ELEVATOR CONTROL SYSTEM

Deliverable II
ECSE 421 – Embedded Systems

McGill University
Professor Brett Meyer

Team RADE
Riham Bichri 260380813
Alex Varghese 260327013
Dmitry Rivkin 260374466
Edward Chiang 260209427

April 3, 2012
Figure D2.1A the Overview of the System (*ZOOM IN to see Labels)
Overview

Figure D2.1A show the early executable model of the elevator system. The system consists of an `elevatorButtonController` (determines which button was pressed from inside the elevator), an `hallButtonController` (determines which hall button was pressed), a `buttonControlBuffer` (sends one output from the `hallButtonController` and `elevatorController` at a time), a `dispatcher` (determines which elevator will get which floor) an `elevatorController` (the order that the floors will be visited for each elevator) and three `door_controllers` (controls the doors).

Dispatcher

Overview and Functions:

The dispatcher process is responsible for determining which floors each elevator should be sent to, and sending these instructions to the elevator controller. The dispatcher does not determine what order the tasks (stops) will be executed in. It must ensure that no user has to spend too long getting to their desired floor. Also it needs to minimize the amount of energy used by the elevator system and communicate with the button controller and the elevator controller.

The dispatcher assigns a different priority to each elevator. Elevator 1 is of low priority, elevator 2 is of medium priority, and elevator 3 is of high priority. When a request is made, it is initially dispatched to the low priority elevator. If the task is not serviced within 45 seconds, it is re-assigned to the medium priority elevator, and if it is not fulfilled within 90 seconds, it is assigned to the high priority elevator. If there are only a few users, only one elevator will be used, and energy will be conserved. If there are many users, the elevator system will use its full capacity to ensure that no one has to wait for the elevator for too long.

Port Declaration:

```
port(
  rqex, flr_rq: in std_logic_vector(8 downto 0); --rqex: from elevator ctrl tells that we arrived at floor
  --flr rq: from buttonctrlBuffer. Indicate new requests
  rqex_new, flr_rq_new: in std_logic; -- flags to indicate information on lines
  clk: in std_logic; --clock
  ok: in std_logic_vector(1 downto 0); -- floor removal from elevator was successful
  add, remove: out std_logic_vector(8 downto 0)); -- add/remove floor from elevator
  add_flag, remove_flag: out std_logic; -- flag to indicate removal/addition
```

The signals `rqex`, `flr_rq`, `add`, and `remove` are all data signals of the same format.

```
0 00 0 0000
```

*Figure D2.1b: bit8: Signal Origin; bit 7-6: Elevator Number; bit 5: Direction; bit 4-0: Desired Floor*

They are all 9 bit std_logic_vectors (8 downto 0). Bit 8 signals whether the request came from inside the elevator or out in the hall. Bits 7 and 6 specify which elevator the request came from (“00” if it came from the hall). Bit 5 specifies if the request is for an up moving or down moving elevator. Bits 4-0 specify the floor the request is for (“00001” means floor 1).

The signals `rqex_new`, `flr_rq_new`, `add_flag`, and `remove_flag` are single bit vectors which are normally set to 0, but setting them to 0 for one clock cycle indicates new information on the data lines. The signal `ok` is feedback from the elevator controller.
Dispatcher Processes:
The dispatcher runs three sub processes: think, timer, and turn. Think implements a Moore-style FSM which executes the algorithm. Timer keeps track of the time. Turn controls the execution of think switching between super-states every two clock cycles.

Think:
When a floor request is made, think checks where it was made from. If it was made from an elevator, dispatch it to that elevator, unless the elevator is already dispatched to that floor. If it was made from the hall, check that elevator 1 isn’t already going to that floor, otherwise dispatch elevator 1 to that floor. Dispatches, and dispatch times (in seconds), are kept track of in local RAMs, one for each elevator.

When the elevator control tells the dispatcher that it arrived has executed a task, think modifies the memory to reflect that change. Every time a second passes, think checks every location in the RAM for elevator 1 to see if any of the dispatches for requests made from the hall have been waiting for over 45 seconds. If they have, think asks the controller to remove that task from elevator 1. If the task is removed successfully, the elevator controller signals successful removal by setting ok to “11” for a clock cycle. If the task is the next stop, the controller will signal that the removal was denied by setting ok to “10” for one clock cycle.

Every 512 seconds the time is reset to zero, and the dispatch times are updated to negative values so that they continue to reflect how long ago each dispatch occurred.

Turn
Every two cycles, turn switches think between handling requests, successful executions, and time updates. Handling requests and successful executions takes 2 cycles, so the entire action can be performed in one turn. The time update function is subdivided into tasks such that each one can be executed in one or two clock cycles. In the worst case scenario, a time update would take 900 clock cycles to execute. Since it only gets 2 clock cycles in every six, in the worst case scenario, a time update would complete 2700 roughly clock cycles after the time was updated. The exact number depends on how many clock cycles the dispatcher has to spend waiting on removal approval from the elevator controller. Since this only happens once every second, with a clock rate of around 50 MHz this would be fast enough.

Timer
The timer increments the time by one every time a seconds passes. When it gets to 511 it loops back to zero.
Test set 1:

In order to test the dispatcher, we must test whether it can dispatch requests properly. There are two cases in which it would do a dispatch. Either a signal came in from a button or a request has been waiting for too long. In the case of a request coming in from outside, we must test whether the dispatcher dispatches it to the correct elevator. For a re-dispatch due to time, we must check that the dispatcher first removes the task from the elevator it was originally assigned to and then dispatches it to the correct elevator.

For the case of a request we will test four cases: A. The request came from the hall. B. The request came from elevator 1. C. The request came from elevator 2. D. The request came from elevator 3. We will look at the output signals and response times to determine if the system is working properly.

Since we don’t want to simulate 90 seconds of a 50MHZ clock, we will reduce the timeout times several orders of magnitude to test the case when the request is re-dispatched to a different elevator because it’s been waiting for too long. Then we will input a floor request coming from the hall. Then, we will not satisfy the request for the appropriate amount of time and see if the task is removed from elevator 1 and given to elevator 2, and then later removed from elevator 2 and assigned to elevator 3.
Elevator Controller

Each elevator is represented by a vector of length 20. The indexes of the array are the 20 floors that will be accessed by the elevator. The value contained in the array is the direction. For example: if an elevator was called from the hall at floor 15 with a request to go up, the array at index 14 will store the value up. If a button was pressed from within the elevator, the direction will be stored as the current direction of the elevator.

How we decide which next floor will be visited:
Each elevator is in a state, either up or down. When this decision is being made, isMoving is false. If the elevator is in the up state:
1. Find the highest requested floor
2. Look for floors requesting to go up between the current and top most requested floor (note that floors where a stop is requested have the direction of the current direction of the elevator). Go to the one closest to the current floor.
3. If there are no floors as described in point 2, go to the highest requested floor.

If the elevator is in the down state:
1. Find the lowest requested floor
2. Look for floors requesting to go down between the current and bottom most requested floor. Go to the one closest to the current floor.
3. If there are no floors as described in point 2, go to the lowest requested floor.

