ECE3561 Project #3

Designing an elevator

Group 39
Wafeeq Jaleel
Tianxiang He

The Ohio State University
April 25, 2022

Introduction

The team has been assigned to design an elevator system for a four-story building using VHDL. The system can be divided into two units: controller and simulator, with the former one controlling the operations and the latter one imitating the movement of an actual elevator. Figure 1 exhibits the overall elevator structure.

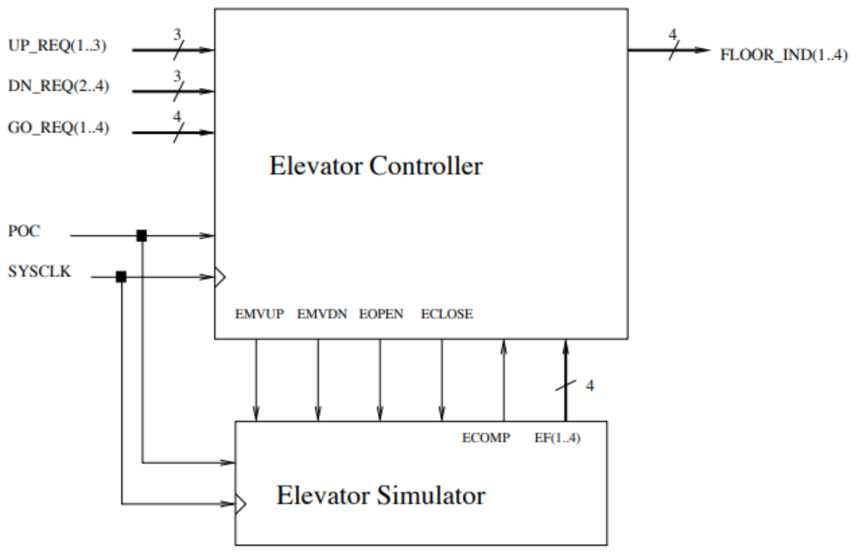


Figure 1: Overall Elevator design

For the controller, UP_REQ(i) is asserted when the up request on the ith floor (except the top floor) has been pushed, and DN_REQ(i) is asserted when the down request on the ith floor (except the bottom floor) has been pushed. GO_REQ(1..4) detects if any floor button inside the elevator has been pushed and assets the corresponding bit. FLOOR_IND(1..4) signal asserts the bit corresponding to the current floor and negates all other bits. When the reseting signal POC is not asserted and the clock signal SYSCLK is at rising edge, the controller detects any forms of request and moves to serve the request while setting the elevator’s direction to either up or down. Once the direction is set, the elevator will serve all requests along its route and it will only start serving requests that require the change of direction after all requests along the current direction have been served. The controller asserts EMVUP or EMVDN signal and send them to the simulator when the elevator is serving an UP_REQ or DN_REQ. When the elevator door needs to be open or closed, EOPEN or ECLOSE would be asserted. OPEN_REQ and CLOSE_REQ are two extra buttons that make the elevator more accessible. When the user presses the open button inside the elevator car, OPEN_REQ would be asserted, which opens the door before it is completely closed. The CLOSE_REQ allows users to close the elevator door when a person enters the elevator car.

The simulator is a synchronous system that shares the same clock signal as the controller. It updates the current floor and sends floor information to the controller through EF(1..4). ECOMP would only be asserted when there is no ongoing operation. When EMVUP is asserted, the simulator negates ECOMP immediately and increases the current floor by one after a two seconds delay, then asserts ECOMP again indicating the current operation has been finished. Same thing happens when EMVDN is asserted except the simulator decreases the floor information by one instead of increasing. When either EOPEN or ECLOSE is asserted, the simulator negates ECOMP and asserts it again after a three seconds delay, which imitates the time used for opening or closing the elevator door.

Design

The design process is divided into four parts, which are elevator, controller, simulator, and wave generator respectively. All the components were coded using VHDL in XILINX software.

Elevator

The overall elevator design can be seen in Figure 1. It has the input signals UP_REQ, DN_REQ, GO_REQ, POC, and SYSCLK and output signal FLOOR_IND. For design ease the elevator was divided into controller and simulator. The details of each signal will be explained in depth in the controller and simulator subsections. The function of Elevator VHDL file was to connect the subcomponents controller and simulator's internal signals. The internal signals are EMVIP, EMVDN, EOPEN, ECLOSE, ECOMP, and EF.

In the Elevator VHDL file the team declared all the external and internal signals as seen in Appendix A Figure A1 and A2. Next the signals from controller and simulator were mapped to their corresponding signals from the elevator which is in Figure A2. The SYSCLK was a 2Hz clock signal.

Controller

The most significant function of the controller is to identify requests from upper floors, lower floors, and current floor respectively. Twelve internal signals are initialized inside the behavioral model of the controller for achieving this goal. FLR_BLW(1..4) and FLR_ABV(1..4) exhibit the floor information below and above the current floor. By combining UP_REQ, DN_REQ, and GO_REQ using "OR" logic all requests at any floor are shown in the four bits array ALL_REQ(1..4), which makes it possible for the team to detect if there is any request from floors above or below. For example, if the elevator is at the second floor and there is an UP_REQ at the third floor and a DN_REQ at the second floor, ALL_REQ, FLR_ABV, and FLR_BLW would be equal to "0110" "1100" "0001" respectively. By comparing ALL_REQ with FLR_BLW and FLR_ABV separately using "AND" logic, the team gets ABV_REQ equals '1' and BLW_REQ equals '0', which means there exists a request somewhere above and no request below.

The team also created Current_UP_REQ, Current_DN_REQ, and Current_GO_REQ to detect if there is any request at the current floor that would open the elevator door. All three signals are concatenated using "OR" logic in the signal "DOOR, " which returns one if the door should be opened. IS_DOOR_OPEN, UP, and DN check the state of the door and the elevator's direction respectively. These can be seen in Appendix B Figure B1 and B2.

As seen in Figure B2 and B3 of Appendix B, when SYSCLK is at the rising edge, all internal outputs to the simulator are initialized to zero. The controller then checks if the reset signal POC is equal to one, otherwise it's going to check whether the door is open using a series of if - else statements. If the door is open when it should be closed, the controller asserts ECLOSE, telling the simulator to close the door. If the door is closed already, the controller goes through four if - else cases to determine the next operation and sends all commands to the simulator.

For the accessibility part, the team added open & close buttons to the original design. The CLOSE_REQ allows users to close the elevator door when a person enters the elevator car, and the OPEN_REQ opens the door before it is completely closed while the elevator is stationary.

Simulator
Since the team did not have a real elevator to control, the team designed a simulator replicating the elevator's expected functions. The assumptions were that it is a synchronous system sharing clock signal and POC with the controller. The VHDL code for this section is in Appendix C.

The simulator had four inputs fed from the controller and two outputs to the controller. EMPVUP and EMVDN indicated the direction of elevator movement. ECLOSE and EOPEN showed whether to keep the door open or closed. While the output ECOMP asserted means the elevator is in an ongoing operation. The EF(1…4) is a bus output that expresses the current floor to the controller. The team had to implement an internal signal named COUNTER for pauses when one of the input signals were asserted. Also, another internal signal named FLOOR which is an unsigned integer for the floor number was used to easily increment and decrement the floor levels. The signal declarations can be seen in Figure C1.

The design process was to use IF-ELSE statements to see if the floor has to go up or down based on the EMVUP and EMVDN inputs to increase floor level and halt the operations for two seconds. For EOPEN and ECLOSE the operation was halted for three seconds. If the POC is asserted the elevator is designed to return to the first floor and halt all ongoing operations. The VHDL code is in Figure C2.

Wave generator
The team created a VHDL test bench to run the simulations. The units under testing were the controller and the simulator. All the external and internal signals were included for easy debugging. Since the clock signal was 2Hz, the SYSCLK was set to cycle every 500ms. Initially all the signals were initialized to 0 and POC to 1. All the declared signals can be seen in APPENDIX D Figure D1, D2, and D3.

Once one clock cycle passed, the POC was negated and the stimulus process began. The team tested the external inputs UP_REQ, DN_REQ, GO_REQ, CLOSE_REQ, and OPEN_REQ. All the testing commands are shown in Figure D4 of APPENDIX D. Once each request is done, the team has to manually negate the request so others can proceed. The simulation software used was ISIM which has a simulation time of 30 seconds.

Results

The results in waveform are represented below in Figure 2. The wave includes all the external and internal signals.

