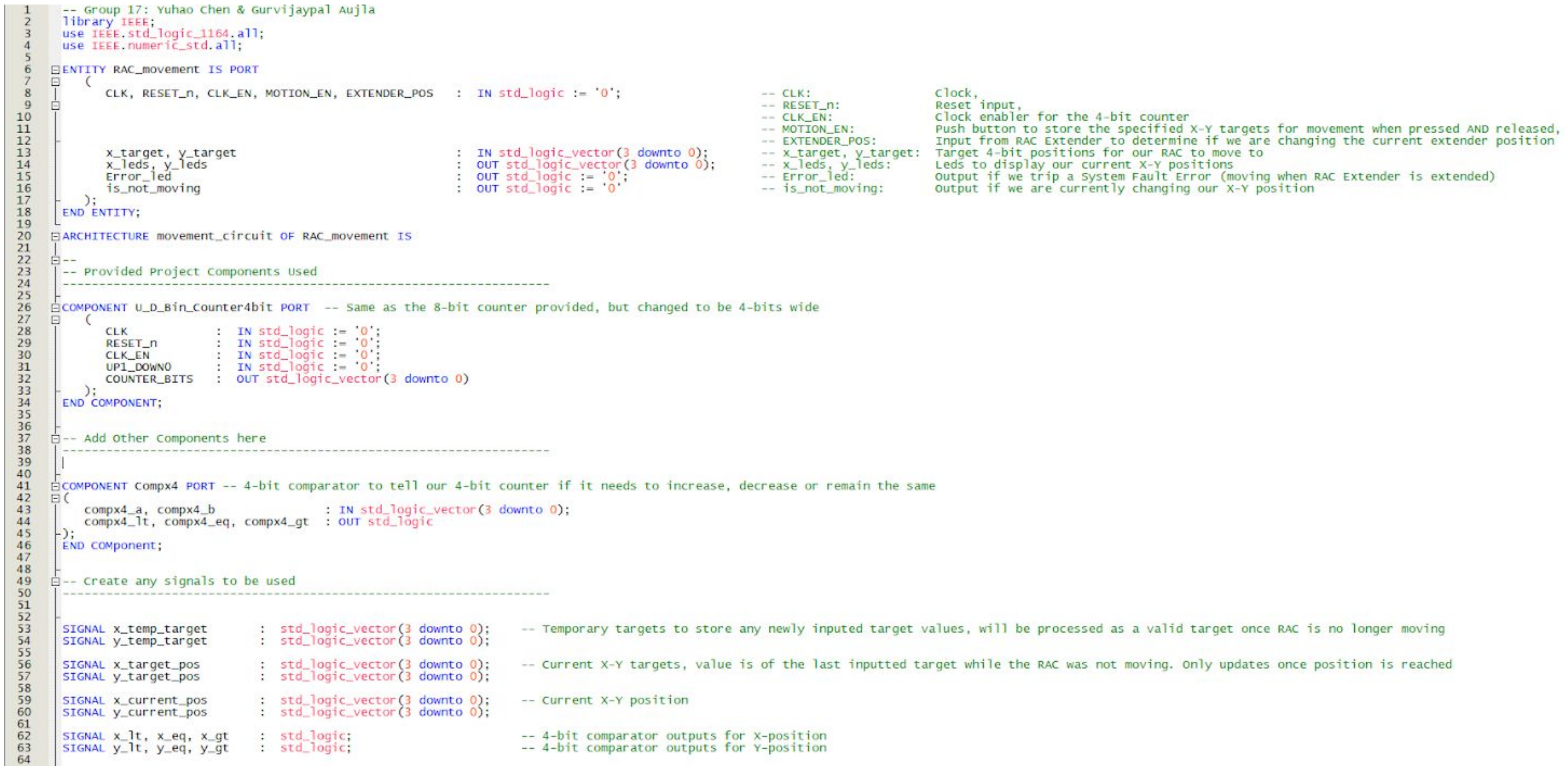
```vhdl

-- 1-bit comparator for 3rd bit of each 4-bit input
inst3: Compx1 port map (
    compx4_a(1), compx4_b(1),
    res_lt(1), res_eq(1), res_gt(1)
);

-- 1-bit comparator for 4th bit of each 4-bit input
inst4: Compx1 port map (
    compx4_a(0), compx4_b(0),
    res_lt(0), res_eq(0), res_gt(0)
);

compx4_lt <= res_lt(3) or (res_eq(3) and res_lt(2)) or (res_eq(3) and res_eq(2) and res_lt(1)) or (res_eq(3) and res_eq(2) and res_eq(1) and res_lt(0)); -- A < B; 4-bit A < 4-bit B
compx4_eq <= res_eq(3) and res_eq(2) and res_eq(1) and res_eq(0);                                                                                          -- A == B; 4-bit A == 4-bit B
compx4_gt <= res_gt(3) or (res_eq(3) and res_gt(2)) or (res_eq(3) and res_eq(2) and res_gt(1)) or (res_eq(3) and res_eq(2) and res_eq(1) and res_gt(0)); -- A > B; 4-bit A > 4-bit B

end dataflow;
```

4-bit Up/Down Binary Counter Design



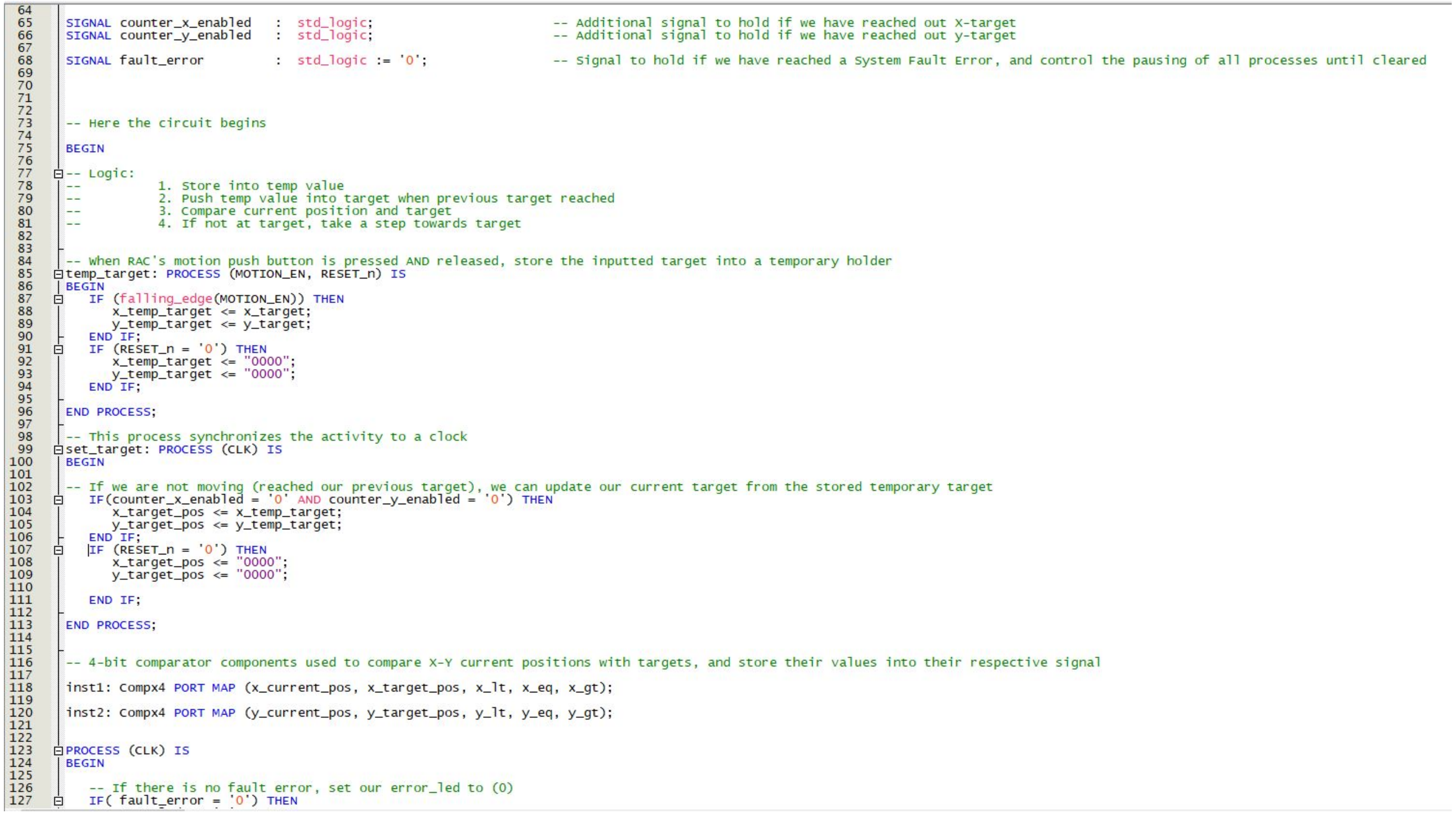
```vhdl
-- Group 17: Yuhao Chen & Gurvijaypal Aujla
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

ENTITY U_D_Bin_Counter4bit IS PORT
    (
        CLK             :  in std_logic := '0';
        RESET_n         :  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;

ARCHITECTURE one OF U_D_Bin_Counter4bit IS

--
-- Provided Project Components Used
--------------------------------------------------------------------

-- Add Other Components here
--------------------------------------------------------------------

 -- Create any signals to be used
--------------------------------------------------------------------

SIGNAL ud_bin_counter    : UNSIGNED(3 downto 0);

-- Here the circuit begins

BEGIN

 -- This process synchronizes the activity to a clock
PROCESS (CLK, RESET_n) IS
BEGIN

    -- Reset the counter if on active low
    IF (RESET_n = '0') THEN
            ud_bin_counter <= "0000";

    -- On clock rise, update counter by 1 digit based on current state of UP1_DOWN0
    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;
END PROCESS;

-- The COUNTER_BITS output is set to the value of the 4-bit binary counter signal
COUNTER_BITS <= std_logic_vector(ud_bin_counter);

END one;
```