--index-floor, element in vector = current direction requested from that floor '1'=up, '0'=down, 'Z'=nothing
signal elevator1Floor, elevator2Floor, elevator3Floor : std_logic_vector (19 downto 0) :="ZZZZZZZZZZZZZZZZZ"; 

Figure D2.A
Example:

To illustrate how the elevator control system works:

Figure D2.2 shows the current requests for the elevator. The elevator is currently at floor 9. The order in which the floors will be visited: 9, 11, 17, 19, 15, 4.

Arrival to desired floor:

As seen from the code in figure D2.B, once this has been evaluated, the elevator is told to move in the appropriate direction. `IsMoving` is now true. A timer is set. The elevator moves until it reaches the desired floor (indicated by the `lightSensor` input). Once it reaches the desired floor, `move` is set so that the elevator stops moving. `AtFloor` is sent to the door controller to notify that the doors should be opened, and a signal `arrived` and pulse `arrivedPulse` are sent to the dispatcher (done at most every 10 clock cycles) and a signal is sent to the button controller to turn off the appropriate lights. The `currentFloor` is set as the `nextFloor` and the state becomes the direction stored in the floor where we just arrived. The elevator can only move if it receives a signal `doorClosed` from the door controller. `IsMoving` is set to 1. If `IsMoving` isn’t set to 1 within 5 minutes, the alarm rings.

```
if ( pulseCounterTaken = "00" or pulseCounterTaken = "11" ) then
    pulseCount <= pulseCount +1;
    if ( pulseCount = "1010" ) then
        arrived <= '0' || '0' || '1' || currentFloor3;
        arrivedPulse='1';
        lightButtonController <= '0' || '1' || '0' || currentFloor1;
        lightPulse='1';
        pulseCount <= "0000";
        pulseCounterTaken <= "00";
    else
        arrivedPulse='0';
        lightPulse='0';
        end if;
    end if;
```

Figure D2.B
Adding-removing floors
The elevator controller gets the floors to visit from the dispatcher through the add line. Whenever the dispatcher requests to add a new floor, a pulse add_flag is sent, and the floor is added to the corresponding elevator list.

```plaintext
-- add new data to the elevator matrices
if (add_flag) then
    if (add(7 downto 6) = "01") then
        elevator1Floor( add(4 downto 0)) <= add(5);
    elseif (add(7 downto 6) = "10") then
        elevator2Floor( add(4 downto 0)) <= add(5);
    elif (add(7 downto 6) = "11") then
        elevator3Floor( add(4 downto 0)) <= add(5);
    end if;
end if;
```

The dispatcher can also remove floors from each elevator. This is done through the remove line, and a pulse flows through remove_flag. When this is done, a signalling pulse is sent through ok. This can only be done if the task to be removed is not the next floor to be accessed by the elevator. If the stop button is pressed, the tasks can be assigned to another elevator through the dispatcher. If the stop button is pressed for longer than 5 minutes, the alarm rings.

```plaintext
-- remove data from the elevator matrices
if (remove_flag) then
    if (remove(7 downto 6) = "01") then
        if (nextDirection1/=remove(4 downto 0) or stopButton ='1') then
            elevator1Floor( remove(4 downto 0)) <= 'Z';
            ok <= '1';
        end if;
    elseif (remove(7 downto 6) = "10" or stopButton ='1') then
        if (nextDirection2/=remove(4 downto 0)) then
            elevator2Floor( remove(4 downto 0)) <= 'Z';
            ok<= '1';
        end if;
    elseif (add (7 downto 6) = "11" or stopButton ='1') then
        if (currentDirection3/=remove(4 downto 0)) then
            elevator3Floor(remove(4 downto 0)) <= 'Z';
            ok<= '1';
        end if;
    end if;
end if;
```

Figure D2.C

Figure D2.D
Test Set 2 (Elevator Controller)

In order to test, we will be feeding the elevator’s `currentFloor` and `currentDirection` with different requests and directions. If the elevator controller is operating incorrectly, we should observe an incorrect signal for `nextDirection` and `nextFloor`. The expected behavior would be the one determined by the algorithm defined above.

In order to determine whether the elevator is correctly detecting that it has reached the desired floor, we will input different values for the `lightSensor` readings. If the output is correct, the elevator should stop moving when it reaches the desired floor. `move` should be Z, a value should be sent to the dispatcher, and the button controller.

We will also test the entire elevator controller by entering a sequence of input floors and checking the resulting output floors.
### Hall Button Controller

The purpose of the hall button controller is to take the hall buttons’ up/down requests signal from each floor and send them to the dispatcher. Each floor, except for first floor and the top floor, has an up and down button. Since there are 20 floors, there will be 38 button signals inputting into the hall button controller. The controller will take the current elevator position and the direction from the elevator controller to determine the on/off status of the button light. The hall button controller will output the requested direction with the floor the request was sent from and the button’s light status.

**The below is the symbol diagram of the hall button controller:**

In the controller, the 38 button signals are sent as either 1 or 0, where if a 1 is sent it means the button is pressed. The current floor is sent as a 5 bit vector so that all 20 floors can be represented. The floor direction signal is a 9 bit vector where the first three bits indicate the button’s location. This is set to “000” since this controller only deals with hall buttons. The other 6 bits represent direction and the floor where the request came from.

U1 to U19 signals are up button signals and the number in the signal’s name represent the floor number. D2 to D20 signals are down button signals. First, the direction signal is found by performing an OR operation on all the up signals and down signals separately. Each floor is associated with a floor number through their binary representation. UL1 to UL19 and DL2 to DL20 represent the light status of the buttons. The controller checks if the floor meets one of the light off requests from the elevator controller so that a light may be turned off.

![Hall Button Controller Diagram](image)
Figure D2.5 is a waveform which shows that when the up button from the first floor (U1) is pressed then “000100001” is outputted to the floor direction. And when U2 is pressed the “000100010” is outputted. It can be seen that the result is sent after 10 clock cycles so that the dispatcher may handle the output correctly. This is the basic functionality of the hall button controller.

Test set 3 (Hall Button Controller)

To determine if the model functions in the desired manner we will perform three major functional tests. The first test will check whether the component send the proper floor request to the dispatcher when a given button is pressed. The second test will check whether the light turns off when a request is met. The last test will check whether the controller does a proper handle when two buttons are pressed at once. It is expected that if the requests are not handled, the button light will not turn on and the user will know he/she must press this button again.
**Elevator Button Controller**

The purpose of the elevator button controller is to take the elevator buttons (floor numbers) requests signal from each elevator and send them to the dispatcher. The controller will also take the current elevator’s position so that it may determine which button to turn off.

To check if a request is met a “at_floor” 8 bit signal is feed at the input, where the first three bits represent which elevator the request came from. The 5th bit represents the direction of the elevator and the rest represent the floor number. It takes a clock input to make sure that the requests are sent every 10 clock cycles to the dispatcher. The controller outputs the status of the lamp (on or off) for the 60 buttons since the stop button will not have lamps. The request outputted is represent as floor_output signal (8 bit signal) and takes the bit structure as the “at_floor” signal mentioned above. However, the 5th bit represents the direction the elevator must take to get to the requested floor.

**Test Set 4 (Elevator Button)**

To make sure this component is fully functional we will have three major tests. One to test if a button is pressed we are given the proper output. Another to see if the lights turn off and on when a request is meet and sent respectively. The last test is to see if two or more buttons is pressed at the same time the controller handles at least one request and lets the user know that if request is not meet the button does not light up.