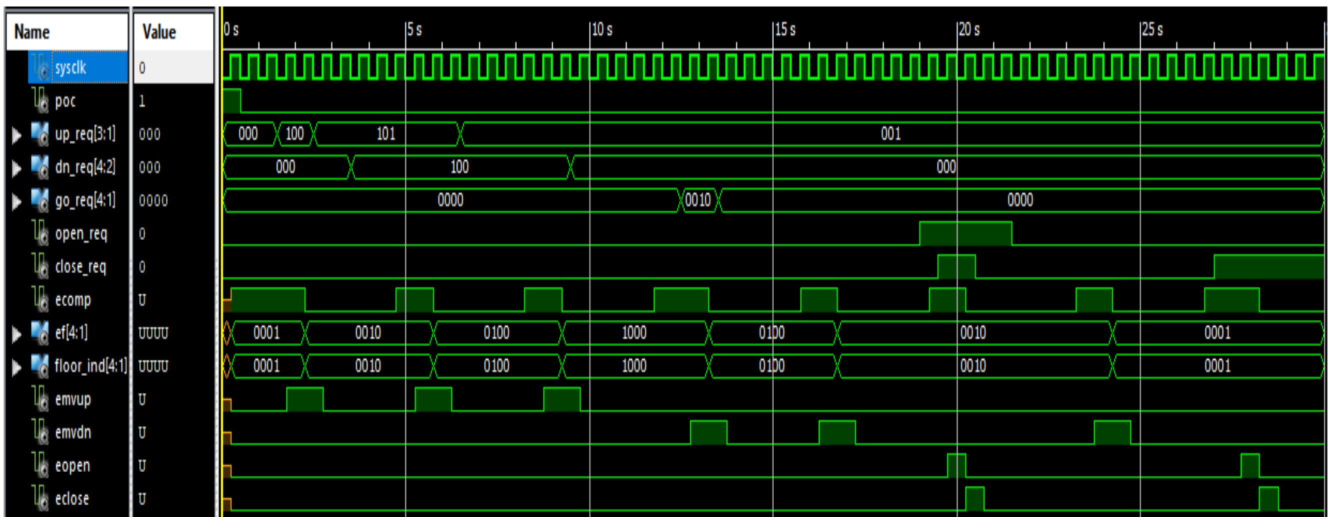


Figure 2: Simulated wave from ISIM

Since it was harder to interpret the data from Figure 2, the team converted it to Table 1. The Table points out the important time frame when requests are made or a change in output signals is observed. Some test times are highlighted in yellow for discussions in the Comparison section.

Table 1: Important time frames from the wave

Test (sec)	**up_ req**	**dn_req**	**go_ req**	**open- req**	**close_ req**	**floor_ ind**	**ecomp**	**emvup**	**emvdn**	**eopen**	**eclose**
1.5	100	000	0000	0	0	0001	1	0	0	0	0
1.75	100	000	0000	0	0	0001	1	1	0	0	0
2.25	100	000	0000	0	0	0010	0	1	0	0	0
2.50	101	000	0000	0	0	0010	0	1	0	0	0
3.50	101	100	0000	0	0	0010	0	0	0	0	0

4.75	101	100	0000	0	0	0010	1	0	0	0	0
5.25	101	100	0000	0	0	0010	1	1	0	0	0
5.75	101	100	0000	0	0	0100	0	1	0	0	0
6.5	001	100	0000	0	0	0100	0	0	0	0	0
8.25	001	100	0000	0	0	0100	1	0	0	0	0
8.75	001	100	0000	0	0	0100	1	1	0	0	0
9.25	001	100	0000	0	0	1000	0	1	0	0	0
9.50	001	000	0000	0	0	1000	0	1	0	0	0
11.75	001	000	0000	0	0	1000	1	0	0	0	0
12.50	001	000	0010	0	0	1000	1	0	0	0	0
12.75	001	000	0010	0	0	1000	1	0	1	0	0
13.25	001	000	0010	0	0	0100	0	0	1	0	0
13.5	001	000	0000	0	0	0100	0	0	1	0	0
15.75	001	000	0000	0	0	0100	1	0	0	0	0
16.25	001	000	0000	0	0	0100	1	0	1	0	0
16.75	001	000	0000	0	0	0010	0	0	1	0	0
19.0	001	000	0000	1	0	0010	0	0	0	0	0
19.5	001	000	0000	1	1	0010	1	0	0	0	0
19.75	001	000	0000	1	1	0010	1	0	0	1	0
20.25	001	000	0000	1	1	0010	0	0	0	0	1
23.25	001	000	0000	0	0	0010	1	0	0	0	0
23.75	001	000	0000	0	0	0010	1	0	1	0	0
24.25	001	000	0000	0	0	0001	0	0	1	0	0
26.75	001	000	0000	0	0	0001	1	0	0	0	0
27.0	001	000	0000	0	1	0001	1	0	0	0	0
27.75	001	000	0000	0	1	0001	1	0	0	1	0
28.25	001	000	0000	0	1	0001	0	0	0	0	1

Comparisons

The team's expectation was to implement the SCAN algorithm. Some of the basic requirements were if there are no requests, the elevator will be idle on the current floor. Once a direction has been set to the elevator, all the requests have to be completed before proceeding to the opposite direction requests. The elevator moved from floor one to floor four before returning to floor one to get the UP_REQ. This can be seen in the wave from Figure 2 or Table 1. The requests were given at seconds 1.5, 2.5 for up requests, 3.5 for down request, and 12.5 for go request.

Some internal signals that worked as expected were the ECOMP signal, which was asserted after two seconds for the EMVUP and EMVDN negation and after three seconds for EOPEN and ECLOSE negation.

The OPEN_REQ and CLOSE_REQ for the design for accessibility were tested and verified that they function as expected from the results in between 19.0sec to 20.25sec in Table 1. The EOPEN signal was asserted 0.75 sec after OPEN_REQ was asserted and ECLOSE was asserted after one clock signal in response to the CLOSE_REQ. Additionally, the CLOSE_REQ was tested for door opening automatically after reaching the expected floor. This test was between 27.0 and 28.25 seconds in Table 1. The EOPEN remained asserted until the CLOSE_REQ forced ECLOSE to be one and EOPEN to be zero.

However, the team were unable to give multiple requests within one second interval. When the requests are delayed for one second, the elevator only completed the last request. For example, when UP_REQ(3), UP_REQ(1), DN_REQ(4), and GO_REQ(2) were asserted the elevator would stop at the second floor and not proceed to floor three or four. The reason for this problem was the propagation delays that made the next request proceed before the current request was finished. The solution the team came up with was to manually negate the signals in the wave generator VHDL file. This can be seen at 6.5, 9.5, and 13.5 seconds in Table 1.

One of the functions of the elevator was to open the elevator once it reaches the requested floor. But the team was unable to produce this function of asserting EOPEN, mainly due to the propagation delay that always kept the elevator moving. The EOPEN was asserted at the last floor request at the end of the simulation at 27.75 second. So the solution would be to increase the simulation time and space out the delays in between requests to allow the EOPEN to function properly.

Conclusions

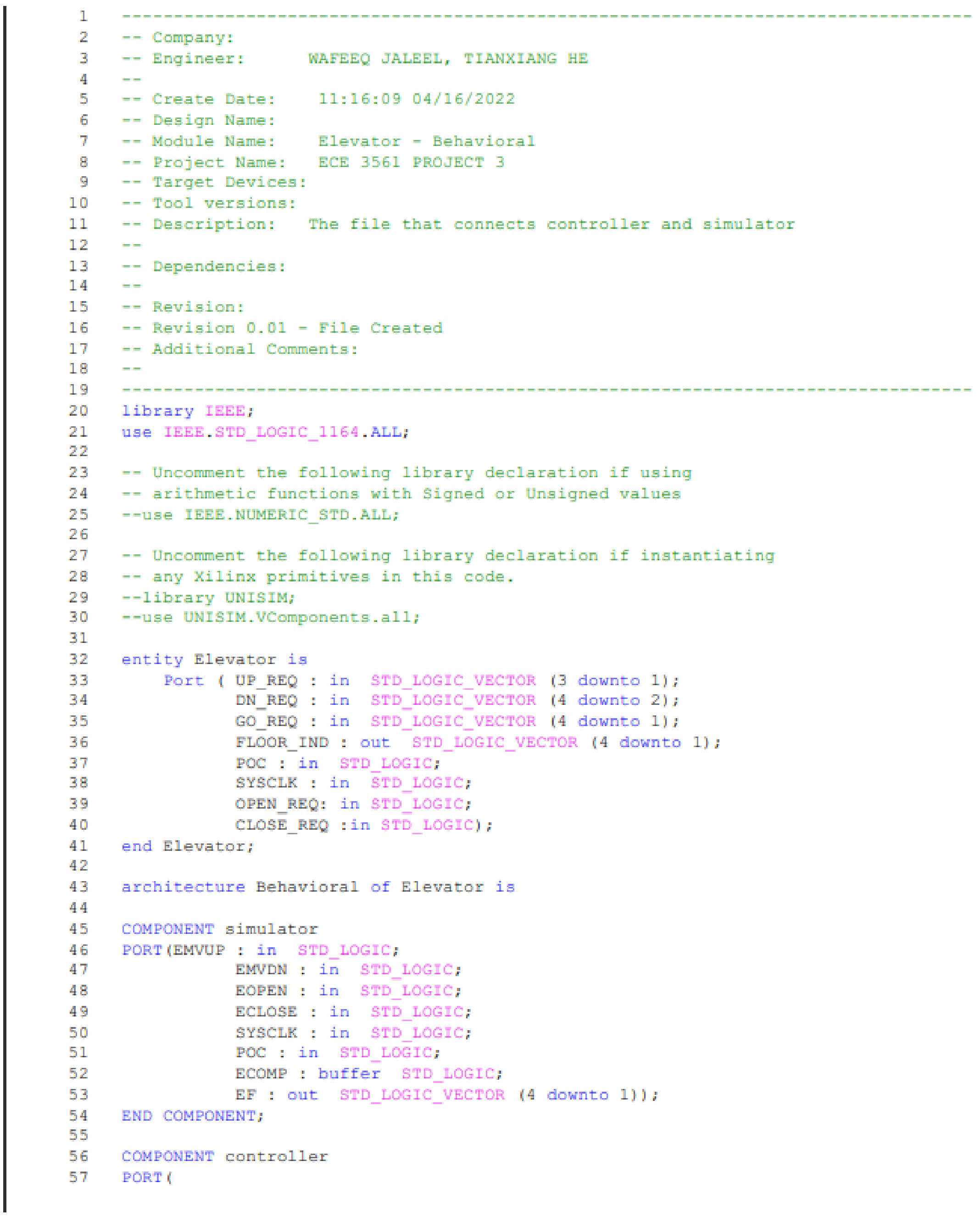
The team was given to design an elevator that had two components, the controller and simulator. Throughout this process, the team learned how to connect two subcomponents under one entity using VHDL. While simulating the top entity "elevator" using VHDL Testbench, the team traced the values of external and internal signals and figured out why the system was generating incorrect results by analyzing source codes line by line. After rectifying the structure of the simulator code several times, the team finally obtained expected simulation results for most of the test cases, which enhanced the team's ability of analyzing and debugging complicated VHDL projects significantly.