RAC Movement Design PART 1/3



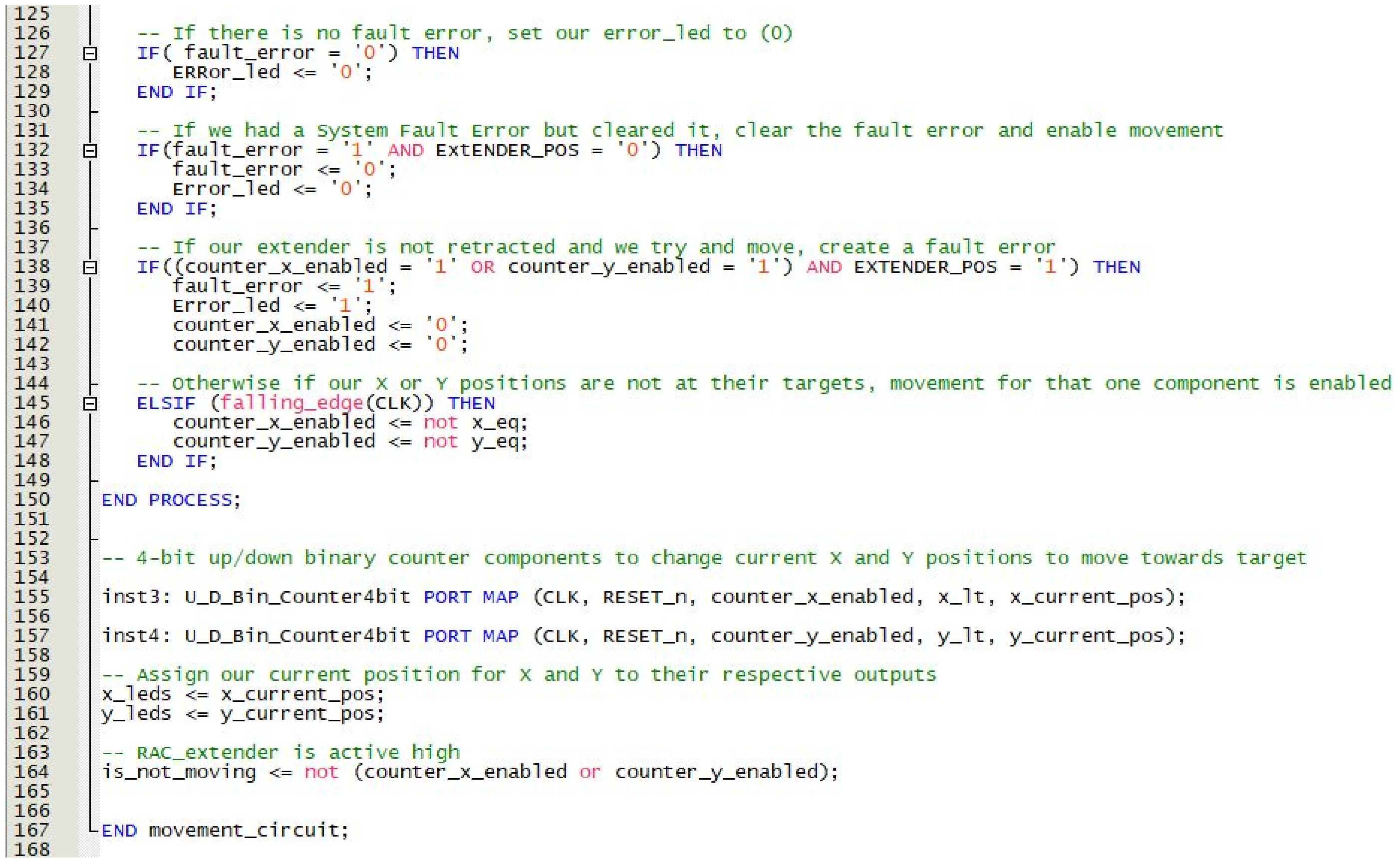
```vhdl
-- Group 17: Yuhao Chen & Gurvijaypal Aujla
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

ENTITY RAC_movement IS PORT
   (
      CLK, RESET_n, CLK_EN, MOTION_EN, EXTENDER_POS    :  IN std_logic := '0';                       -- CLK:                  Clock,
                                                                                                     -- RESET_n:              Reset input,
                                                                                                     -- CLK_EN:               Clock enabler for the 4-bit counter
                                                                                                     -- MOTION_EN:            Push button to store the specified X-Y targets for movement when pressed AND released,
                                                                                                     -- EXTENDER_POS:         Input from RAC Extender to determine if we are changing the current extender position
      x_target, y_target                               :  IN std_logic_vector(3 downto 0);           -- x_target, y_target:   Target 4-bit positions for our RAC to move to
      x_leds, y_leds                                   :  OUT std_logic_vector(3 downto 0);          -- x_leds, y_leds:       Leds to display our current X-Y positions
      Error_led                                        :  OUT std_logic := '0';                      -- Error_led:            Output if we trip a System Fault Error (moving when RAC Extender is extended)
      is_not_moving                                    :  OUT std_logic := '0'                       -- is_not_moving:        Output if we are currently changing our X-Y position
   );
END ENTITY;

ARCHITECTURE movement_circuit OF RAC_movement IS

--
-- Provided Project Components Used
------------------------------------------------------------------

COMPONENT U_D_Bin_Counter4bit PORT  -- Same as the 8-bit counter provided, but changed to be 4-bits wide
   (
      CLK            :  IN std_logic := '0';
      RESET_n        :  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 COMPONENT;

-- Add Other Components here
------------------------------------------------------------------


COMPONENT Compx4 PORT -- 4-bit comparator to tell our 4-bit counter if it needs to increase, decrease or remain the same
(
   compx4_a, compx4_b                  : IN std_logic_vector(3 downto 0);
   compx4_lt, compx4_eq, compx4_gt    : OUT std_logic
);
END Component;

-- Create any signals to be used
------------------------------------------------------------------


SIGNAL x_temp_target       :  std_logic_vector(3 downto 0);    -- Temporary targets to store any newly inputed target values, will be processed as a valid target once RAC is no longer moving
SIGNAL y_temp_target       :  std_logic_vector(3 downto 0);

SIGNAL x_target_pos        :  std_logic_vector(3 downto 0);    -- Current X-Y targets, value is of the last inputted target while the RAC was not moving. Only updates once position is reached
SIGNAL y_target_pos        :  std_logic_vector(3 downto 0);

SIGNAL x_current_pos       :  std_logic_vector(3 downto 0);    -- Current X-Y position
SIGNAL y_current_pos       :  std_logic_vector(3 downto 0);

SIGNAL x_lt, x_eq, x_gt    :  std_logic;                       -- 4-bit comparator outputs for X-position
SIGNAL y_lt, y_eq, y_gt    :  std_logic;                       -- 4-bit comparator outputs for Y-position