**Button Control Buffer**

The buffer takes a floor request and pulses from the elevator controller and the hall button controller as inputs at the same time and sends one output and impulse at the same time while storing the other till the next cycle.
Door Controller

The door controller software will have a timer. Once the timer exceeds the max allowable value, it will send a signal to the alarm system to sound off the alarm (not the same as 5 minutes alarm). The software will need to take the input from the load sensor and motion sensor, and then send a signal to the elevator controller to close or open the door.

*The below is the symbol diagram of the door controller:*

![Figure D2.6]

The door signal tells the motors to open or close the door. The controller takes weight readings from the load sensor, and these values are read only at one clock pulse. If the max weight is surpassed then the elevator doors will be kept open. A ‘1’ at the `at_floor` signal will indicate it is on a floor. Once the elevator reaches a floor it opens the doors, releases the lock and a signal is sent to start the timer. If time is up then alarm starts ringing until the door is closed. If there is an object in front of the doors a ‘1’ is sent to the `motion_sensor` signal and the doors are kept open. The doors are kept open for 5 seconds and should be sufficient time for the users to get in. Once the elevator closes the doors it sends a ‘1’ to the `ready_to_go` signal meaning the elevator is free to move again.

Figure D2.7 below it can be seen when the controller receives from the elevator controller that it is on the floor it opens the doors for 5 clock cycles.
Test Set 5 (Door Controller)

The following component will be tested with four functional tests. One will test if door opens and closes when arriving and leaving a floor. Second test will see if when the max weight is reached the doors stay open. Third test is to check if doors stay open when an obstacle is at the front doors. The last test will check that when the time is up the alarm should stay on until the door is closed.
Appendix1: VHDL Code (Dispatcher)

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dispatcher is
port(
  rqex,flr_rq:in std_logic_vector(8 downto 0); -- request executed and floor request. bit 8 is car. 7,6 are elevator number, 01 = elevator 1, 5 up, rest floor #
  rqex_new,flr_rq_new: in std_logic; -- flag a new input
  clk: in std_logic; --clock
  ok:in std_logic_vector(1 downto 0);
  add_flag,remove_flag: out std_logic;
  add,remove: out std_logic_vector(8 downto 0)); -- same bit breakdown as rqex, tells elevator control to add or remove a stop
end dispatcher;
architecture one of dispatcher is
component RAM
PORT
  (address : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
  clock : IN STD_LOGIC := '1';
  data : IN STD_LOGIC_VECTOR (19 DOWNTO 0);
  wren : IN STD_LOGIC ;
  q : OUT STD_LOGIC_VECTOR (19 DOWNTO 0)
);
end component;

signal flr_rq_flag: std_logic:='0';
signal rqex_flag: std_logic:='0';
signal t_flag: std_logic:='0';
signal tnew: std_logic:='0';
signal writ1: std_logic:='0';
signal writ2: std_logic:='0';
signal writ3: std_logic:='0';
signal cyc:std_logic:='0';
signal stateA, stateB,stateC1: std_logic:='0';
signal stateC:std_logic_vector(1 downto 0):="10";
signal stateC2, stateC3:std_logic_vector(2 downto 0):="000";
signal addr: std_logic_vector(5 downto 0);
signal dat1,dat2,dat3,task1,task2,task3: std_logic_vector(19 downto 0);
signal i,j,k,t,clock_count,dummy: integer:=0;
signal tn:std_logic_vector(1 downto 0):="01";
begin

ram1: RAM port map (address=> addr, clock=> clk, data=> dat1, wren=>writ1, q=> task1);
ram2: RAM port map (address=> addr, clock=> clk, data=> dat2, wren=>writ2, q=> task2);
ram3: RAM port map (address=> addr, clock=> clk, data=> dat3, wren=>writ3, q=> task3);

think: process(clk)
begin
if(clk'event and clk='1') then
  add_flag<='0';
  remove_flag<='0';
  if(flr_rq_new='1') then
    flr_rq_flag<='1';
  end if;
  if(rqex_new='1') then
    rqex_flag<='1';
  end if;
  if(tnew='1') then
    t_flag<='1';
  end if;
  if(flr_rq_flag='1' and tn="01") then
    case stateA is
    when '0'=>
      addr<=flr_rq(5 downto 0);
      dat1<="1" & flr_rq & std_logic_vector(to_signed(t,10));
      dat2<="1" & flr_rq & std_logic_vector(to_signed(t,10));
      dat3<="1" & flr_rq & std_logic_vector(t);
      writ1<='0';
      writ2<='0';
      writ3<='0';
      stateA<='1';
    when '1'=>
      if (flr_rq(8)='1') then
        case flr_rq(7 downto 6) is
        when "01"=>
          if(task1(19)='1') then
            dat1<=task1(19)&'1'&task1(17 downto 0);
          else
            add<=flr_rq;
            add_flag<='1';
          end if;
          writ1<='1';
        when "10"=>
          if(task2(19)='1') then
            dat2<=task2(19)&"1"&task2(17 downto 0);
          else
            add<=flr_rq;
            add_flag<='1';
          end if;
        end case;
      end if;
    end case;
  end if;
end if(clk'event and clk='1') then

end think;
end
else
  add<=flr_rq;
  add_flag<='1';
  end if;
  writ2<='1';
  when "11"=>
    if(task3(19)='1') then
      dat3<=task1(19)&"1"&task1(17 downto 0);
    else
      add<=flr_rq;
      add_flag<='1';
      end if;
      writ3<='1';
      when "00"=>
        end case;
    else
      if(task1(19)='0') then
        add<= flr_rq(8) & "01" & flr_rq(5 downto 0);
      end if;
      end if;
      stateA<='0';
      flr_rq_flag<='0';
      end case;
      if(rqex_flag='1' and tn="10") then
        case stateB is
          when '0'=>
            writ1<='0';
            writ2<='0';
            writ3<='0';
          case rqex(7 downto 6) is
            when "01"=>
              dat1<="00000000000000000000";
              writ1<='1';
            when "10"=>
              dat2<="00000000000000000000";
              writ2<='1';
            when "11"=>
              dat3<="00000000000000000000";
              writ3<='1';
            when others=>
              end case;
            when '1'=>
              stateB<='0';
              writ1<='0';
              writ2<='0';
              writ3<='0';
              rqex_flag<='0';
              end case;
            end if;
        if(t_flag='1' and tn="11") then
          if(t=0) then
stateC<="00";
end if;
case stateC is
when "00" =>
if cyc='1' then
i<=0;
stateC<="01";
end if;
when "01" =>
case stateC1 is
when '0' =>
addr<=std_logic_vector(to_unsigned(i,6));
write1<='0';
write2<='0';
write3<='0';
stateC1<='1';
when '1' =>
if(task1(19)='1') then
  dat1<=task1(19 downto 10) & std_logic_vector("0000000000" - signed(task1(9 downto 0)));
  write1<='1';
  end if;
if(task2(19)='1') then
  dat2<=task2(19 downto 10) & std_logic_vector("0000000000" - signed(task2(9 downto 0)));
  write2<='1';
  end if;
if(task3(19)='1') then
  dat3<=task3(19 downto 10) & std_logic_vector("0000000000" - signed(task3(9 downto 0)));
  write3<='1';
  end if;
stateC1<='0';
i<=i+1;
if(i=64) then
  stateC<="10";
end if;
end case;
when "10" =>
case stateC2 is
when "000" =>
if cyc='1' then
  j<=0;
  stateC2<="001";
  end if;
when "001" =>
  write1<='0';
  write2<='0';
  write3<='0';
  addr<=std_logic_vector(to_unsigned(j,6));
  stateC2<="010";
when "010" =>
  if(task1(19)='1' and (t_signed(task1(9 downto 0))> 45)and(task1(18)='0')) then
    remove<=task1(18 downto 10);
    remove_flag<='1';
    stateC2<="011";
  end if;
  body
else
stateC2<="101";
end if;
when "011"=>