The biggest problem the team encountered happened in the initial stages of simulating. The ISIM software produced a blank page even though the code compiled properly. Same error happened multiple times on different computers. Since there was no way to debug the error, the team created a new project and pasted the code, which solved the problem.

Some of the constraints while using VHDL were that there was not a built-in delay function. Moreover, since the VHDL Testbench assumes ideal timing and does not reflect the realities of finite propagation delays and setup/hold times, the simulation results might be divergent from actual elevator operation, which could lead to systematic errors in actual operation.

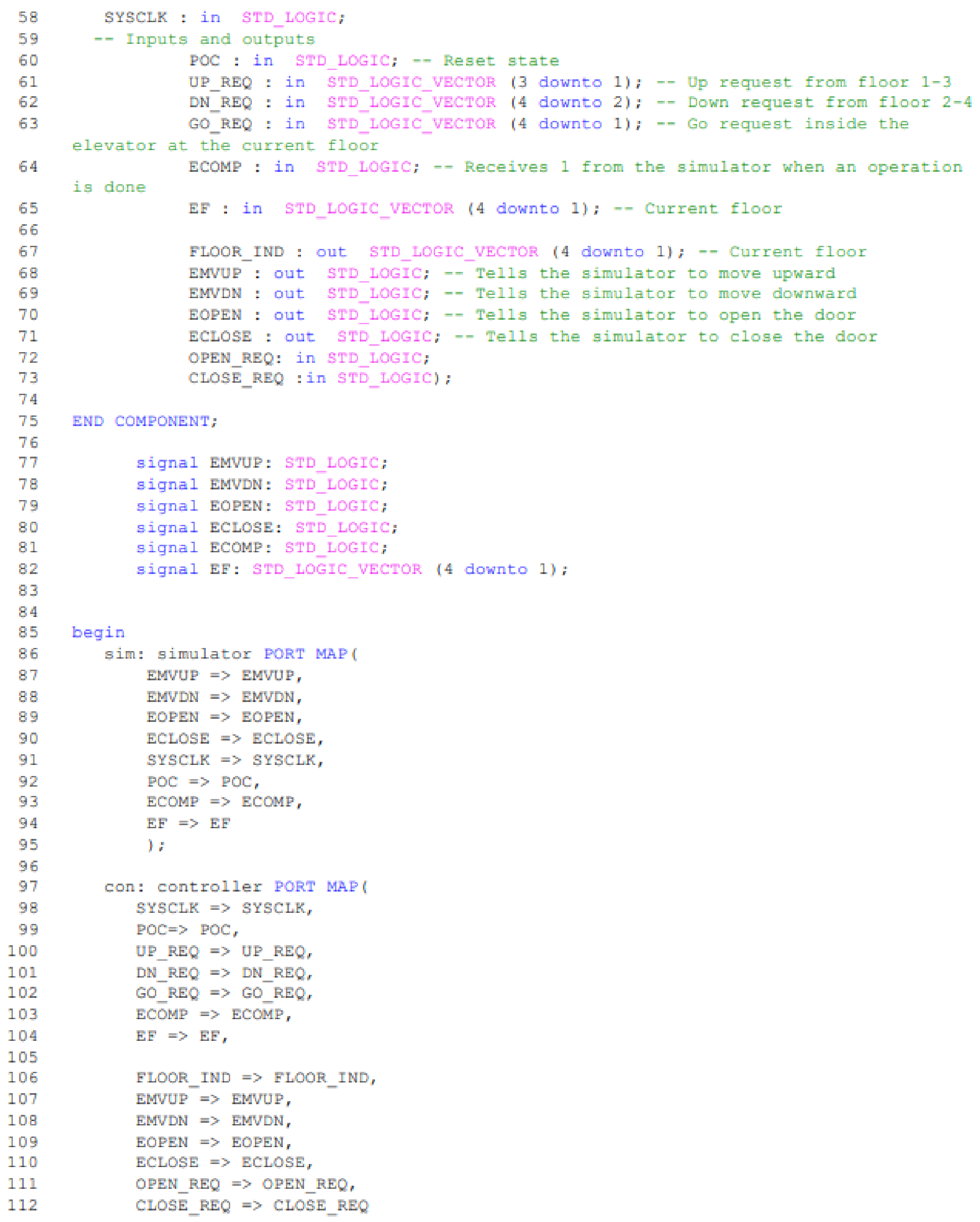
APPENDIX

APPENDIX A: Elevator VHDL codes



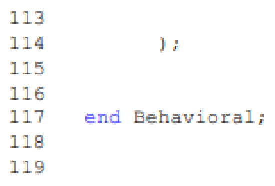
```vhdl
----------------------------------------------------------------------------------
-- Company:
-- Engineer:      WAFEEQ JALEEL, TIANXIANG HE
--
-- Create Date:    11:16:09 04/16/2022
-- Design Name:
-- Module Name:    Elevator - Behavioral
-- Project Name:   ECE 3561 PROJECT 3
-- Target Devices:
-- Tool versions:
-- Description:   The file that connects controller and simulator
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Elevator is
    Port ( UP_REQ : in  STD_LOGIC_VECTOR (3 downto 1);
           DN_REQ : in  STD_LOGIC_VECTOR (4 downto 2);
           GO_REQ : in  STD_LOGIC_VECTOR (4 downto 1);
           FLOOR_IND : out  STD_LOGIC_VECTOR (4 downto 1);
           POC : in  STD_LOGIC;
           SYSCLK : in  STD_LOGIC;
           OPEN_REQ: in STD_LOGIC;
           CLOSE_REQ :in STD_LOGIC);
end Elevator;

architecture Behavioral of Elevator is

COMPONENT simulator
PORT(EMVUP : in  STD_LOGIC;
           EMVDN : in  STD_LOGIC;
           EOPEN : in  STD_LOGIC;
           ECLOSE : in  STD_LOGIC;
           SYSCLK : in  STD_LOGIC;
           POC : in  STD_LOGIC;
           ECOMP : buffer  STD_LOGIC;
           EF : out  STD_LOGIC_VECTOR (4 downto 1));
END COMPONENT;

COMPONENT controller
PORT(
```

Figure A1: Elevator VHDL signal declaration of the components



```
    SYSCLK : in  STD_LOGIC;
   -- Inputs and outputs
            POC : in  STD_LOGIC; -- Reset state
            UP_REQ : in  STD_LOGIC_VECTOR (3 downto 1); -- Up request from floor 1-3
            DN_REQ : in  STD_LOGIC_VECTOR (4 downto 2); -- Down request from floor 2-4
            GO_REQ : in  STD_LOGIC_VECTOR (4 downto 1); -- Go request inside the elevator at the current floor
            ECOMP : in  STD_LOGIC; -- Receives 1 from the simulator when an operation is done
            EF : in  STD_LOGIC_VECTOR (4 downto 1); -- Current floor

            FLOOR_IND : out  STD_LOGIC_VECTOR (4 downto 1); -- Current floor
            EMVUP : out  STD_LOGIC; -- Tells the simulator to move upward
            EMVDN : out  STD_LOGIC; -- Tells the simulator to move downward
            EOPEN : out  STD_LOGIC; -- Tells the simulator to open the door
            ECLOSE : out  STD_LOGIC; -- Tells the simulator to close the door
            OPEN_REQ: in STD_LOGIC;
            CLOSE_REQ :in STD_LOGIC);

END COMPONENT;

      signal EMVUP: STD_LOGIC;
      signal EMVDN: STD_LOGIC;
      signal EOPEN: STD_LOGIC;
      signal ECLOSE: STD_LOGIC;
      signal ECOMP: STD_LOGIC;
      signal EF: STD_LOGIC_VECTOR (4 downto 1);


begin
   sim: simulator PORT MAP(
       EMVUP => EMVUP,
       EMVDN => EMVDN,
       EOPEN => EOPEN,
       ECLOSE => ECLOSE,
       SYSCLK => SYSCLK,
       POC => POC,
       ECOMP => ECOMP,
       EF => EF
       );

   con: controller PORT MAP(
      SYSCLK => SYSCLK,
      POC=> POC,
      UP_REQ => UP_REQ,
      DN_REQ => DN_REQ,
      GO_REQ => GO_REQ,
      ECOMP => ECOMP,
      EF => EF,

      FLOOR_IND => FLOOR_IND,
      EMVUP => EMVUP,
      EMVDN => EMVDN,
      EOPEN => EOPEN,
      ECLOSE => ECLOSE,
      OPEN_REQ => OPEN_REQ,
      CLOSE_REQ => CLOSE_REQ
```