```

RAC Movement Design PART 2/3



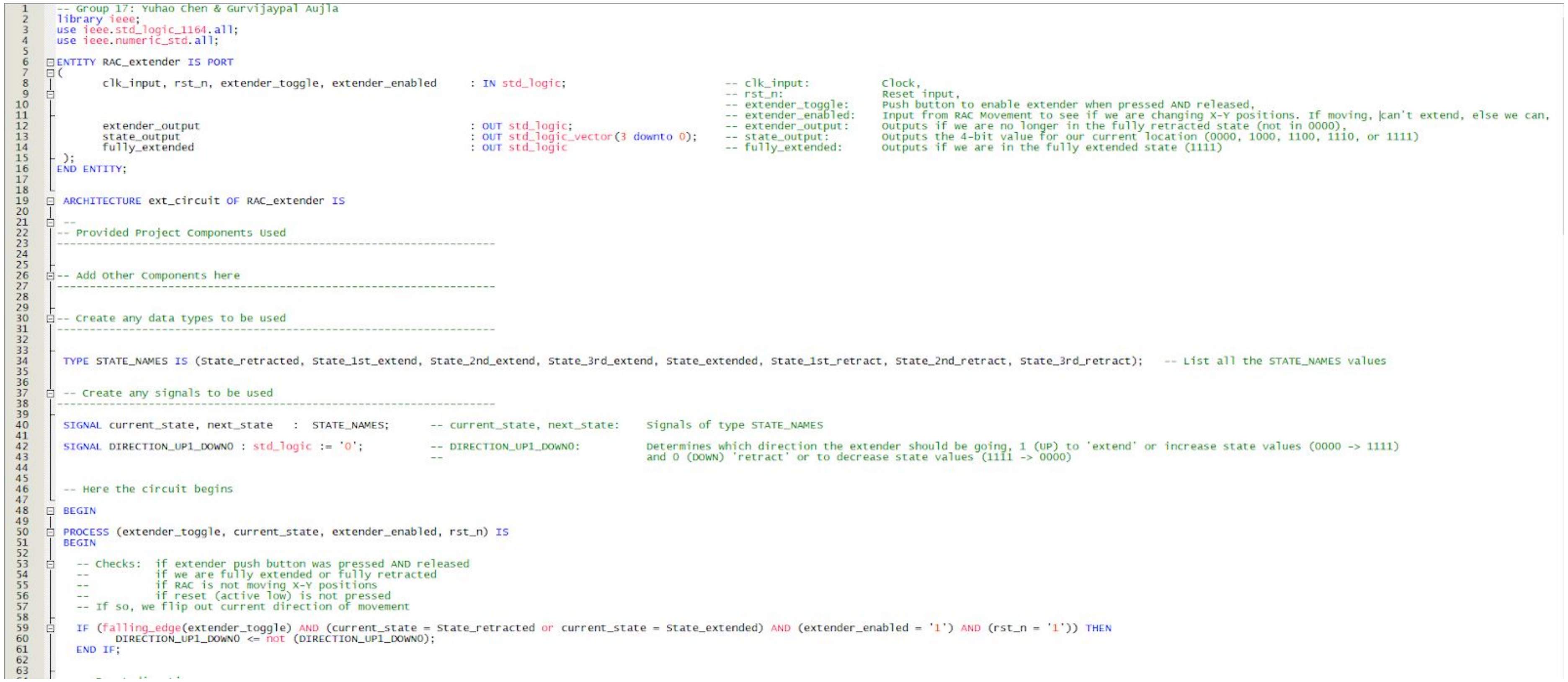
```vhdl

SIGNAL counter_x_enabled   :  std_logic;                       -- Additional signal to hold if we have reached out X-target
SIGNAL counter_y_enabled   :  std_logic;                       -- Additional signal to hold if we have reached out y-target

SIGNAL fault_error         :  std_logic := '0';                -- Signal to hold if we have reached a System Fault Error, and control the pausing of all processes until cleared




-- Here the circuit begins

BEGIN

-- Logic:
--          1. Store into temp value
--          2. Push temp value into target when previous target reached
--          3. Compare current position and target
--          4. If not at target, take a step towards target


-- When RAC's motion push button is pressed AND released, store the inputted target into a temporary holder
temp_target: PROCESS (MOTION_EN, RESET_n) IS
BEGIN
   IF (falling_edge(MOTION_EN)) THEN
      x_temp_target <= x_target;
      y_temp_target <= y_target;
   END IF;
   IF (RESET_n = '0') THEN
      x_temp_target <= "0000";
      y_temp_target <= "0000";
   END IF;

END PROCESS;

-- This process synchronizes the activity to a clock
set_target: PROCESS (CLK) IS
BEGIN

-- If we are not moving (reached our previous target), we can update our current target from the stored temporary target
   IF(counter_x_enabled = '0' AND counter_y_enabled = '0') THEN
      x_target_pos <= x_temp_target;
      y_target_pos <= y_temp_target;
   END IF;
   IF (RESET_n = '0') THEN
      x_target_pos <= "0000";
      y_target_pos <= "0000";

   END IF;

END PROCESS;


-- 4-bit comparator components used to compare X-Y current positions with targets, and store their values into their respective signal

inst1: Compx4 PORT MAP (x_current_pos, x_target_pos, x_lt, x_eq, x_gt);

inst2: Compx4 PORT MAP (y_current_pos, y_target_pos, y_lt, y_eq, y_gt);


PROCESS (CLK) IS
BEGIN

   -- If there is no fault error, set our error_led to (0)
   IF( fault_error = '0') THEN
```

RAC Movement Design PART 3/3



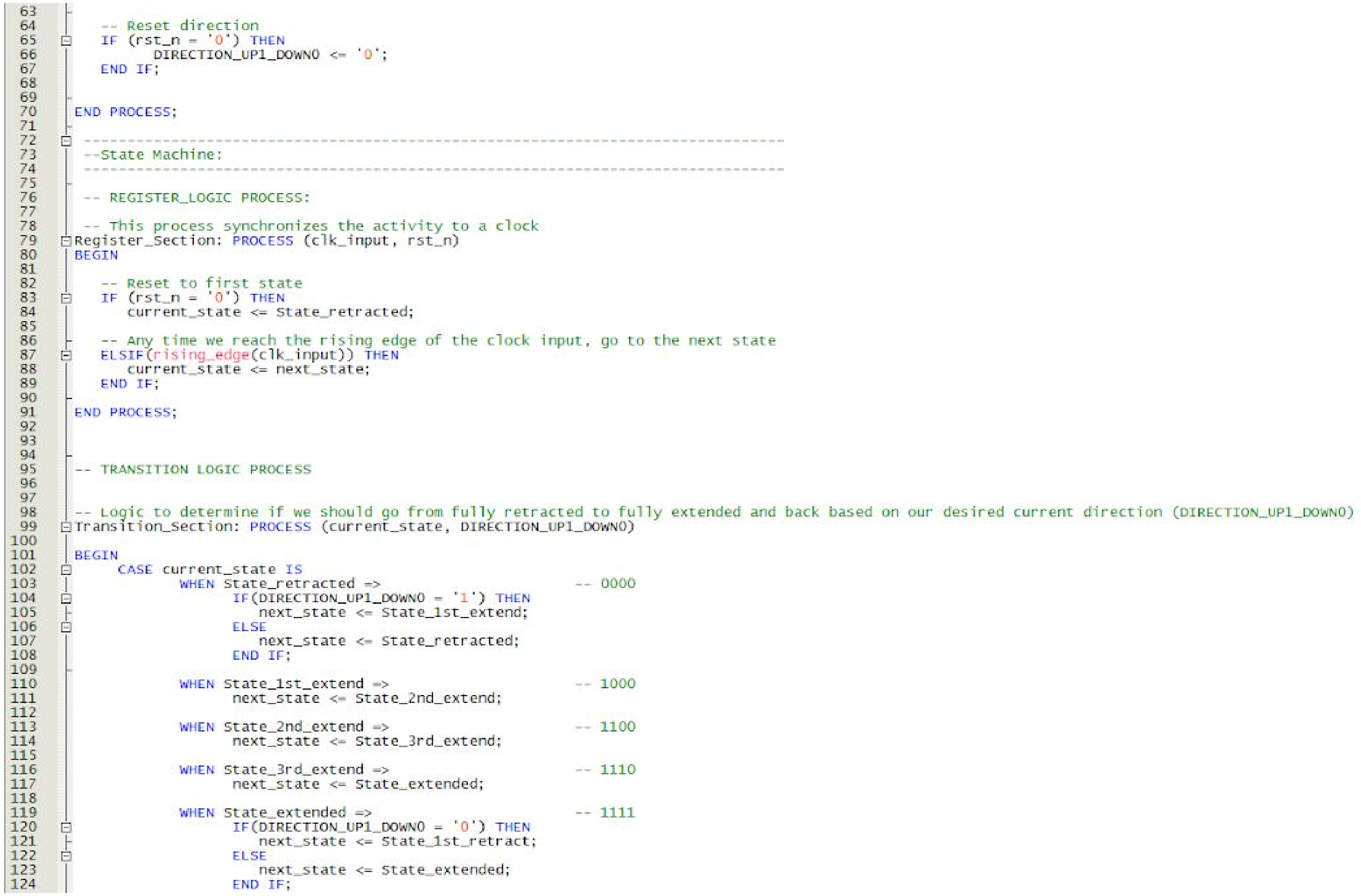
```
   -- If there is no fault error, set our error_led to (0)
   IF( fault_error = '0') THEN
      ERRor_led <= '0';
   END IF;

   -- If we had a System Fault Error but cleared it, clear the fault error and enable movement
   IF(fault_error = '1' AND ExtENDER_POS = '0') THEN
      fault_error <= '0';
      Error_led <= '0';
   END IF;

   -- If our extender is not retracted and we try and move, create a fault error
   IF((counter_x_enabled = '1' OR counter_y_enabled = '1') AND EXTENDER_POS = '1') THEN
      fault_error <= '1';
      Error_led <= '1';
      counter_x_enabled <= '0';
      counter_y_enabled <= '0';

   -- Otherwise if our X or Y positions are not at their targets, movement for that one component is enabled
   ELSIF (falling_edge(CLK)) THEN
      counter_x_enabled <= not x_eq;
      counter_y_enabled <= not y_eq;
   END IF;

END PROCESS;


-- 4-bit up/down binary counter components to change current X and Y positions to move towards target

inst3: U_D_Bin_Counter4bit PORT MAP (CLK, RESET_n, counter_x_enabled, x_lt, x_current_pos);

inst4: U_D_Bin_Counter4bit PORT MAP (CLK, RESET_n, counter_y_enabled, y_lt, y_current_pos);

-- Assign our current position for X and Y to their respective outputs
x_leds <= x_current_pos;
y_leds <= y_current_pos;

-- RAC_extender is active high
is_not_moving <= not (counter_x_enabled or counter_y_enabled);


END movement_circuit;
```