if cyc='1' then
if (ok="11") then
dat1="00000000000000000000";
writ1='1';
add<=task1(18) & "10" & task1(15 downto 10);
add_flag<='1';
dat2<task1;
writ2<='1';
stateC2<="100";
elsif(ok="10")then
stateC2<="100";
end if;
end if;
when "100"=>

if cyc='1' then
j<=j+1;
writ1<='0';
writ2<='0';
writ3<='0';
stateC2<="001";
if(j=64) then
stateC2<="000";
stateC<="11";
end if;
end if;
when others =>
end case;
when "11"=>
case stateC3 is
when "000" =>
if cyc='1' then
k<=0;
stateC3<="001";
end if;
when "001" =>
write1<='0';
write2<='0';
write3<='0';
addr<=std_logic_vector(to_unsigned(k,6));
stateC3<="010";
when "010" =>
if((task2(19)=1)and (t-signed(task2(9 downto 0))> 90)and (task2(18)=0)) then
remove<=task2(18 downto 0);
remove_flag<='1';
stateC3<="011";
else
stateC3<="100";
end if;
when "011" =>

if cyc='1' then
if (ok="11") then
dat2<="00000000000000000000000000000000";
writ2<='1';
add<=task2(18) & "11" & task2(15 downto 10);
add_flag<='1';
dat3<=task2;
writ3<='1';
stateC3<="100";
elsif(ok="10") then
stateC3<="100";
end if;
end if;
when "100"=>
if cyc='1' then
writ1<='0';
writ2<='0';
writ3<='0';
j<=j+1;
stateC3<="001";
if(j=64) then
stateC3<="000";
stateC<="10";
t_flag<='0';
end if;
end if;
when others=>
end case;
end case;
end if;
end if;
end process think;

timer: process (clk)
begin
if(clk'event and clk='1') then
dummy<=dummy+1;
if dummy=50000000 then -- whatever the clock frequency is
t<=t+1;
dummy<=0;
if(t=511) then
t<=0;
end if;
end if;
end if;
end process timer;

turn: process (clk)
begin
if(clk'event and clk='1') then
    clock_count<=clock_count+1;
    if(clock_count=2) then
        clock_count<=0;
        cyc<='0';
    if(tn="01")then
        tn<="10";
        elsif (tn="10") then
        tn<="11";
        elsif(tn="11") then
        tn<="01";
        end if;
    elsif(clock_count=1) then
        cyc<='1';
    end if;
end if;
end process turn;
end if;
end process turn;
end architecture one;
Appendix 2: VHDL Code (Elevator Controller)

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

entity elevatorController is
    port(
        clk, reset: in std_logic; -- clock
        add_flag, remove_flag: in std_logic; -- signal new floor to be added
        add, remove: in std_logic_vector(8 downto 0); -- contains the information on elevator allocation
        lightSensor1, lightSensor2, lightSensor3 : in std_logic_vector(4 downto 0); -- sensor from each of the 3 elevators
        stopButton: in std_logic; -- button to stop the elevator
        doorClosed1, doorClosed2, doorClosed3 : in std_logic; -- Signal to tell whether the door is closed or open
        move1, move2, move3: out std_logic := 'Z'; -- control movement of the 3 elevators. '1' is up, '0' is down, 'Z' is no movement
        atFloor1, atFloor2, atFloor3: out std_logic; -- signal to the door controller to open for each elevator
        arrived: out std_logic_vector (8 downto 0); -- information on the floor where we arrived for the dispatcher
        arrivedPulse: out std_logic; -- a pulse sent to the dispatcher when new information is send on the arrived line
        elevatorDisplay1, elevatorDisplay2, elevatorDisplay3 : out std_logic_vector (4 downto 0); -- lines to elevator displays for each elevator
        setAlarm : out std_logic; -- signal to the alarm
        ok : out std_logic_vector (1 downto 0); -- when a requested change from the dispatcher has been made, this line equals 1.
        lightButtonController: out std_logic_vector (8 downto 0); -- Send to the button controller the floor where we arrived
        lightPulse: out std_logic);
    end elevatorController;

architecture arch of elevatorController is

    type state_type is (up, down); -- The state of the elevators (currently going up or down)
    signal state1, state2, state3 : state_type; -- States for each elevators

    signal elevator1Floor, elevator2Floor, elevator3Floor : std_logic_vector (19 downto 0) := "ZZZZZZZZZZZZZZZZZZZZZ"; --index=floor, element in vector = current direction requested from that floor '1'=up, '0'=down, 'Z'=nothing

    signal currentFloor1, currentFloor2, currentFloor3 : std_logic_vector (4 downto 0) := "00000"; -- The floor where the elevator is leaving from

    signal nextFloor1, nextFloor2, nextFloor3 : std_logic_vector (4 downto 0) := "00000"; -- Stores the next floor that each elevator has to go to

    begin

    end elevatorController;
-- the next direction where the elevator has to go
signal isMoving1, isMoving2, isMoving3 : std_logic := '0';
-- To store whether the elevator is currently moving or not
signal pulseCounterTaken : std_logic_vector (1 downto 0) :="00";
-- Store which elevator is currently using the arrived line
signal pulseCount : std_logic_vector (3 downto 0) := "0000";
-- Number of clock pulses before next use of the line
signal timer1, timer2, timer3: integer;
-- Stores the time in the timer for each elevator
signal startTiming1, startTiming2, startTiming3 : std_logic := '0';
-- Flags to start timing
begin
  elevatorDisplay1 <= lightSensor1;
  -- Constantly update the elevator displays with the light sensor readings
  elevatorDisplay2 <= lightSensor2;
elevatorDisplay3 <= lightSensor3;
  -- Timers. After 5 minutes (timer1,2 and 3 can be reset to change the time) will ring the alarm
  -- Alarm goes off if it takes more than 10 minutes to go from current floor to next floor, or if the stop button has been pressed for more than 5 minutes
  timerr: process (clk)
  begin
    if (clk'event and clk='1' and startTiming1 = '1') then
      timer1<= timer1+1;
      if (timer1 = 1000000000000000000000000000000000000000000000000000) then
        setAlarm <= '1';
      end if;
    end if;
  end process;
  timerr2: process (clk)
  begin
    if (clk'event and clk='1' and startTiming2 = '1') then
      timer2<= timer2+1;
      if (timer2 = 1000000000000000000000000000000000000000000000000000) then
        setAlarm <= '1';
      end if;
    end if;
  end process;
  timerr3: process (clk)
  begin
    if (clk'event and clk='1' and startTiming3 = '1') then
      timer3<= timer3+1;
      if (timer3 = 1000000000000000000000000000000000000000000000000000) then
        setAlarm <= '1';
      end if;
    end if;
  end process;