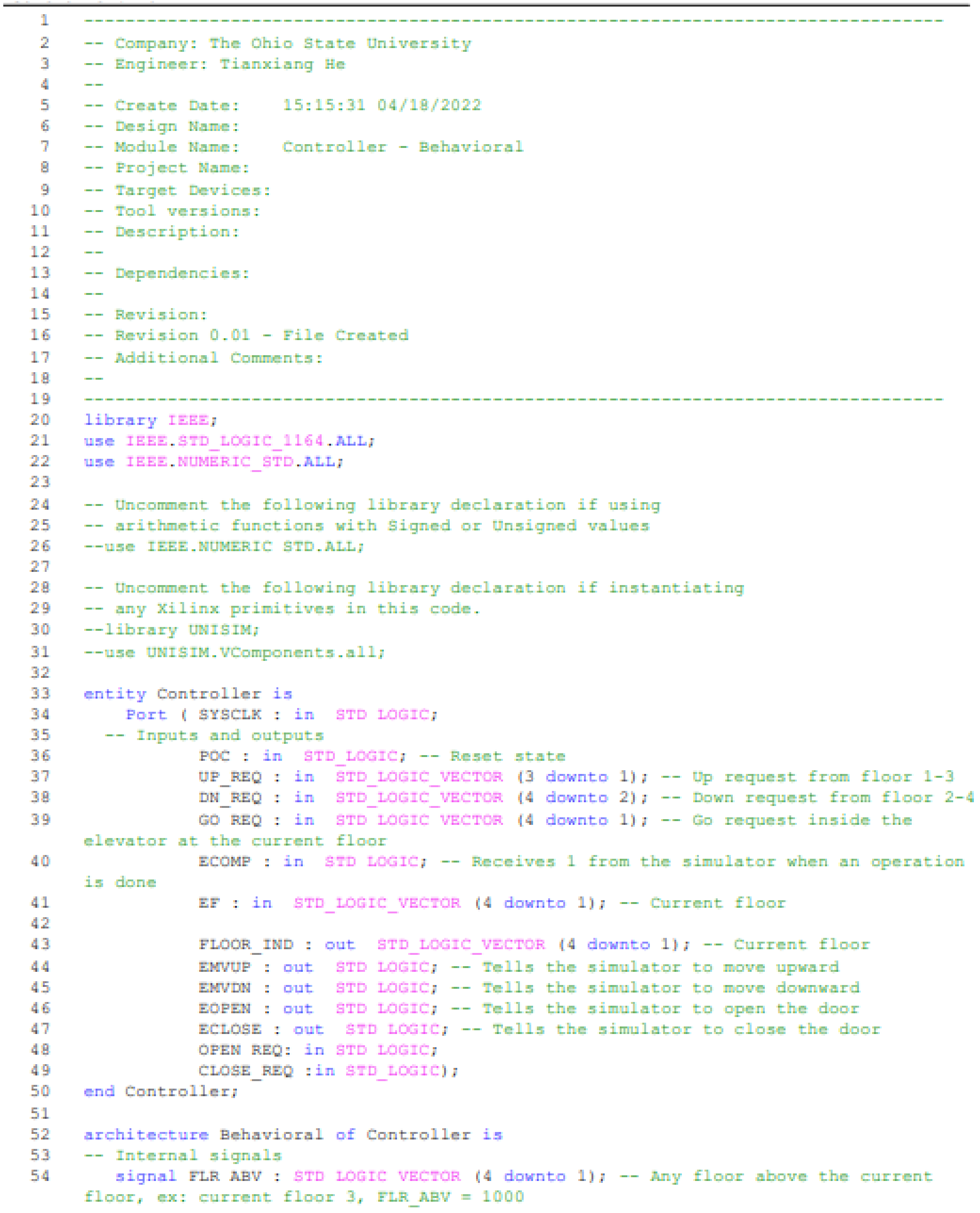
Figure A2: Elevator VHDL internal signals and port mapping



```

       );


end Behavioral;


```

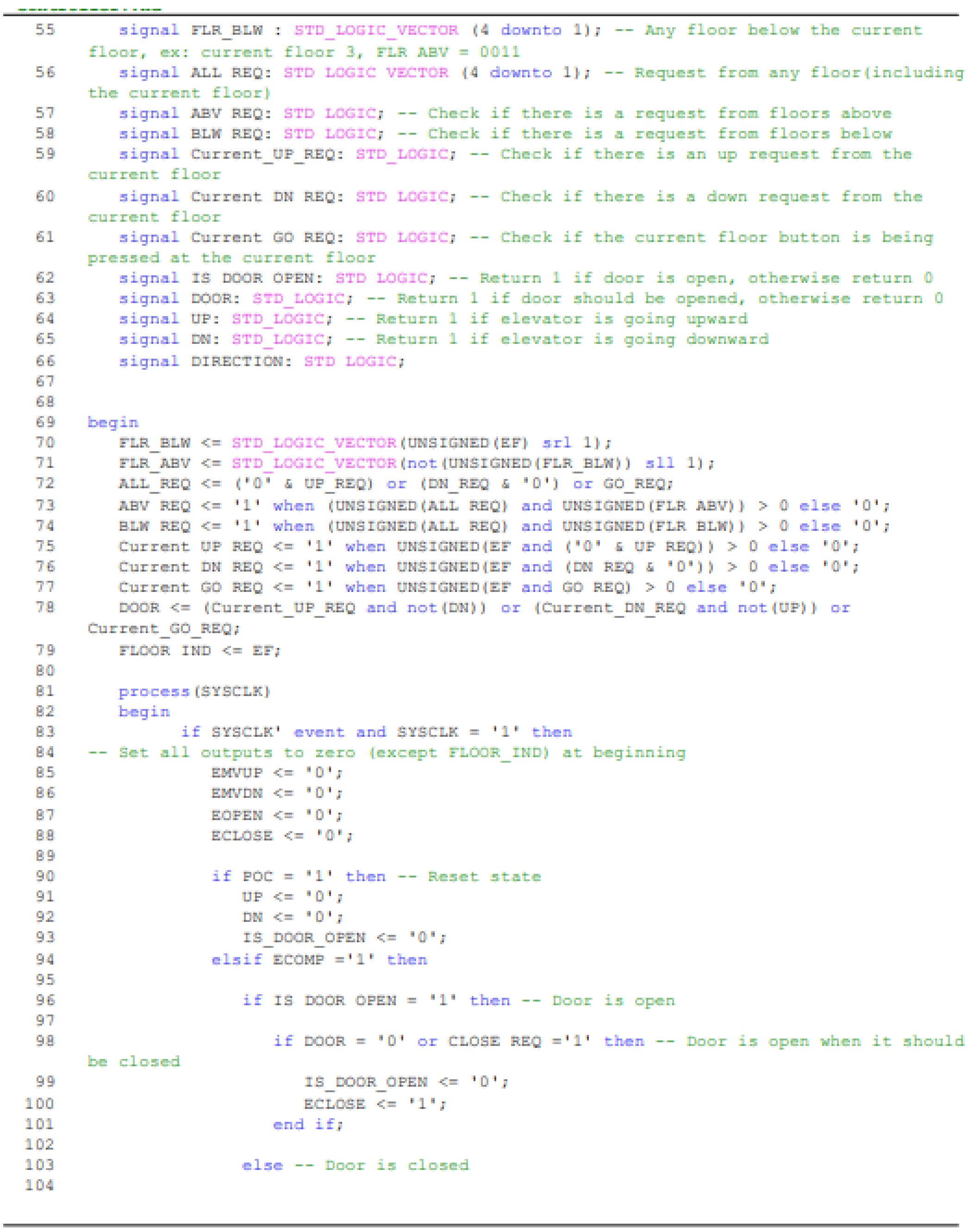
Figure A3: Elevator VHDL ending the code