RAC Extender Design PART 1/3



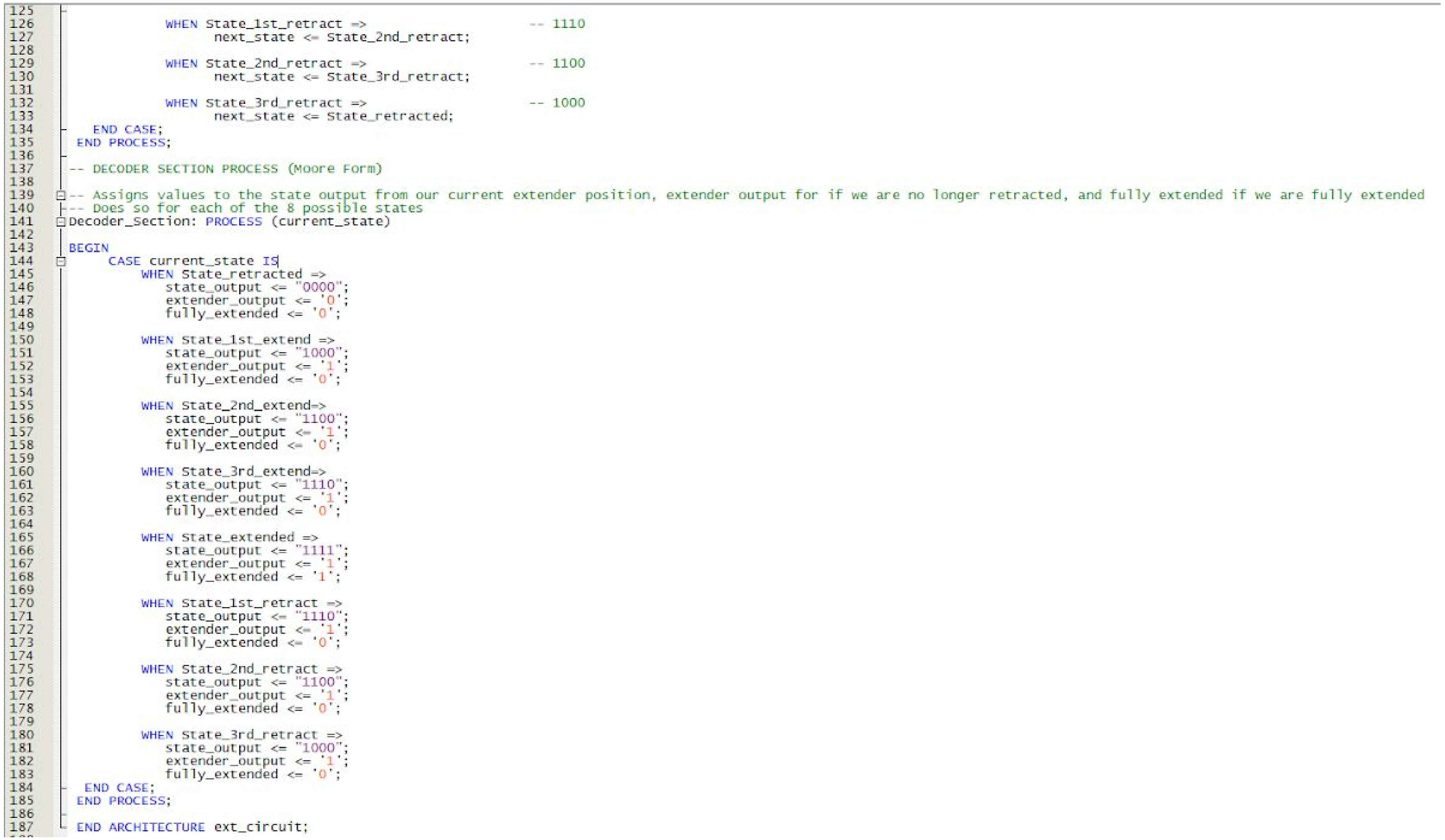
```vhdl
-- Group 17: Yuhao Chen & Gurvijaypal Aujla
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY RAC_extender IS PORT
(
       clk_input, rst_n, extender_toggle, extender_enabled     : IN std_logic;                               -- clk_input:           Clock,
                                                                                                              -- rst_n:               Reset input,
                                                                                                              -- extender_toggle:     Push button to enable extender when pressed AND released,
                                                                                                              -- extender_enabled:    Input from RAC Movement to see if we are changing X-Y positions. If moving, can't extend, else we can,
       extender_output                                         : OUT std_logic;                              -- extender_output:     Outputs if we are no longer in the fully retracted state (not in 0000),
       state_output                                            : OUT std_logic_vector(3 downto 0);           -- state_output:        Outputs the 4-bit value for our current location (0000, 1000, 1100, 1110, or 1111)
       fully_extended                                          : OUT std_logic                               -- fully_extended:      Outputs if we are in the fully extended state (1111)
 );
END ENTITY;


 ARCHITECTURE ext_circuit OF RAC_extender IS

 --
-- Provided Project Components Used
------------------------------------------------------------------


-- Add Other Components here
------------------------------------------------------------------


-- Create any data types to be used
------------------------------------------------------------------


 TYPE STATE_NAMES IS (State_retracted, State_1st_extend, State_2nd_extend, State_3rd_extend, State_extended, State_1st_retract, State_2nd_retract, State_3rd_retract);   -- List all the STATE_NAMES values


 -- Create any signals to be used
------------------------------------------------------------------

 SIGNAL current_state, next_state   :  STATE_NAMES;      -- current_state, next_state:    Signals of type STATE_NAMES

 SIGNAL DIRECTION_UP1_DOWN0 : std_logic := '0';          -- DIRECTION_UP1_DOWN0:          Determines which direction the extender should be going, 1 (UP) to 'extend' or increase state values (0000 -> 1111)
                                                         --                               and 0 (DOWN) 'retract' or to decrease state values (1111 -> 0000)


 -- Here the circuit begins

 BEGIN

 PROCESS (extender_toggle, current_state, extender_enabled, rst_n) IS
 BEGIN

   -- Checks:  if extender push button was pressed AND released
   --          if we are fully extended or fully retracted
   --          if RAC is not moving X-Y positions
   --          if reset (active low) is not pressed
   -- If so, we flip out current direction of movement

   IF (falling_edge(extender_toggle) AND (current_state = State_retracted or current_state = State_extended) AND (extender_enabled = '1') AND (rst_n = '1')) THEN
         DIRECTION_UP1_DOWN0 <= not (DIRECTION_UP1_DOWN0);
   END IF;

```

RAC Extender Design PART 2/3



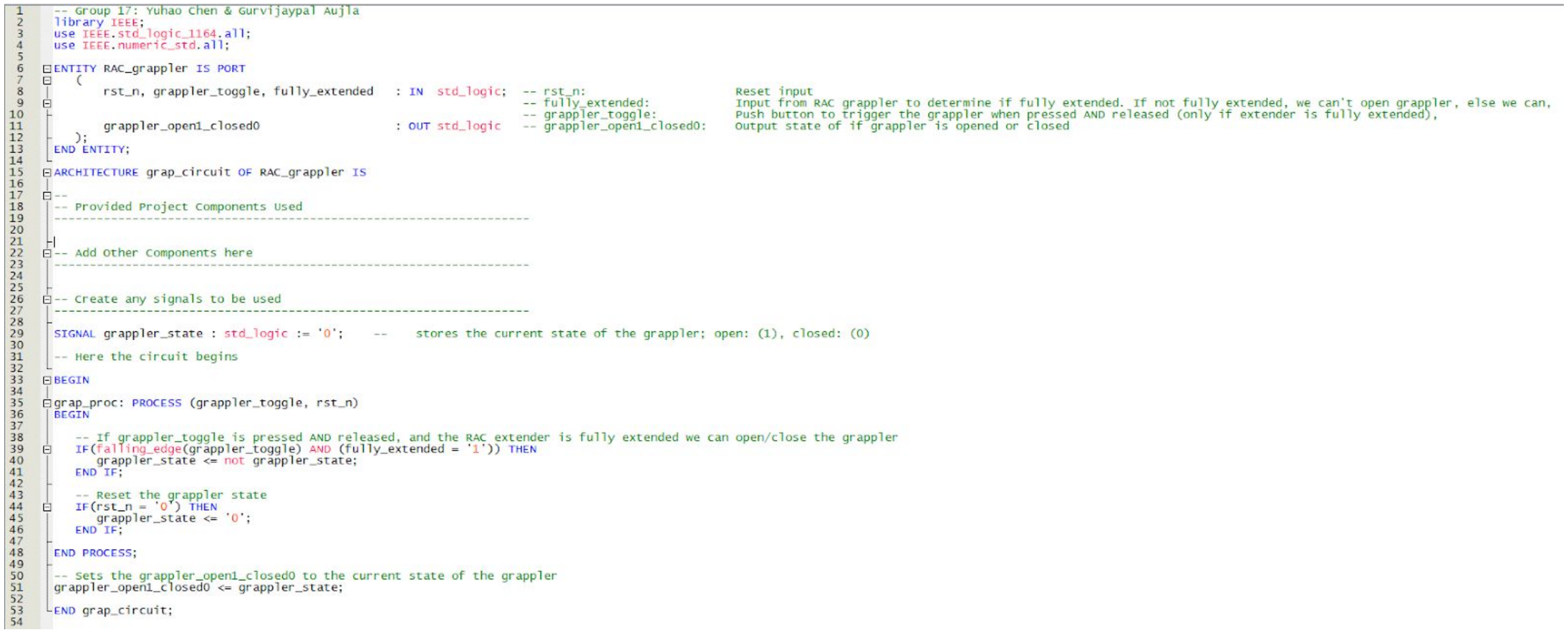
```vhdl

   -- Reset direction
   IF (rst_n = '0') THEN
         DIRECTION_UP1_DOWN0 <= '0';
   END IF;


END PROCESS;

 ---------------------------------------------------------------------------------
 --State Machine:
 ---------------------------------------------------------------------------------

 -- REGISTER_LOGIC PROCESS:

 -- This process synchronizes the activity to a clock
Register_Section: PROCESS (clk_input, rst_n)
BEGIN

   -- Reset to first state
   IF (rst_n = '0') THEN
      current_state <= State_retracted;

   -- Any time we reach the rising edge of the clock input, go to the next state
   ELSIF(rising_edge(clk_input)) THEN
      current_state <= next_state;
   END IF;

END PROCESS;



-- TRANSITION LOGIC PROCESS


-- Logic to determine if we should go from fully retracted to fully extended and back based on our desired current direction (DIRECTION_UP1_DOWN0)
Transition_Section: PROCESS (current_state, DIRECTION_UP1_DOWN0)

BEGIN
     CASE current_state IS
            WHEN State_retracted =>                               -- 0000
                  IF(DIRECTION_UP1_DOWN0 = '1') THEN
                     next_state <= State_1st_extend;
                  ELSE
                     next_state <= State_retracted;
                  END IF;

            WHEN State_1st_extend =>                              -- 1000
                  next_state <= State_2nd_extend;

            WHEN State_2nd_extend =>                              -- 1100
                  next_state <= State_3rd_extend;

            WHEN State_3rd_extend =>                              -- 1110
                  next_state <= State_extended;

            WHEN State_extended =>                                -- 1111
                  IF(DIRECTION_UP1_DOWN0 = '0') THEN
                     next_state <= State_1st_retract;
                  ELSE
                     next_state <= State_extended;
                  END IF;
```