  -- Chooses the next floor to be accessed, moves the elevator towards the selected floor, updates elevator lists.
  p1: process (clk)
  begin
    ok <= '0';
when resetting, reset all directions to up. Also reset all the current floor to zero.
if reset = '0' then
    state1 <= up;
    state2 <= up;
    state3 <= up;
    currentFloor1<="00000";
    currentFloor2<="00000";
    currentFloor3<="00000";
    -- at each rising clock pulse
    elsif (clk'event and clk='1')
    then

        -- Select next floor for elevator1
        case state1 is
            -- When current state is up.

            when up =>
                if (isMoving1 = '0')
                    then -- if the elevator1 is not currently moving
                        startTiming1 <= '0';
                        -- Reset the timer
                        timer1 <= 0;
                        -- get top most floor
                        for i in currentFloor1 to 19 loop
                            if (elevator1Floor(i) /= Z) then
                                nextFloor1 <= i;
                                nextDirection1 <= elevator1Floor(i);
                                end if;
                            end loop;
                        -- look for any direction that is moving up
                        for j in currentFloor1 to nextFloor1 loop
                            if (elevator1(j)='1') then
                                -- if going up
                                nextFloor1<= j;
                                -- change the next floor to that floor
                                nextDirection1 <= '1';
                                exit;
                                -- exit immediatly
                            end if;
                        end loop;
                        -- At this point, either the next higher floor going up or the top most floor will be accessed next
                        if (currentFloor1 /= nextFloor1)
                            then -- if a higher floor has been selected
                                move1 <= '1';
                                -- send signal to elevator to move up
                                isMoving1 <= '1';
                                -- the elevator is moving
                            end if;
                            -- Once the elevator is moving
else
    startTiming1 <= '1';
    -- start the timer
    if (lightSensor1 = nextFloor1) then
        -- once we arrive to the destination floor
        move1 <= 'Z';
        -- Stop the elevator from moving
        atFloor1 <= '1';
        -- we can open the door
        -- Let the dispatcher know that the elevator has arrived to its destination floor
        if (pulseCounterTaken = "00" or pulseCounterTaken = "01") then
            -- if the line is not used or taken by elevator1
            pulseCounterTaken <= "01";
            -- The line is taken by this elevator
            pulseCount <= pulseCount + 1;
            if (pulseCount = "1010") then
                -- After 10 clock pulses
                arrived <= '0'&"01"&"1"&currentFloor1; -- Send to dispatcher the floor where we arrived
                arrivedPulse <= '1';
                -- send a signal for one clock pulse
                lightButtonController <= '0'&"01"&"1"&currentFloor1;
                lightPulse <= '1';
                pulseCount <= "0000"; --Reset pulse count
                pulseCounterTaken <= "00";
                -- Set the line free
            end if;
        else
            arrivedPulse <= '0';
            lightPulse <= '0';
            end if;
        currentFloor1 <= nextFloor1;
        -- replace the current floor with the next floor
        if (nextDirection1 = '1') then state <= up;
            -- Replace the old direction with the new current one
        elsif (nextDirection1 = '0') then state <= down;
            move1 <= 'Z';
            -- stop moving
            end if;
        if (doorClosed1 = '1') then isMoving1 <= '0'; end if;
        -- Can only move again once the door is closed
        end if;
    end if;
end if;

when down =>
    if (isMoving1 = '0')
        then -- if the elevator1 is not currently moving
            startTiming1 <= '0';
            timer1 <= 0;
        -- get bottom most floor
            for i in currentFloor1 downto 0 loop
                look for bottom most floor
        end if;
    end if;
if (elevator1Floor(i) /= 'Z') then 
nextFloor1 <= i;
nextDirection1 <= elevator1Floor(i);
end if;
end loop;
-- look for any direction that is moving down 
for j in currentFloor1 downto 0 loop 
if (elevator1(j) = '0') then 
-- if going down
nextFloor1 <= j;
-- change the next floor to that floor
nextDirection1 <= '0';
exit;
-- exit immediately
end if;
end loop;
-- At this point, either the next lower floor going down or the bottom most floor will be accessed next
if (currentFloor1 /= nextFloor1) then 
-- if a lower floor has been selected
move1 <= '0';
-- send signal to elevator to move down
isMoving1 <= '1';
-- the elevator is moving
end if;
else 
-- if it is currently moving
startTiming1 <= '1';
if (lightSensor1 = nextFloor1) then 
-- once we arrive to the destination floor
move1 <= 'Z';
-- Stop the elevator from moving
atFloor1 <= '1';
-- we can open the door
if (pulseCounterTaken = "00" or pulseCounterTaken = "01") then 
-- if the line is not used or taken by elevator1
pulseCounterTaken <= "01";
-- The line is taken by this elevator
pulseCount <= pulseCount + 1;
if (pulseCount = "1010") then
-- after at 10 clock pulses
arrived <= '0'&"01"&"0"&currentFloor1; -- Send to dispatcher the floor where we arrived
arrivedPulse <= '1';
-- send a signal for one clock pulse
lightButtonController <= '0'&"01"&"0"&currentFloor1;
lighPulse <= '1';
pulseCount <= "0000"; --Reset pulse count
pulseCounterTaken <= "00";
-- Set the line free
else
arrivedPulse <= '0';
lightPulse <= '0';
end if;
end if;
currentFloor1 <= nextFloor1;
-- replace the current floor with the next floor
if (nextDirection1 = '1') then state<=up;
-- Replace the old direction with the new current one
elsif (nextDirection1 = '0') then state<=down;
move1<='Z';
-- stop moving
end if;
if (doorClosed1 = '1') then
isMoving1 <='0';
atFloor1 <= '0';
end if;
-- Can only move again once the door is closed
end if;
end if;
end if;
end case;
-- Select next floor for elevator2

case state2 is

-- When current state is up.
when up =>
if ( isMoving2 = '0')
then -- if the elevator1 is not currently moving
startTiming2 <='0';
timer1 <= 0;
-- get top most floor
for i in currentFloor2 to 19 loop
if (elevator2Floor(i)='Z') then
nextFloor1 <= i;
nextDirection2 <= elevator2Floor(i);
end if;
end loop;
-- look for any direction that is moving up
for j in currentFloor2 to nextFloor2 loop
if (elevator2(j)='1') then
-- if going up
nextFloor2<= j;
-- change the next floor to that floor
nextDirection2 <= '1';
exit;
-- exit immediately
end if;
end loop;
-- at this point, either the next higher floor going up or the top most floor will be accessed next
if (currentFloor2 /= nextFloor2)
then -- if a higher floor has been selected
move2 <= '1';
-- send signal to elevator to move up
isMoving2 <= '1';
end if;
else
-- if it is currently moving
startTiming2 <= '1';
if (lightSensor2 = nextFloor2) then
-- once we arrive to the destination floor
move2 <= 'Z';
-- Stop the elevator from moving
atFloor2 <= '1';
-- we can open the door
if ( pulseCounterTaken="00" or pulseCounterTaken="10") then
-- if the line is not used or taken by elevator2
pulseCounterTaken <="10";
-- The line is taken by this elevator
pulseCount <= pulseCount +1;
if (pulseCount = "1010") then
arrived <= '0'&"10"&"0"&currentFloor2; -- Send to dispatcher the floor where we arrived
arrivedPulse<='1';
-- send a signal for one clock pulse
lightButtonController <= '0'&"10"&"1"&currentFloor1;
lighPulse<= '1';
pulseCount <= "0000"; --Reset pulse count
pulseCounterTaken <= "00";
-- Set the line free
else
arrivedPulse<='0';
lighPulse <='1';
end if;
end if;
currentFloor2 <= nextFloor2;
-- replace the current floor with the next floor
if ( nextDirection2 = '1') then state<=up;
-- Replace the old direction with the new current one
elsif (nextDirection2 = '0') then state<=down;
end if;
move2<='Z';
-- stop moving
if (doorClosed2 = '1') then isMoving2 <='0'; end if;
-- Can only move again once the door is closed
end if;
end if;
when down =>
if ( isMoving2 = '0')
then -- if the elevator1 is not currently moving
startTiming2 <='0';
timer2 <= '0';
-- get bottom most floor
for i in currentFloor2 downto 0 loop