APPENDIX B: Controller VHDL codes



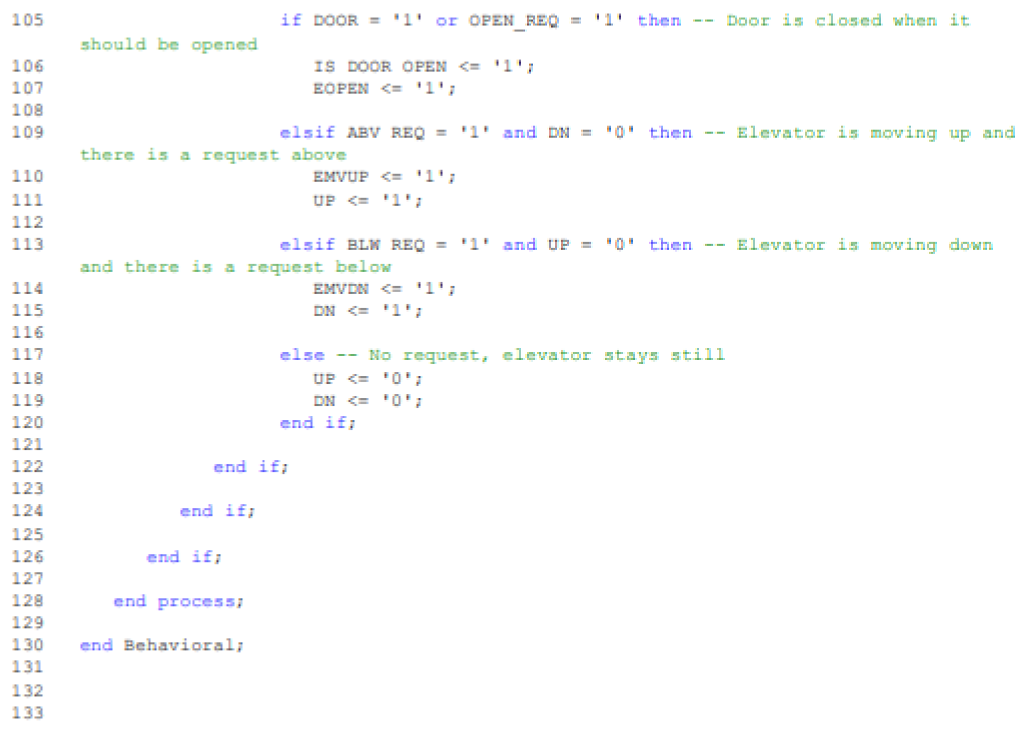
```vhdl
----------------------------------------------------------------------------------
-- Company: The Ohio State University
-- Engineer: Tianxiang He
--
-- Create Date:    15:15:31 04/18/2022
-- Design Name:
-- Module Name:    Controller - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Controller is
    Port ( SYSCLK : in  STD_LOGIC;
  -- Inputs and outputs
           POC : in  STD_LOGIC; -- Reset state
           UP_REQ : in  STD_LOGIC_VECTOR (3 downto 1); -- Up request from floor 1-3
           DN_REQ : in  STD_LOGIC_VECTOR (4 downto 2); -- Down request from floor 2-4
           GO_REQ : in  STD_LOGIC_VECTOR (4 downto 1); -- Go request inside the elevator at the current floor
           ECOMP : in  STD_LOGIC; -- Receives 1 from the simulator when an operation is done
           EF : in  STD_LOGIC_VECTOR (4 downto 1); -- Current floor

           FLOOR_IND : out  STD_LOGIC_VECTOR (4 downto 1); -- Current floor
           EMVUP : out  STD_LOGIC; -- Tells the simulator to move upward
           EMVDN : out  STD_LOGIC; -- Tells the simulator to move downward
           EOPEN : out  STD_LOGIC; -- Tells the simulator to open the door
           ECLOSE : out  STD_LOGIC; -- Tells the simulator to close the door
           OPEN_REQ: in STD_LOGIC;
           CLOSE_REQ :in STD_LOGIC);
end Controller;

architecture Behavioral of Controller is
-- Internal signals
   signal FLR_ABV : STD_LOGIC_VECTOR (4 downto 1); -- Any floor above the current floor, ex: current floor 3, FLR_ABV = 1000
```

Figure B1: Controller VHDL code page one



```vhdl
   signal FLR_BLW : STD_LOGIC_VECTOR (4 downto 1); -- Any floor below the current floor, ex: current floor 3, FLR_ABV = 0011
   signal ALL_REQ: STD_LOGIC_VECTOR (4 downto 1); -- Request from any floor(including the current floor)
   signal ABV_REQ: STD_LOGIC; -- Check if there is a request from floors above
   signal BLW_REQ: STD_LOGIC; -- Check if there is a request from floors below
   signal Current_UP_REQ: STD_LOGIC; -- Check if there is an up request from the current floor
   signal Current_DN_REQ: STD_LOGIC; -- Check if there is a down request from the current floor
   signal Current_GO_REQ: STD_LOGIC; -- Check if the current floor button is being pressed at the current floor
   signal IS_DOOR_OPEN: STD_LOGIC; -- Return 1 if door is open, otherwise return 0
   signal DOOR: STD_LOGIC; -- Return 1 if door should be opened, otherwise return 0
   signal UP: STD_LOGIC; -- Return 1 if elevator is going upward
   signal DN: STD_LOGIC; -- Return 1 if elevator is going downward
   signal DIRECTION: STD_LOGIC;


begin
   FLR_BLW <= STD_LOGIC_VECTOR(UNSIGNED(EF) srl 1);
   FLR_ABV <= STD_LOGIC_VECTOR(not(UNSIGNED(FLR_BLW)) sll 1);
   ALL_REQ <= ('0' & UP_REQ) or (DN_REQ & '0') or GO_REQ;
   ABV_REQ <= '1' when (UNSIGNED(ALL_REQ) and UNSIGNED(FLR_ABV)) > 0 else '0';
   BLW_REQ <= '1' when (UNSIGNED(ALL_REQ) and UNSIGNED(FLR_BLW)) > 0 else '0';
   Current_UP_REQ <= '1' when UNSIGNED(EF and ('0' & UP_REQ)) > 0 else '0';
   Current_DN_REQ <= '1' when UNSIGNED(EF and (DN_REQ & '0')) > 0 else '0';
   Current_GO_REQ <= '1' when UNSIGNED(EF and GO_REQ) > 0 else '0';
   DOOR <= (Current_UP_REQ and not(DN)) or (Current_DN_REQ and not(UP)) or Current_GO_REQ;
   FLOOR_IND <= EF;

   process(SYSCLK)
   begin
         if SYSCLK' event and SYSCLK = '1' then
-- Set all outputs to zero (except FLOOR_IND) at beginning
            EMVUP <= '0';
            EMVDN <= '0';
            EOPEN <= '0';
            ECLOSE <= '0';

            if POC = '1' then -- Reset state
               UP <= '0';
               DN <= '0';
               IS_DOOR_OPEN <= '0';
            elsif ECOMP ='1' then

               if IS_DOOR_OPEN = '1' then -- Door is open

                  if DOOR = '0' or CLOSE_REQ ='1' then -- Door is open when it should be closed
                     IS_DOOR_OPEN <= '0';
                     ECLOSE <= '1';
                  end if;

               else -- Door is closed

```

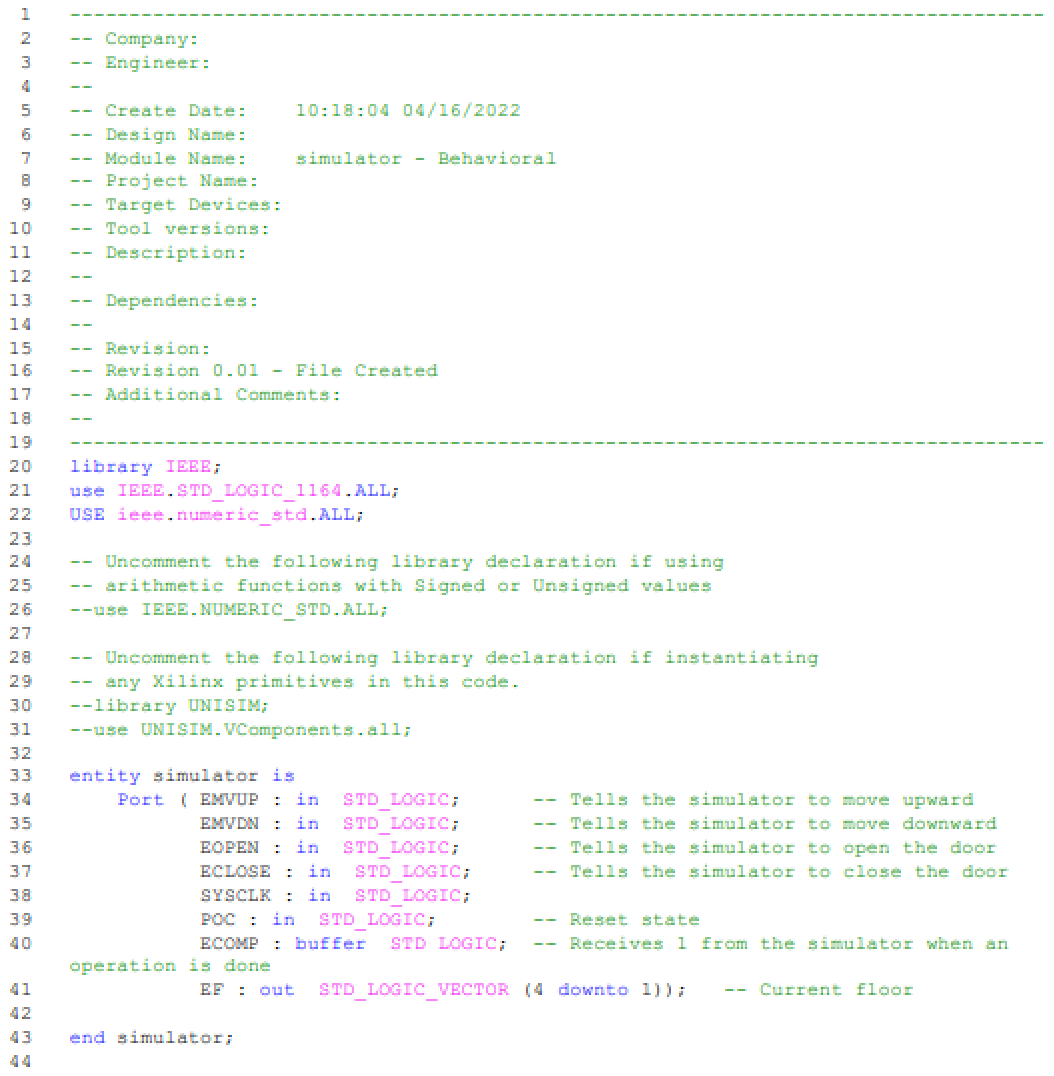
Figure B2: Controller VHDL code page two



```vhdl
                  if DOOR = '1' or OPEN_REQ = '1' then -- Door is closed when it should be opened
                     IS DOOR OPEN <= '1';
                     EOPEN <= '1';

                  elsif ABV REQ = '1' and DN = '0' then -- Elevator is moving up and there is a request above
                     EMVUP <= '1';
                     UP <= '1';

                  elsif BLW REQ = '1' and UP = '0' then -- Elevator is moving down and there is a request below
                     EMVDN <= '1';
                     DN <= '1';

                  else -- No request, elevator stays still
                     UP <= '0';
                     DN <= '0';
                  end if;

            end if;

         end if;

      end if;

   end process;

end Behavioral;
```