RAC Movement Design PART 3/3



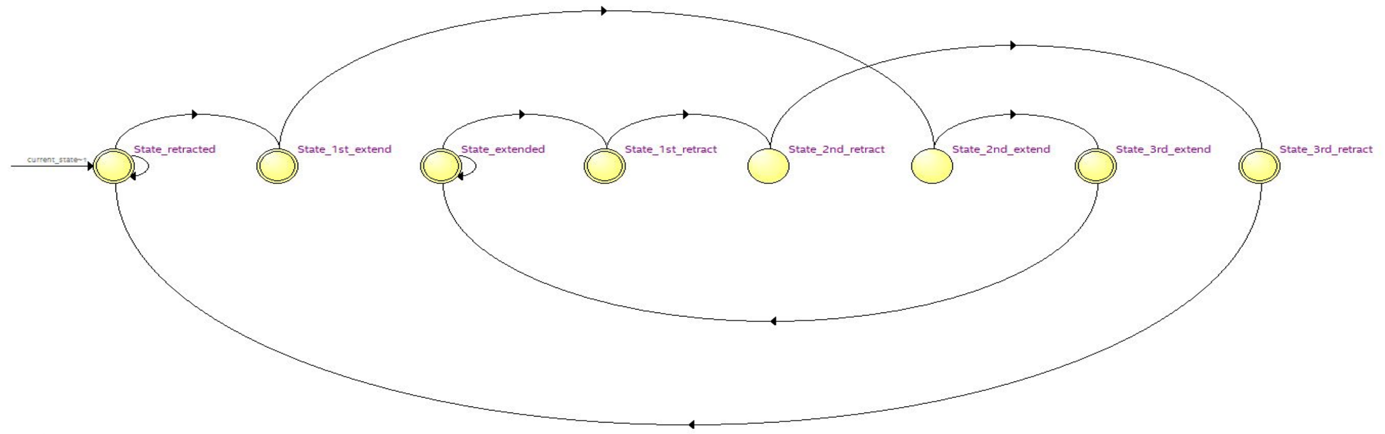
```vhdl

            WHEN State_1st_retract =>                           -- 1110
                  next_state <= State_2nd_retract;

            WHEN State_2nd_retract =>                           -- 1100
                  next_state <= State_3rd_retract;

            WHEN State_3rd_retract =>                           -- 1000
                  next_state <= State_retracted;
   END CASE;
 END PROCESS;

-- DECODER SECTION PROCESS (Moore Form)

-- Assigns values to the state output from our current extender position, extender output for if we are no longer retracted, and fully extended if we are fully extended
-- Does so for each of the 8 possible states
Decoder_Section: PROCESS (current_state)

BEGIN
     CASE current_state IS
         WHEN State_retracted =>
            state_output <= "0000";
            extender_output <= '0';
            fully_extended <= '0';

         WHEN State_1st_extend =>
            state_output <= "1000";
            extender_output <= '1';
            fully_extended <= '0';

         WHEN State_2nd_extend=>
            state_output <= "1100";
            extender_output <= '1';
            fully_extended <= '0';

         WHEN State_3rd_extend=>
            state_output <= "1110";
            extender_output <= '1';
            fully_extended <= '0';

         WHEN State_extended =>
            state_output <= "1111";
            extender_output <= '1';
            fully_extended <= '1';

         WHEN State_1st_retract =>
            state_output <= "1110";
            extender_output <= '1';
            fully_extended <= '0';

         WHEN State_2nd_retract =>
            state_output <= "1100";
            extender_output <= '1';
            fully_extended <= '0';

         WHEN State_3rd_retract =>
            state_output <= "1000";
            extender_output <= '1';
            fully_extended <= '0';
  END CASE;
 END PROCESS;

 END ARCHITECTURE ext_circuit;
```

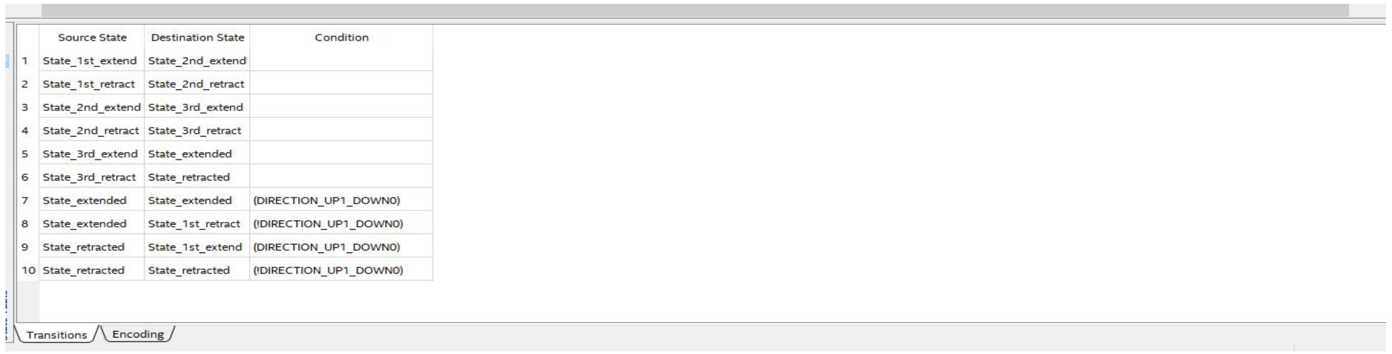
RAC Grappler Design



```vhdl
-- Group 17: Yuhao Chen & Gurvijaypal Aujla
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

ENTITY RAC_grappler IS PORT
   (
       rst_n, grappler_toggle, fully_extended   : IN  std_logic;  -- rst_n:                   Reset input
                                                                  -- fully_extended:          Input from RAC grappler to determine if fully extended. If not fully extended, we can't open grappler, else we can,
                                                                  -- grappler_toggle:         Push button to trigger the grappler when pressed AND released (only if extender is fully extended),
       grappler_open1_closed0                   : OUT std_logic   -- grappler_open1_closed0:  Output state of if grappler is opened or closed
   );
END ENTITY;

ARCHITECTURE grap_circuit OF RAC_grappler IS

--
-- Provided Project Components Used
--------------------------------------------------------------------


-- Add Other Components here
--------------------------------------------------------------------


-- Create any signals to be used
--------------------------------------------------------------------

SIGNAL grappler_state : std_logic := '0';     --    stores the current state of the grappler; open: (1), closed: (0)

-- Here the circuit begins

BEGIN

grap_proc: PROCESS (grappler_toggle, rst_n)
BEGIN

   -- If grappler_toggle is pressed AND released, and the RAC extender is fully extended we can open/close the grappler
   IF(falling_edge(grappler_toggle) AND (fully_extended = '1')) THEN
      grappler_state <= not grappler_state;
   END IF;

   -- Reset the grappler state
   IF(rst_n = '0') THEN
      grappler_state <= '0';
   END IF;

END PROCESS;

-- Sets the grappler_open1_closed0 to the current state of the grappler
grappler_open1_closed0 <= grappler_state;

END grap_circuit;
```