-- look for bottom most floor
if (elevator2Floor(i)='Z') then
nextFloor2 <= i;
nextDirection2 <= elevator2Floor(i);
end if;
end loop;
-- look for any direction that is moving down
for j in currentFloor2 downto 0 loop
if (elevator2(j)='0') then
-- if going down
nextFloor2 <= j;
-- change the next floor to that floor
nextDirection2 <= '0';
exit;
-- exit immediatly
end if;
end loop;
-- At this point, either the next lower floor going down or the bottom most floor will be accessed next
if (currentFloor2 /= nextFloor2)
then -- if a lower floor has been selected
move2 <= '0';
-- send signal to elevator to move down
isMoving2 <= '1';
-- the elevator is moving
end if;
else
-- if it is currently moving
startTiming2 <= '0';

if (lightSensor2 = nextFloor2) then
-- once we arrive to the destination floor
move2 <= 'Z';
-- Stop the elevator from moving
atFloor2 <= '1';
-- we can open the door

if ( pulseCounterTaken="00" or pulseCounterTaken="10") then
-- if the line is not used or taken by elevator1
pulseCounterTaken <="10";
-- The line is taken by this elevator
pulseCount <= pulseCount +1;
if (pulseCount = "1010")
then -- after at 10 clock pulses
arrived <= '0'&"10"&"0"&currentFloor2; -- Send to dispatcher the floor where we arrived
arrivedPulse<='1';
-- send a signal for one clock pulse
pulseCount <= "0000"; --Reset pulse count
pulseCounterTaken <= "00";
-- Set the line free
else arrivedPulse<='0';
end if;
end if;
currentFloor2 <= nextFloor2;
-- replace the current floor with the next floor
if (nextDirection2 = '1') then state<=up;
-- Replace the old direction with the new current one
elsif (nextDirection2 = '0') then state<=down;
move2<='Z';
-- stop moving
end if;
if (doorClosed2 = '1') then
isMoving2 <='0';
atFloor2 <= '0';
end if;
-- Can only move again once the door is closed
end if;
end if;
end if;
end case;
-- Select next floor for elevator3
case state3 is
-- When current state is up.
when up =>
startTiming3 <=='0';
timer3 <= '0';
if (isMoving3 = '0')
then -- if the elevator1 is not currently moving
-- get top most floor
for i in currentFloor3 to 19 loop
if (elevator3Floor(i)/='Z') then
nextFloor3 <= i;
nexDirection3 <= elevator3(i);
end if;
end loop;
-- look for any direction that is moving up
for j in currentFloor3 to nextFloor3 loop
if (elevator3(j)='1') then
-- if going up
nextFloor3<='0';
-- change the next floor to that floor
nextDirection3 <= '1';
exit;
-- exit immediately
end if;
end loop;
-- At this point, either the next higher floor going up or the top most floor will be accessed next
if (currentFloor3 /= nextFloor3)
then -- if a higher floor has been selected
move3 <= '1';
-- send signal to elevator to move up
isMoving3 <= '1';
end if;
else
-- if it is currently moving
startTiming3 <= '1';
if (lightSensor3 = nextFloor3) then
-- once we arrive to the destination floor
move3 <= 'Z';
-- Stop the elevator from moving
atFloor3 <= '1';
-- we can open the door

if (pulseCounterTaken = "00" or pulseCounterTaken = "11") then
-- if the line is not used or taken by elevator1
pulseCounterTaken <= "11";
-- The line is taken by this elevator
pulseCount <= pulseCount + 1;
if (pulseCount = "1010") then
arrived <= '0'&"01"&"0"&currentFloor3; -- Send to dispatcher the floor where we arrived
arrivedPulse <= '1';
-- send a signal for one clock pulse
lightButtonController <= '0'&"11"&"0"&currentFloor1;
lightPulse <= '1';
pulseCount <= "0000"; --Reset pulse count
pulseCounterTaken <= "00";
-- Set the line free
else
arrivedPulse <= '0';
lightPulse <= '0';
end if;
end if;
currentFloor3 <= nextFloor3;
-- replace the current floor with the next floor
if (nextDirection3 = '1') then state <= up;
-- Replace the old direction with the new current one
elsif (nextDirection3 = '0') then state <= down;
end if;
if (doorClosed3 = '1') then isMoving3 <= '0'; end if;
-- Can only move again once the door is closed
end if;
end if;

when down =>
if (isMoving3 = '0')
then -- if the elevator1 is not currently moving
startTiming3 <= '0';
timer3 <= 0;
-- get bottom most floor
for i in currentFloor3 downto 0 loop
-- look for bottom most floor
if (elevator3Floor(i) /= 'Z') then
nextFloor3 <= i;
end if;
end loop;
else
-- if it is currently moving
startTiming3 <= '1';
if (lightSensor3 = nextFloor3) then
-- once we arrive to the destination floor
move3 <= 'Z';
-- Stop the elevator from moving
atFloor3 <= '1';
-- we can open the door