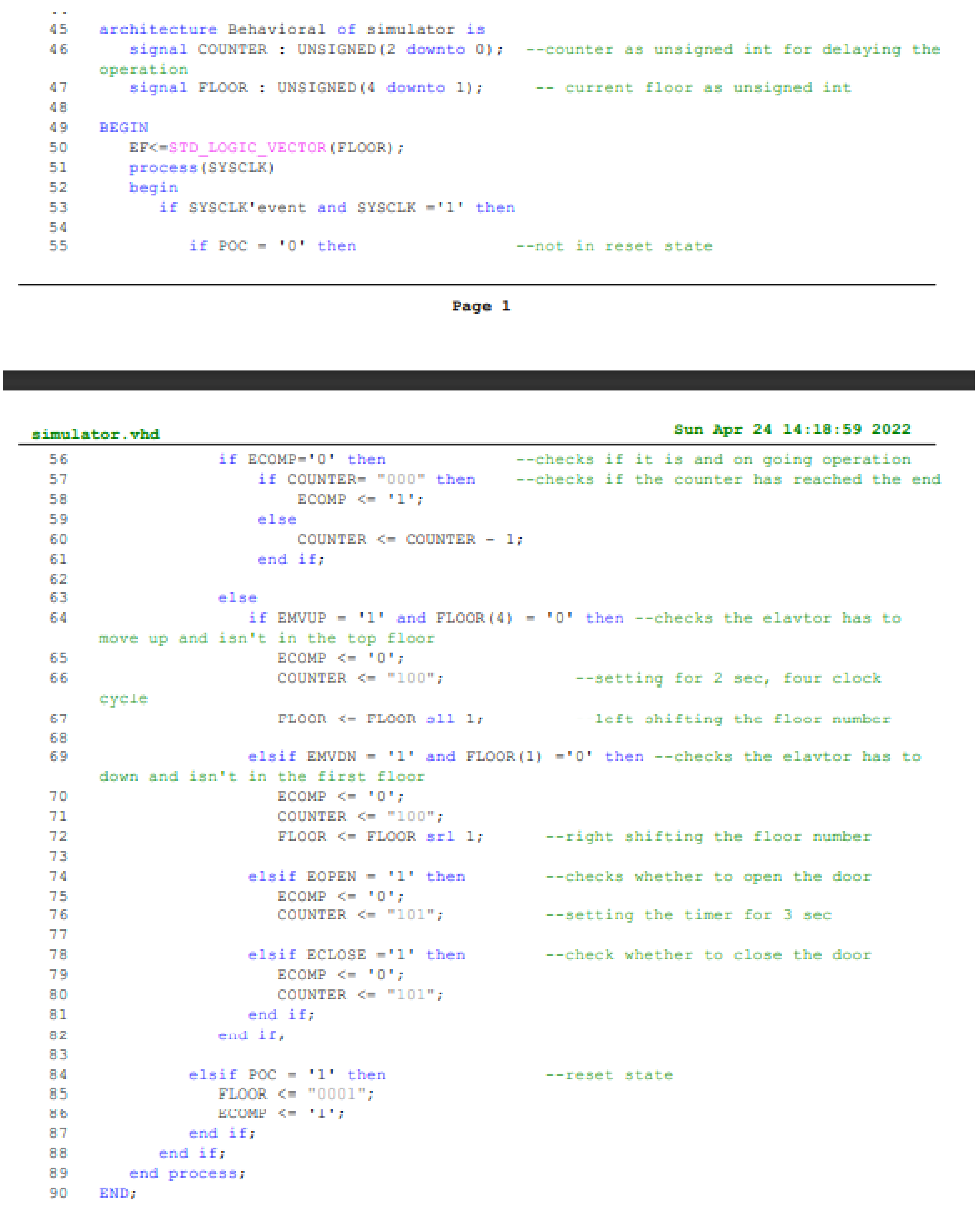
Figure B3: Controller VHDL code page three

APPENDIX C: Simulator



```
----------------------------------------------------------------------------------
-- Company: 
-- Engineer: 
-- 
-- Create Date:    10:18:04 04/16/2022 
-- Design Name: 
-- Module Name:    simulator - Behavioral 
-- Project Name: 
-- Target Devices: 
-- Tool versions: 
-- Description: 
--
-- Dependencies: 
--
-- Revision: 
-- Revision 0.01 - File Created
-- Additional Comments: 
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity simulator is
    Port ( EMVUP : in  STD_LOGIC;       -- Tells the simulator to move upward
           EMVDN : in  STD_LOGIC;       -- Tells the simulator to move downward
           EOPEN : in  STD_LOGIC;       -- Tells the simulator to open the door
           ECLOSE : in  STD_LOGIC;      -- Tells the simulator to close the door
           SYSCLK : in  STD_LOGIC;
           POC : in  STD_LOGIC;         -- Reset state
           ECOMP : buffer  STD_LOGIC;  -- Receives 1 from the simulator when an 
operation is done
           EF : out  STD_LOGIC_VECTOR (4 downto 1));    -- Current floor

end simulator;