RAC Extender State Diagram for State Machine

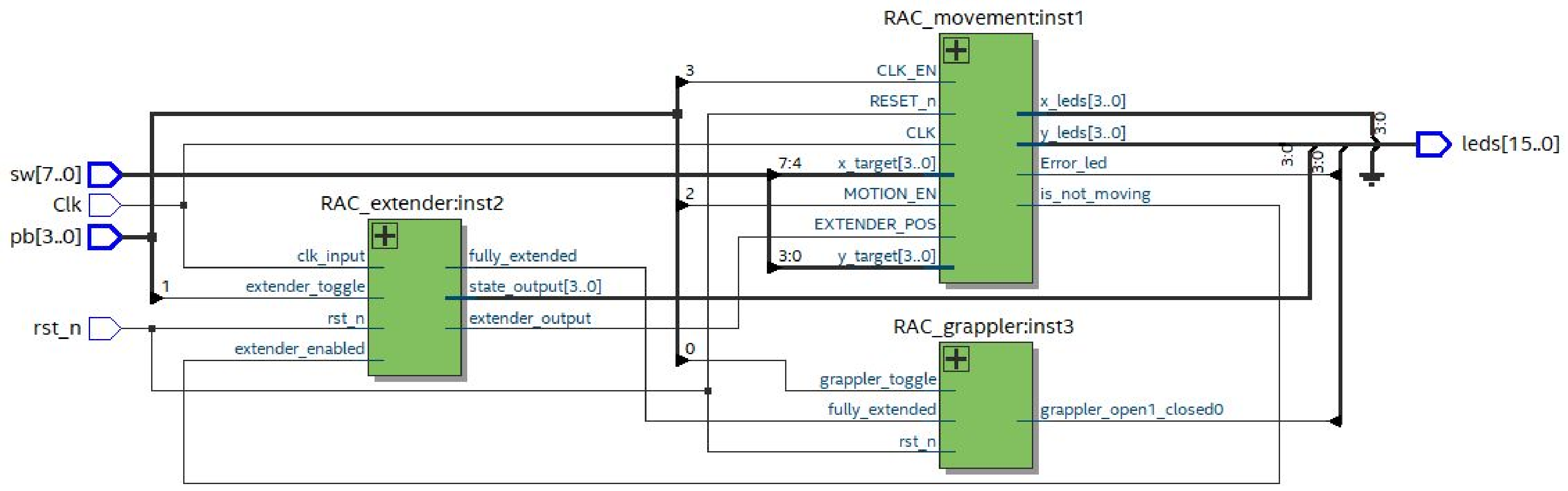




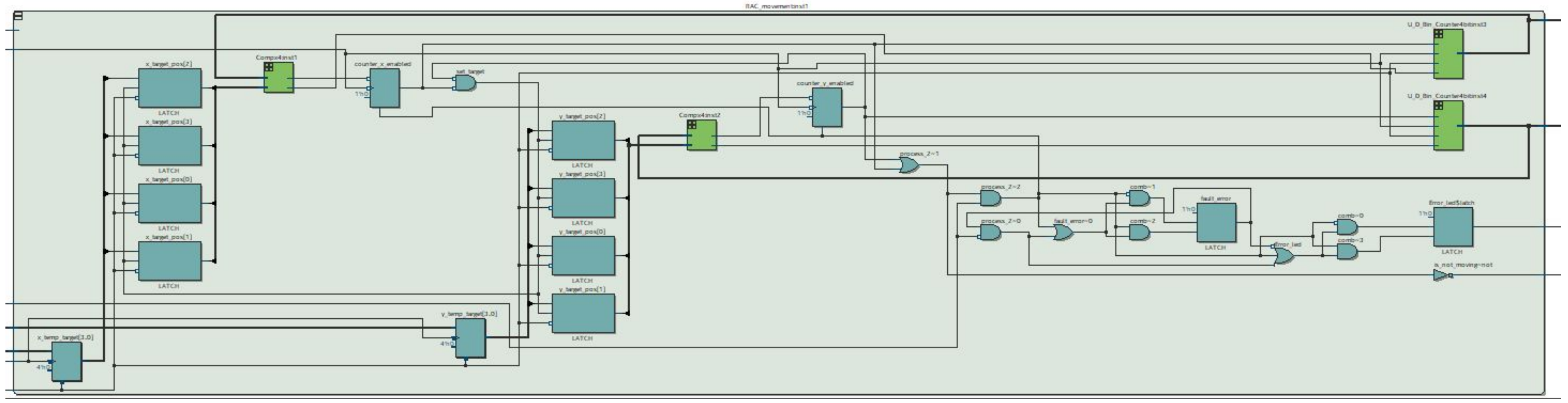
	Source State	Destination State	Condition
1	State_1st_extend	State_2nd_extend	
2	State_1st_retract	State_2nd_retract	
3	State_2nd_extend	State_3rd_extend	
4	State_2nd_retract	State_3rd_retract	
5	State_3rd_extend	State_extended	
6	State_3rd_retract	State_retracted	
7	State_extended	State_extended	(DIRECTION_UP1_DOWN0)
8	State_extended	State_1st_retract	(!DIRECTION_UP1_DOWN0)
9	State_retracted	State_1st_extend	(DIRECTION_UP1_DOWN0)
10	State_retracted	State_retracted	(!DIRECTION_UP1_DOWN0)

When in state_retracted, we continue to loop in to this state until the condition of DIRECTION_UP1_DOWN0 changes from 0 to 1. After this we change to state_1st_extended which has an output of (1000), this state has no loop and will directly take us to state_2nd_extend. This state has an output of (1100), and has no loop and will directly take us to state state_3rd_extend. This state has an output of (1110), and has no loop and will directly take us to state state_extended where the extender is fully extended. Here we will continue to loop in to ourself until the condition DIRECTION_UP1_DOWN0 changes from 1 to 0 and will take us back to state_retracted via the following states. We will first go to state_1st_retract which has a value of (1110) and no loop so this will directly take us to state_2nd_retract. Here we have a value of (1100) and no loop so this will directly take us to state_3rd_retract. Here we have a value of (1000) and no loop so this will directly take us to state_retracted. And thus we are back to the beginning.