if (pulseCounterTaken = "00" or pulseCounterTaken = "11") then
-- if the line is not used or taken by elevator1
pulseCounterTaken <= "11";
-- The line is taken by this elevator
pulseCount <= pulseCount + 1;
if (pulseCount = "1010") then
arrived <= '0'&"01"&"0"&currentFloor3; -- Send to dispatcher the floor where we arrived
arrivedPulse <= '1';
-- send a signal for one clock pulse
lightButtonController <= '0'&"11"&"0"&currentFloor1;
lightPulse <= '1';
pulseCount <= "0000"; --Reset pulse count
pulseCounterTaken <= "00";
-- Set the line free
else
arrivedPulse <= '0';
lightPulse <= '0';
end if;
end if;
currentFloor3 <= nextFloor3;
-- replace the current floor with the next floor
if (nextDirection3 = '1') then state <= up;
-- Replace the old direction with the new current one
elsif (nextDirection3 = '0') then state <= down;
end if;
if (doorClosed3 = '1') then isMoving3 <= '0'; end if;
-- Can only move again once the door is closed
end if;
end if;
```plaintext
nextDirection3 <= elevator3(i);
end if;
end loop;
-- look for any direction that is moving down
for j in currentFloor3 downto 0 loop
if (elevator3Floor(j)='0') then
    -- If going down
    nextFloor3<= j;
    -- change the next floor to that floor
    nextDirection3 <= '0';
    exit;
    -- exit immediately
    end if;
end loop;
-- At this point, either the next lower floor going down or the bottom most floor will be accessed next
if (currentFloor3 /= nextFloor3) then
    -- if a lower floor has been selected
    move3 <= '0';
    -- send signal to elevator to move down
    isMoving3 <= '1';
    -- the elevator is moving
    end if;
else
    -- if it is currently moving
    if (lightSensor3 = nextFloor3) then
        -- once we arrive to the destination floor
        move3 <='Z';
        -- Stop the elevator from moving
        atFloor3 <= '1';
        -- we can open the door
        if ( pulseCounterTaken="00" or pulseCounterTaken="01") then
            -- if the line is not used or taken by elevator1
            pulseCounterTaken <="01";
            -- The line is taken by this elevator
            pulseCount <= pulseCount +1;
            if (pulseCount = "1010" )
                then -- after at 10 clock pulses
                    arrived <= '0'&"11"&"0"&currentFloor1; -- Send to dispatcher the floor where we arrived
                    arrivedPulse<='1';
                    -- send a signal for one clock pulse
                    pulseCount <= "0000"; --Reset pulse count
                    pulseCounterTaken <= "00";
                    -- Set the line free
                    else arrivedPulse<='0';
                    end if;
                end if;
        currentFloor1 <= nextFloor1;
        -- replace the current floor with the next floor
        if ( nextDirection1 = '1') then state<=up;
        -- Replace the old direction with the new current one
```
elsif (nextDirection1 = '0') then state<=down;
  end if;
  move1<='Z';
  -- stop moving
  if (doorClosed = '1') then
    isMoving <='0';
    atFloor <= '0';
  end if;
  -- Can only move again once the door is closed
  end if;
end if;
end case;
-- add new data to the elevator matrices
if (add_flag) then
  -- if add flag is on, enter new information in floor for correct elevator
  if (add(7 downto 6) = "01") then
    elevator1Floor( add(4 downto 0)) <= add(5);
  elsif (add(7 downto 6) = "10") then
    elevator2Floor( add(4 downto 0)) <= add(5);
  elsif (add (7 downto 6) = "11") then
    elevator3Floor( add(4 downto 0)) <= add(5);
  end if;
end if;
if (stopButton = '1')
  then
    move <='Z';
  end if;
-- remove data from the elevator matrices
if (remove_flag) then
  if (remove(7 downto 6) = "01") then
    if (nextDirection1/=remove(4 downto 0) or stopButton = '1') then
      -- do not remove if we are asked to remove the next direction
    elevator1Floor( remove(4 downto 0)) <= 'Z';
    ok <= '1';
  end if;
  elsif (remove(7 downto 6) = "10" or stopButton = '1') then
    if (nextDirection2/=remove(4 downto 0)) then
      elevator2Floor( remove(4 downto 0)) <= 'Z';
      ok<= '1';
    end if;
  elsif (add (7 downto 6) = "11" or stopButton = '1') then
    if (currentDirection3/=remove(4 downto 0)) then
      elevator3Floor(remove(4 downto 0)) <= 'Z';
      ok<= '1';
    end if;
  end if;
end if;
end if;
end if;
end if;
end process;
begin
  end arch;
Appendix 3: VHDL Code (Hall Button Controller)

import std_logic from the IEEE library
library ieee;
use ieee.std_logic_1164.all;
entity hallbuttoncontroller is
  port(U1,U2,D2,U3,D3,U4,D4,U5,D5,U6,D6,U7,D7,U8,D8,U9,D9,U10,D10,
       U11,D11,U12,D12,U13,D13,U14,D14,U15,D15,U16,D16,U17,D17,U18,D18,U19,D19,D20:in
       std_logic;
       at_floor_direction: in std_logic_vector(8 downto 0);
       enable: in std_logic;
       new_request:in std_logic;
       clock: in std_logic;
       UL1,UL2,DL2,UL3,DL3,UL4,DL4,UL5,DL5,UL6,DL6,UL7,DL7,UL8,DL8,UL9,DL9,UL10,DL10,
       UL11,DL11,UL12,DL12,UL13,DL13,UL14,DL14,UL15,DL15,UL16,DL16,UL17,DL17,UL18,DL18,
       UL19,DL19,DL20 :out std_logic;

       --countest: out std_logic_vector(0 to 3);
       request_sent: out std_logic;
       floor_direction: out std_logic_vector( 8 downto 0));
end entity hallbuttoncontroller;

architecture buttoncontroller of hallbuttoncontroller is

--Temp signals
signal temp1, temp2, temp4: std_logic;
signal temp3: std_logic_vector(5 downto 0);
signal floortemp_direction: std_logic_vector(5 downto 0);

signal flag,reset: std_logic;
signal count: std_logic_vector(0 to 3);

--require a counter to determine 10 clock cycles.
component counter
  port (Clock_enable: in std_logic;
        Clock: in std_logic;
        Reset: in std_logic;
        Output: out std_logic_vector(0 to 3));
end component;

begin

  G1: counter port map (clock_enable=>flag, clock => clock,reset => reset,output => count);

  --countest<= count;
update: process (enable)
begin
if (enable = '1') then
    if(U1 = '1') then
        floortemp_direction <= "100001";
    elsif (U2 = '1') then
        floortemp_direction <= "100010";
    elsif (D2 = '1') then
        floortemp_direction <= "000010";
    elsif(U3 = '1') then
        floortemp_direction <= "100011";
    elsif(D3 = '1') then
        floortemp_direction <= "000011";
    elsif(U4 = '1') then
        floortemp_direction <= "100100";
    elsif(D4 = '1') then
        floortemp_direction <= "000100";
    elsif(U5 = '1') then
        floortemp_direction <= "100101";
    elsif(D5 = '1') then
        floortemp_direction <= "000101";
    elsif(U6 = '1') then
        floortemp_direction <= "100110";
    elsif(D6 = '1') then
        floortemp_direction <= "000110";
    elsif(U7 = '1') then
        floortemp_direction <= "100111";
    elsif(D7 = '1') then
        floortemp_direction <= "000111";
    elsif(U8 = '1') then
        floortemp_direction <= "101000";
    elsif(D8 = '1') then
        floortemp_direction <= "001000";
    elsif(U9 = '1') then
        floortemp_direction <= "101001";
    elsif(D9 = '1') then
        floortemp_direction <= "001001";
elsif(U10 = '1') then
floortemp_direction <= "101010";
elif (D10 = '1') then
floortemp_direction <= "001010";
elif(U11 = '1') then
floortemp_direction <= "101011";
elif (D11 = '1') then
floortemp_direction <= "001011";
elif(U12 = '1') then
floortemp_direction <= "101100";
elif(D12 = '1') then
floortemp_direction <= "001100";
elif(U13 = '1') then
floortemp_direction <= "101101";
elif(D13 = '1') then
floortemp_direction <= "001101";
elif(U14 = '1') then
floortemp_direction <= "101110";
elif (D14 = '1') then
floortemp_direction <= "001110";
elif(U15 = '1') then
floortemp_direction <= "101111";
elif(D15 = '1') then
floortemp_direction <= "001111";
elif(U16 = '1') then
floortemp_direction <= "110000";
elif(D16 = '1') then
floortemp_direction <= "010000";
elif(U17 = '1') then
floortemp_direction <= "110001";
elif(D17 = '1') then
floortemp_direction <= "010001";
elif(U18 = '1') then
floortemp_direction <= "110010";
elif(D18 = '1') then
floortemp_direction <= "010010";
elif(U19 = '1') then
floortemp_direction <= "110011";
elif(D19 = '1') then
floortemp_direction <= "010011";
elif(D20 = '1') then
floortemp_direction <= "110100";
else
floortemp_direction <= "000000";
end if;
end if;
end process update;