```

Figure C1: Simulator VHDL signal declaration

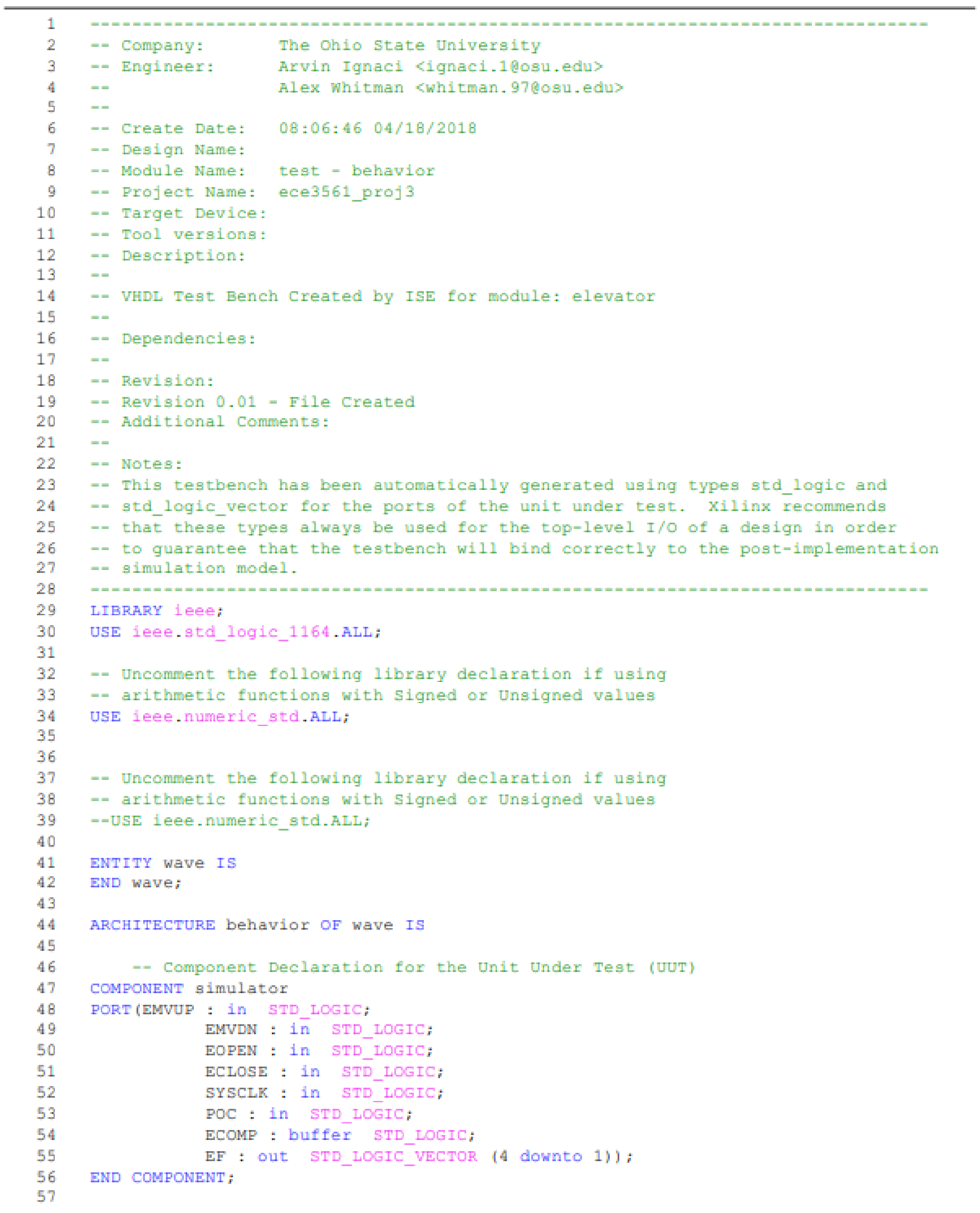


```vhdl
--
architecture Behavioral of simulator is
   signal COUNTER : UNSIGNED(2 downto 0);  --counter as unsigned int for delaying the operation
   signal FLOOR : UNSIGNED(4 downto 1);     -- current floor as unsigned int

BEGIN
   EF<=STD_LOGIC_VECTOR(FLOOR);
   process(SYSCLK)
   begin
      if SYSCLK'event and SYSCLK ='1' then

         if POC = '0' then                --not in reset state
```

```vhdl
            if ECOMP='0' then             --checks if it is and on going operation
                if COUNTER= "000" then    --checks if the counter has reached the end
                    ECOMP <= '1';
                else
                    COUNTER <= COUNTER - 1;
                end if;

            else
               if EMVUP = '1' and FLOOR(4) = '0' then --checks the elavtor has to move up and isn't in the top floor
                  ECOMP <= '0';
                  COUNTER <= "100";             --setting for 2 sec, four clock cycle
                  FLOOR <= FLOOR sll 1;         --left shifting the floor number

               elsif EMVDN = '1' and FLOOR(1) ='0' then --checks the elavtor has to down and isn't in the first floor
                  ECOMP <= '0';
                  COUNTER <= "100";
                  FLOOR <= FLOOR srl 1;      --right shifting the floor number

               elsif EOPEN = '1' then        --checks whether to open the door
                  ECOMP <= '0';
                  COUNTER <= "101";          --setting the timer for 3 sec

               elsif ECLOSE ='1' then        --check whether to close the door
                  ECOMP <= '0';
                  COUNTER <= "101";
               end if;
            end if;

         elsif POC = '1' then                --reset state
            FLOOR <= "0001";
            ECOMP <= '1';
         end if;
      end if;
   end process;
END;
```

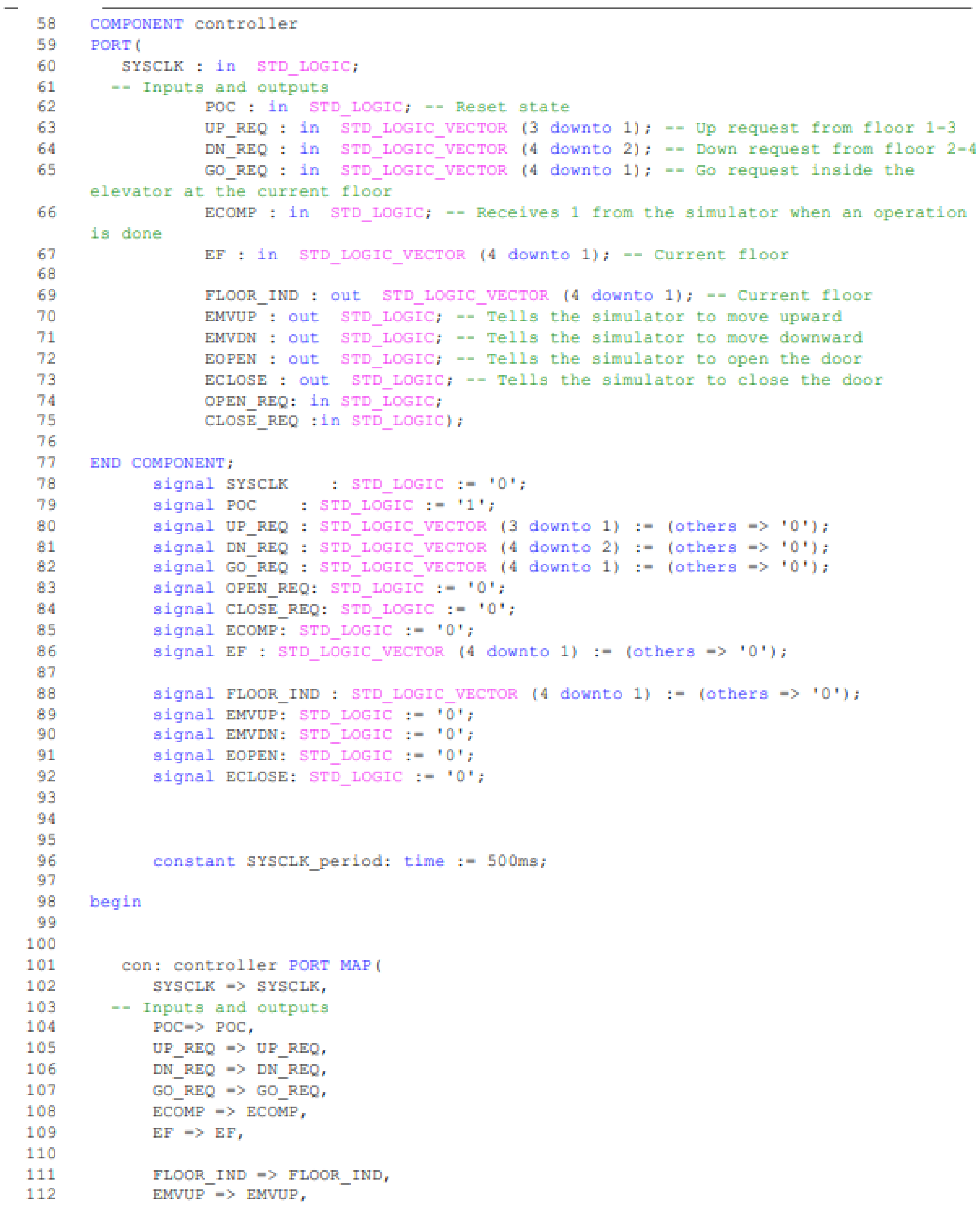
Figure C2: Simulator VHDL function of the simulator

APPENDIX D: Wave generator



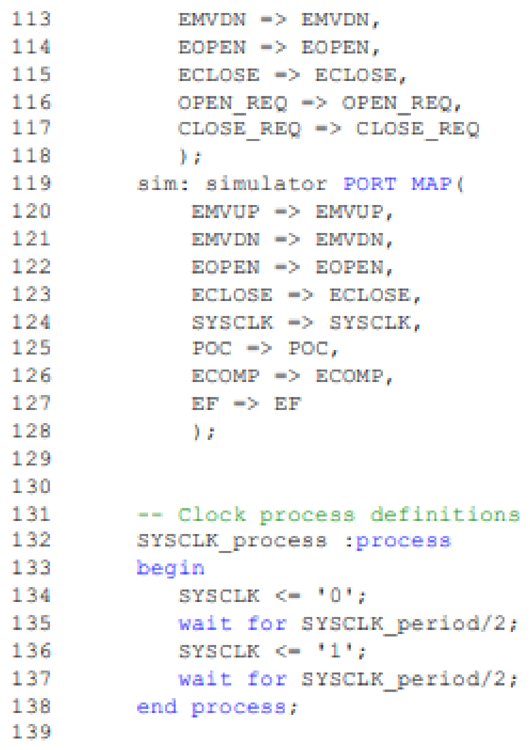
```vhdl
--------------------------------------------------------------------------------
-- Company:       The Ohio State University
-- Engineer:      Arvin Ignaci <ignaci.1@osu.edu>
--                Alex Whitman <whitman.97@osu.edu>
--
-- Create Date:   08:06:46 04/18/2018
-- Design Name:
-- Module Name:   test - behavior
-- Project Name:  ece3561_proj3
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: elevator
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
USE ieee.numeric_std.ALL;


-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY wave IS
END wave;

ARCHITECTURE behavior OF wave IS

    -- Component Declaration for the Unit Under Test (UUT)
COMPONENT simulator
PORT(EMVUP : in  STD_LOGIC;
           EMVDN : in  STD_LOGIC;
           EOPEN : in  STD_LOGIC;
           ECLOSE : in  STD_LOGIC;
           SYSCLK : in  STD_LOGIC;
           POC : in  STD_LOGIC;
           ECOMP : buffer  STD_LOGIC;
           EF : out  STD_LOGIC_VECTOR (4 downto 1));
END COMPONENT;