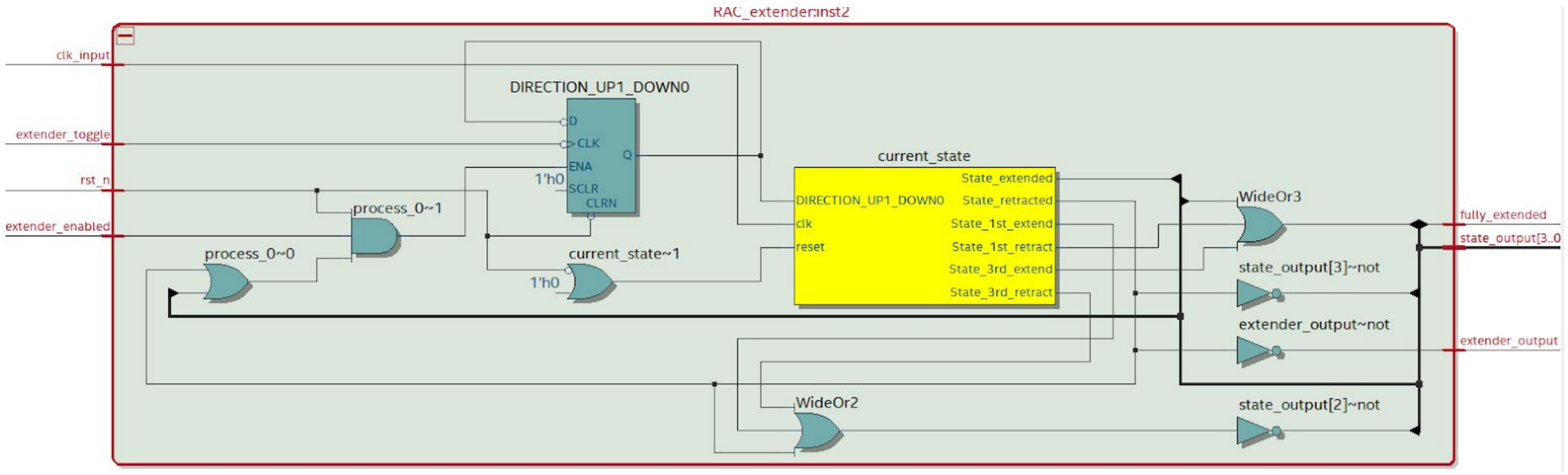
RAC Block Diagram for Top Level



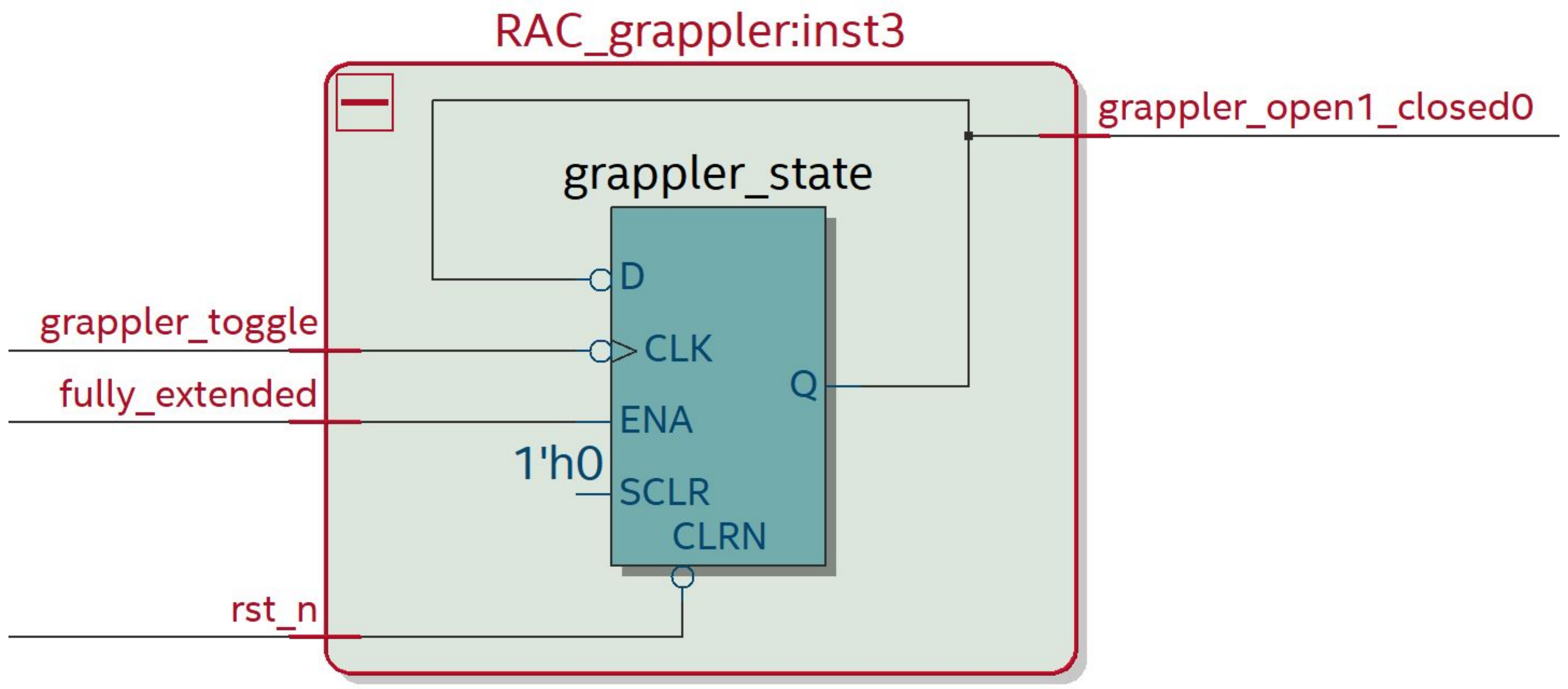
RAC Block Diagram for Movement Component



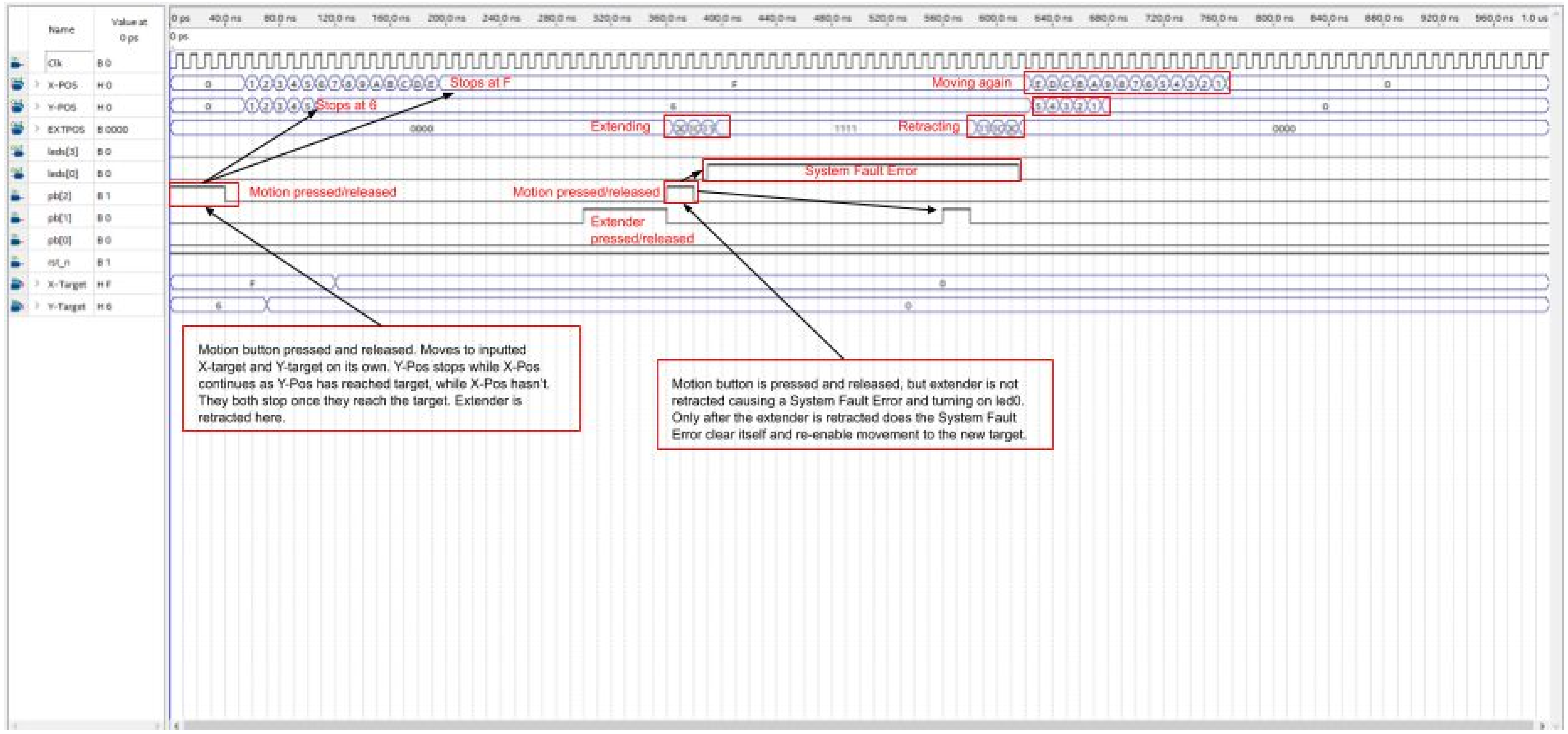
RAC Block Diagram for Extender Component



RAC Block Diagram for Grappler



Functional Simulation of X/Y Transport Operations

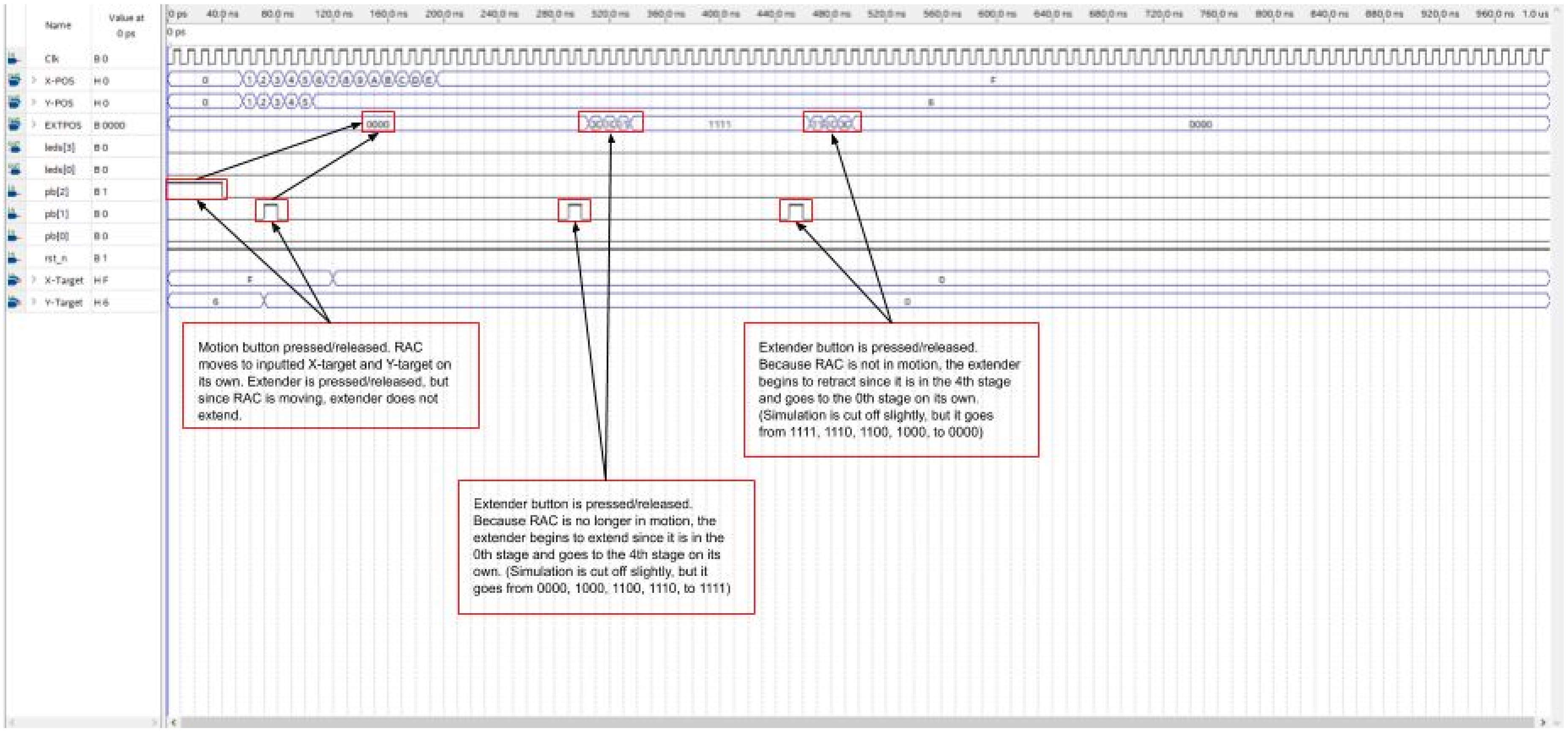


Motion button pressed and released. Moves to inputted X-target and Y-target on its own. Y-Pos stops while X-Pos continues as Y-Pos has reached target, while X-Pos hasn't. They both stop once they reach the target. Extender is retracted here.