control: process (floortemp_direction,new_request)
begin

request_sent<= '0';
if ( count = "1001") then
floor_direction <= "000" & (floortemp_direction);
end if;
if(count ="1001" AND new_request ='0' ) then
if(floortemp_direction = "100001") then
UL1 <= '1';
request_sent<= '1';
elsif (floortemp_direction = "100010") then
UL2 <= '1';
request_sent<= '1';
elsif (floortemp_direction = "000010") then
DL2 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "100011") then
UL3 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "000011") then
DL3 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "100100") then
UL4 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "000100") then
DL4 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "100101") then
UL5 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "000101") then
DL4 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "100110") then
UL6 <= '1';
request_sent<= '1';
elsif (floortemp_direction = "000110")then
DL6 <= '1';
request_sent<= '1';
elsif(floortemp_direction = "100111") then

end if;
UL7 <= '1';
request_sent<= '1';
elif(floortemp_direction = "000111")then
    DL7 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101000") then
    UL8 <= '1';
    request_sent<= '1';
elif (floortemp_direction = "001000") then
    DL8 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101001") then
    UL9 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "001001") then
    DL9 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101010") then
    UL10 <= '1';
    request_sent<= '1';
elif (floortemp_direction = "001100") then
    DL10 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101100") then
    UL11 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "001101") then
    DL11 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101101") then
    UL12 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "001110") then
    DL12 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101110") then
    UL13 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "001111") then
    DL13 <= '1';
    request_sent<= '1';
elif(floortemp_direction = "101111") then
    UL14 <= '1';
    request_sent<= '1';
elif (floortemp_direction = "001110") then
    DL14 <= '1';

request_sent<= '1';
elif(floortemp_direction = "101111") then
  UL15 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "001111") then
  DL15 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "110000") then
  UL16 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "010000") then
  DL16 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "110001") then
  UL17 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "010001") then
  DL17 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "110010") then
  UL18 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "010010") then
  DL18 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "110011") then
  UL19 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "010011") then
  DL19 <= '1';
  request_sent<= '1';
elif(floortemp_direction = "110100") then
  DL20 <= '1';
  request_sent<= '1';
end if;

end if;

temp3 <= at_floor_direction(5)& at_floor_direction(4)& at_floor_direction(3)& at_floor_direction(2)& at_floor_direction(1) & at_floor_direction(0);
--Set to zero when request is meet causing the light to be turned off
if (new_request = '1') then
  if(temp3 = "100001") then
    \(\)
UL1 <= '0';
elsif (temp3 = "100010") then
    UL2 <= '0';
    elsif (temp3 = "000010") then
        DL2 <= '0';
    elsif(temp3 = "100011") then
        UL3 <= '0';
    elsif( temp3 ="000011") then
        DL3 <= '0';
    elsif(temp3 = "100100") then
        UL4 <= '0';
    elsif(temp3 ="000100") then
        DL4 <= '0';
    elsif(temp3 = "100101") then
        UL5 <= '0';
    elsif(temp3 = "000101") then
        DL5 <= '0';
    elsif(temp3 = "100110") then
        UL6 <= '0';
    elsif(temp3 = "000110")then
        DL6 <= '0';
    elsif(temp3 = "100111") then
        UL7 <= '0';
    elsif(temp3 = "000111")then
        DL7 <= '0';
    elsif(temp3 = "101000") then
        UL8 <= '0';
    elsif (temp3 = "001000") then
        DL8 <= '0';
    elsif(temp3 = "101001") then
        UL9 <= '0';
    elsif(temp3 = "001001") then
        DL9 <= '0';
    elsif(temp3 = "101010") then
        UL10 <= '0';
    elsif (temp3 = "001010") then
        DL10 <= '0';
    elsif(temp3 = "101011") then
        UL11 <= '0';
    elsif(temp3 = "001011") then
DL11 <= '0';
elsif(temp3 = "101100") then
  UL12 <= '0';
elsif(temp3 = "001100") then
  DL12 <= '0';
elsif(temp3 = "101101") then
  UL13 <= '0';
elsif(temp3 = "001101") then
  DL13 <= '0';
elsif(temp3 = "101110") then
  UL14 <= '0';
elsif(temp3 = "001110") then
  DL14 <= '0';
elsif(temp3 = "101111") then
  UL15 <= '0';
elsif(temp3 = "001111") then
  DL15 <= '0';
elsif(temp3 = "110000") then
  UL16 <= '0';
elsif(temp3 = "010000") then
  DL16 <= '0';
elsif(temp3 = "110001") then
  UL17 <= '0';
elsif(temp3 = "010001") then
  DL17 <= '0';
elsif(temp3 = "110010") then
  DL20 <= '0';
else
  end if;
end if;
end process control;

dl buttoncontroller;
Appendix 4: VHDL Code (Door Controller)

```vhdl
--import std_logic from the IEEE library
library ieee;
use ieee.std_logic_1164.all;

entity door_controller is
  port(
    weight: in std_logic_vector(10 downto 0);  --threshold 2000 kg (11111010000)
    at_floor: in std_logic;
    motion_sensor, clock: in std_logic;
    time_isup, door_status: in std_logic;
    countest: out std_logic_vector(0 to 3);

    start_timer, alarm, door, ready_to_go: out std_logic);
end entity door_controller;

architecture doorcontrol of door_controller is

signal flag: std_logic;
signal done_withfloor, opened_door: std_logic;
signal count: std_logic_vector(0 to 3);

component counter
  port (Clock_enable: in std_logic;
        Clock: in std_logic;
        Reset: in std_logic;
        Output: out std_logic_vector(0 to 3));
end component;

begin

G1: counter port map (clock_enable=>'0', clock => clock, reset => at_floor, output => count);
  countest <= count;

mux: process(at_floor, clock, door_status, time_isup)
begin
  done_withfloor <= '0';
  start_timer <= '0';
  if (time_isup = '1' AND door_status = '1') then --door is open and time is up then ring alarm
    alarm <= '1';
  else
    alarm <= '0';
  end if;

  if (at_floor = '1' and door_status = '0') then --if door is closed and at floor and stopped moving
    start_timer <= '1';
    if (count < "0101") then
      door <= '1';  --keep door open for 5 clock cycles
    end if;

  elsif((clock = '1') and((weight >= "11111010000") OR motion_sensor = '1')) then
    door <= '1';
  end if;

  if (count >= "0101") then
    door <= '0';
  end if;

end process

if (count >= "0101") then
  door <= '0';
END IF;
```

```
done_withfloor <= '1';
--open and closed doors so done with
END IF;

ready_to_go <= '1';
END process mux;

End doorcontrol;