```

Figure D1: Wave VHDL signal declaration part 1



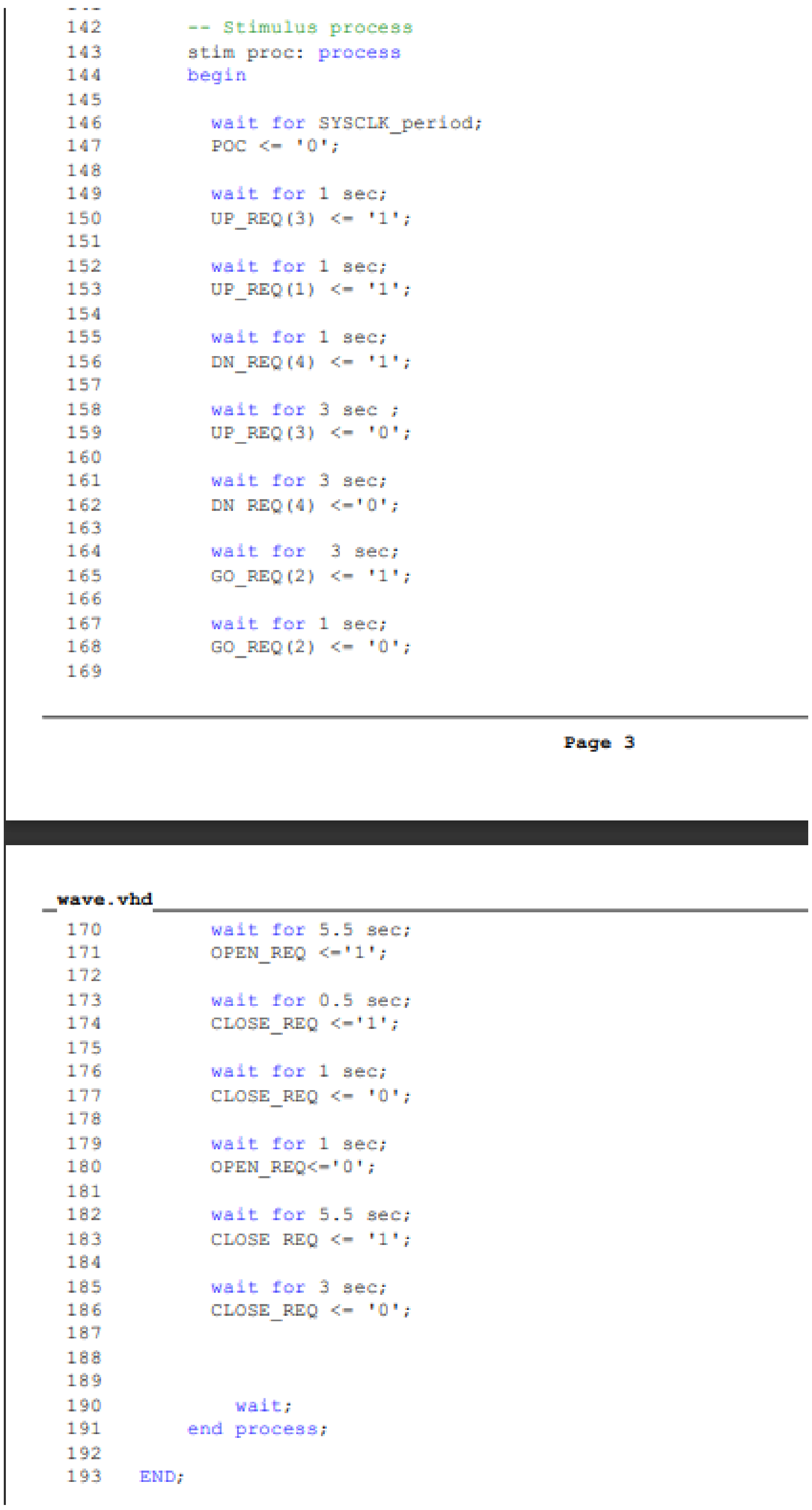
```vhdl
COMPONENT controller
PORT(
   SYSCLK : in  STD_LOGIC;
  -- Inputs and outputs
          POC : in  STD_LOGIC; -- Reset state
          UP_REQ : in  STD_LOGIC_VECTOR (3 downto 1); -- Up request from floor 1-3
          DN_REQ : in  STD_LOGIC_VECTOR (4 downto 2); -- Down request from floor 2-4
          GO_REQ : in  STD_LOGIC_VECTOR (4 downto 1); -- Go request inside the elevator at the current floor
          ECOMP : in  STD_LOGIC; -- Receives 1 from the simulator when an operation is done
          EF : in  STD_LOGIC_VECTOR (4 downto 1); -- Current floor

          FLOOR_IND : out  STD_LOGIC_VECTOR (4 downto 1); -- Current floor
          EMVUP : out  STD_LOGIC; -- Tells the simulator to move upward
          EMVDN : out  STD_LOGIC; -- Tells the simulator to move downward
          EOPEN : out  STD_LOGIC; -- Tells the simulator to open the door
          ECLOSE : out  STD_LOGIC; -- Tells the simulator to close the door
          OPEN_REQ: in STD_LOGIC;
          CLOSE_REQ :in STD_LOGIC);

END COMPONENT;
      signal SYSCLK    : STD_LOGIC := '0';
      signal POC    : STD_LOGIC := '1';
      signal UP_REQ : STD_LOGIC_VECTOR (3 downto 1) := (others => '0');
      signal DN_REQ : STD_LOGIC_VECTOR (4 downto 2) := (others => '0');
      signal GO_REQ : STD_LOGIC_VECTOR (4 downto 1) := (others => '0');
      signal OPEN_REQ: STD_LOGIC := '0';
      signal CLOSE_REQ: STD_LOGIC := '0';
      signal ECOMP: STD_LOGIC := '0';
      signal EF : STD_LOGIC_VECTOR (4 downto 1) := (others => '0');

      signal FLOOR_IND : STD_LOGIC_VECTOR (4 downto 1) := (others => '0');
      signal EMVUP: STD_LOGIC := '0';
      signal EMVDN: STD_LOGIC := '0';
      signal EOPEN: STD_LOGIC := '0';
      signal ECLOSE: STD_LOGIC := '0';



      constant SYSCLK_period: time := 500ms;

begin


   con: controller PORT MAP(
      SYSCLK => SYSCLK,
  -- Inputs and outputs
      POC=> POC,
      UP_REQ => UP_REQ,
      DN_REQ => DN_REQ,
      GO_REQ => GO_REQ,
      ECOMP => ECOMP,
      EF => EF,

      FLOOR_IND => FLOOR_IND,
      EMVUP => EMVUP,
```

Figure D2: Wave VHDL signal declaration part 2



```vhdl
   EMVDN => EMVDN,
   EOPEN => EOPEN,
   ECLOSE => ECLOSE,
   OPEN_REQ => OPEN_REQ,
   CLOSE_REQ => CLOSE_REQ
   );
sim: simulator PORT MAP(
    EMVUP => EMVUP,
    EMVDN => EMVDN,
    EOPEN => EOPEN,
    ECLOSE => ECLOSE,
    SYSCLK => SYSCLK,
    POC => POC,
    ECOMP => ECOMP,
    EF => EF
    );


-- Clock process definitions
SYSCLK_process :process
begin
   SYSCLK <= '0';
   wait for SYSCLK_period/2;
   SYSCLK <= '1';
   wait for SYSCLK_period/2;
end process;

```

Figure D3: Wave VHDL signal declaration part 3



```
    -- Stimulus process
    stim proc: process
    begin

      wait for SYSCLK_period;
      POC <= '0';

      wait for 1 sec;
      UP_REQ(3) <= '1';

      wait for 1 sec;
      UP_REQ(1) <= '1';

      wait for 1 sec;
      DN_REQ(4) <= '1';

      wait for 3 sec ;
      UP_REQ(3) <= '0';

      wait for 3 sec;
      DN REQ(4) <='0';

      wait for  3 sec;
      GO_REQ(2) <= '1';

      wait for 1 sec;
      GO_REQ(2) <= '0';

```

Page 3

wave.vhd

```
      wait for 5.5 sec;
      OPEN_REQ <='1';

      wait for 0.5 sec;
      CLOSE_REQ <='1';

      wait for 1 sec;
      CLOSE_REQ <= '0';

      wait for 1 sec;
      OPEN_REQ<='0';

      wait for 5.5 sec;
      CLOSE REQ <= '1';

      wait for 3 sec;
      CLOSE_REQ <= '0';



        wait;
    end process;

END;
```

Figure D4: Wave VHDL stimulus process