Motion button is pressed and released, but extender is not retracted causing a System Fault Error and turning on led0. Only after the extender is retracted does the System Fault Error clear itself and re-enable movement to the new target.

Link if blurry

Functional Simulation of Extender Operations



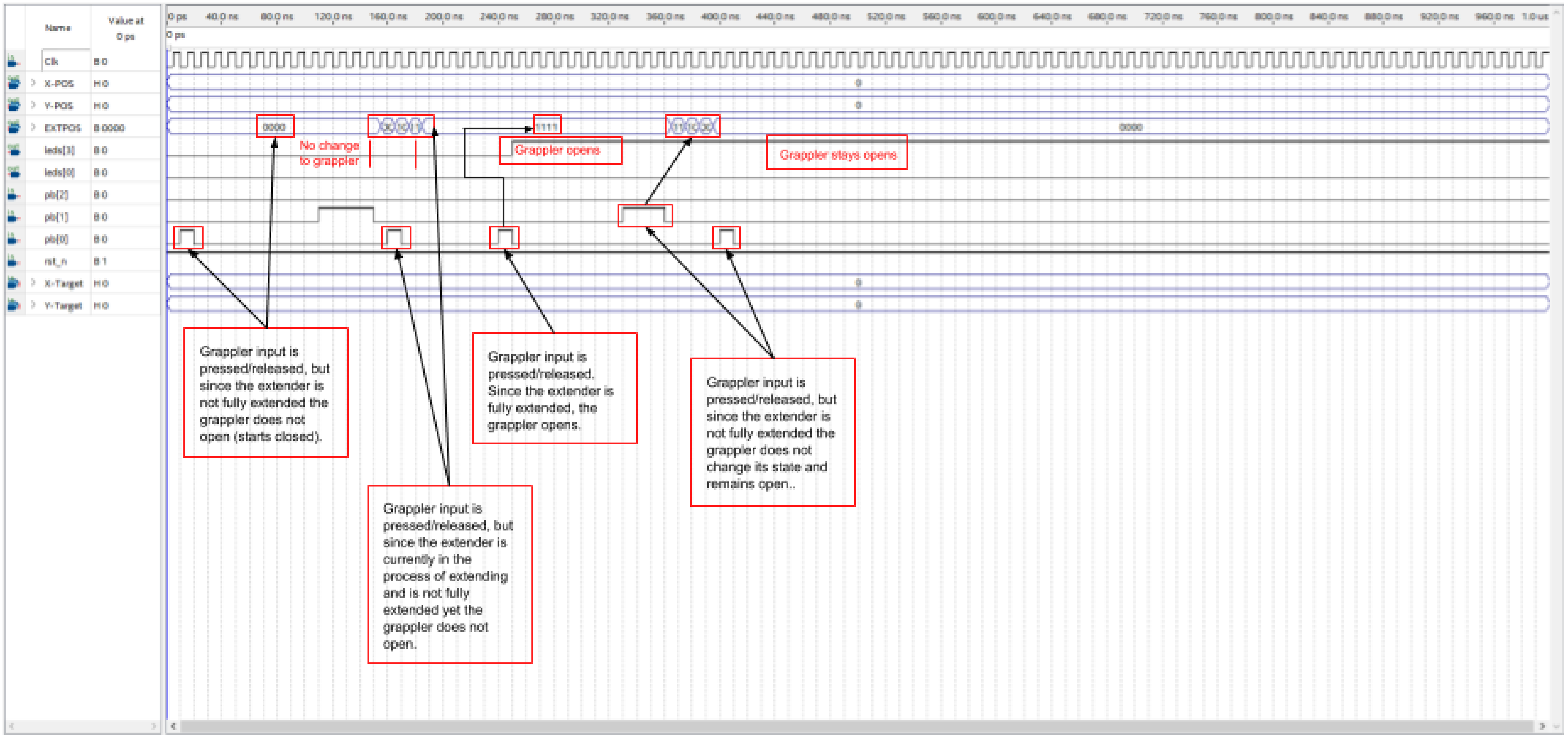
Motion button pressed/released. RAC moves to inputted X-target and Y-target on its own. Extender is pressed/released, but since RAC is moving, extender does not extend.

Extender button is pressed/released. Because RAC is no longer in motion, the extender begins to extend since it is in the 0th stage and goes to the 4th stage on its own. (Simulation is cut off slightly, but it goes from 0000, 1000, 1100, 1110, to 1111)

Extender button is pressed/released. Because RAC is not in motion, the extender begins to retract since it is in the 4th stage and goes to the 0th stage on its own. (Simulation is cut off slightly, but it goes from 1111, 1110, 1100, 1000, to 0000)

Link if blurry

Functional Simulation of Grappler Operations



Grappler input is pressed/released, but since the extender is not fully extended the grappler does not open (starts closed).

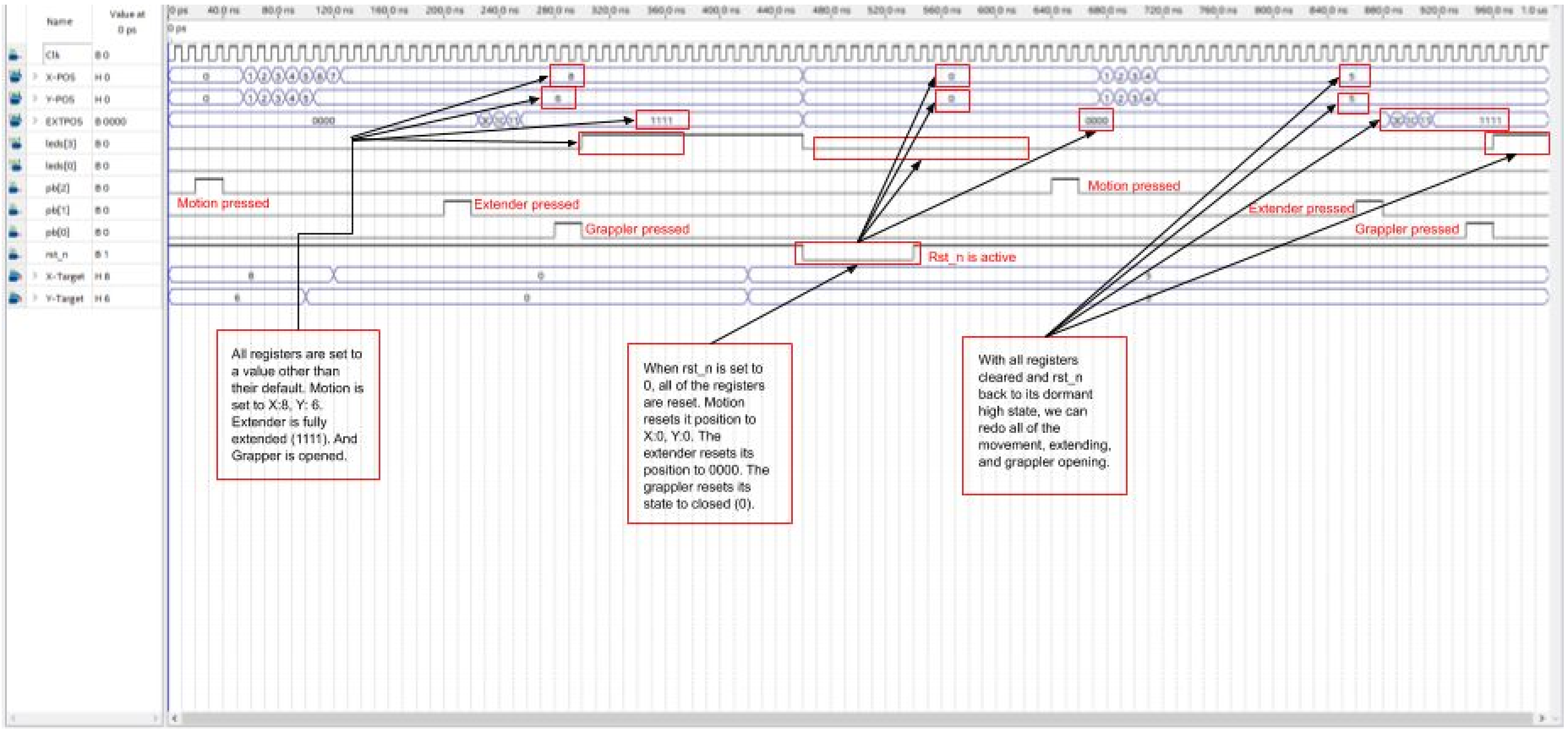
Grappler input is pressed/released, but since the extender is currently in the process of extending and is not fully extended yet the grappler does not open.

Grappler input is pressed/released. Since the extender is fully extended, the grappler opens.

Grappler input is pressed/released, but since the extender is not fully extended the grappler does not change its state and remains open..

Link if blurry

Functional Simulation of Registers Being Reset When rst_n is Active



All registers are set to a value other than their default. Motion is set to X:8, Y: 6. Extender is fully extended (1111). And Grapper is opened.

When rst_n is set to 0, all of the registers are reset. Motion resets it position to X:0, Y:0. The extender resets its position to 0000. The grappler resets its state to closed (0).

With all registers cleared and rst_n back to its dormant high state, we can redo all of the movement, extending, and grappler opening.

Link if blurry