|  |  |  |
| --- | --- | --- |
| 1 | // | Alan Li |
| 2 | // | 01/15/2022 |
| 3 | // | EE 371 |
| 4 | // | Lab #1 |
| 5 |  |  |
| 6 | // | counter takes 1-bit enter and exit as inputs and return 5-bit cout |
|  | as | output. |
| 7 | // | The module functions as its name. When there is an enter signal, the |

cout increse by 1. When there is an exit signal, the cout decrease by 1.

8 // The range for the counter is from 0 to 25. 9

10 module counter(clk, reset, enter, exit, cout); 11

1. input logic clk, reset, enter, exit;
2. output logic [4:0] cout;
3. logic [4:0] ps, ns; 15
4. // Each decimal from 0 to 25 has been assigned with a 5-bit binary number.
5. parameter [4:0] zero = 5'b0, 18 one = 5'b1,

19 two = 5'b10,

20 three = 5'b11,

21 four = 5'b100,

22 five = 5'b101,

23 six = 5'b110,

24 seven = 5'b111,

25 eight = 5'b1000,

26 nine = 5'b1001,

27 ten = 5'b1010,

28 eleven = 5'b1011,

29 twelve = 5'b1100,

1. thirteen = 5'b1101,
2. fourteen = 5'b1110,
3. fifteen = 5'b1111,

33 sixteen = 5'b10000,

1. seventeen = 5'b10001,
2. eighteen = 5'b10010,
3. nineteen = 5'b10011,

37 twenty = 5'b10100,

1. twentyone = 5'b10101,
2. twentytwo = 5'b10110,
3. twentythree = 5'b10111,
4. twentyfour = 5'b11000,
5. twentyfive = 5'b11001; 43
6. // 25 states for the counter. Each state represent a different number.
7. // When there is a enter signal, the counter goes to next state which the number increase by one and vice versa.
8. // At state 0, if there is another exit signal(which should not happen in reality), the state will stay at zero.
9. // At state 25, if there is another enter signal, the state will stay at 25.

|  |  |  |
| --- | --- | --- |
| 48 | case(ps) |  |
| 49 | zero: if(enter) | ns = one; |
| 50 | else | ns = zero; |
| 51 | one: if(enter) | ns = two; |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 52 | else if(exit) | ns | = | zero; |  |
| 53 | else | ns | = | one; |
| 54 | two: if(enter) | ns | = | three; |
| 55 | else if(exit) | ns | = | one; |
| 56 | else | ns | = | two; |
| 57 | three:if(enter) | ns | = | four; |
| 58 | else if(exit) | ns | = | two; |
| 59 | else | ns | = | three; |
| 60 | four: if(enter) | ns | = | five; |
| 61 | else if(exit) | ns | = | three; |
| 62 | else | ns | = | four; |
| 63 |  |  |  |  |
| 64 | /\* |  |  |  |
| 65 | five: if(exit) | ns | = | four; |
| 66 | else | ns | = | five; // | for |
| 67 | \*/ |  |  |  |  |
| 68 |  |  |  |  |  |
| 69 |  |  |  |  |  |
| 70 | five: if(enter) | ns | = | six; |  |
| 71 | else if(exit) | ns | = | four; |  |
| 72 | else | ns | = | five; // | for |
| 73 |  |  |  |  |  |
| 74 | six: if(enter) | ns | = | seven; | |
| 75 | else if(exit) | ns | = | five; | |
| 76 | else | ns | = | six; | |
| 77 | seven: if(enter) | ns | = | eight; | |
| 78 | else if(exit) | ns | = | six; | |
| 79 | else | ns | = | seven; | |
| 80 | eight:if(enter) | ns | = | nine; | |
| 81 | else if(exit) | ns | = | seven; | |
| 82 | else | ns | = | eight; | |
| 83 | nine: if(enter) | ns | = | ten; | |
| 84 | else if(exit) | ns | = | eight; | |
| 85 | else | ns | = | nine; | |
| 86 | ten: if(enter) | ns | = | eleven; | |
| 87 | else if(exit) | ns | = | nine; | |
| 88 | else | ns | = | ten; | |
| 89 | eleven:if(enter) | ns | = | twelve; | |
| 90 | else if(exit) | ns | = | ten; | |
| 91 | else | ns | = | eleven; | |
| 92 | twelve:if(enter) | ns | = | thirteen; | |
| 93 | else if(exit) | ns | = | eleven; | |
| 94 | else | ns | = | twelve; | |
| 95 | thirteen:if(enter) | ns | = | fourteen; | |
| 96 | else if(exit) | ns | = | twelve; | |
| 97 | else | ns | = | thirteen; | |
| 98 | fourteen: if(enter) | ns | = | fifteen; | |
| 99 | else if(exit) | ns | = | thirteen; | |
| 100 | else | ns | = | fourteen; | |
| 101 | fifteen:if(enter) | ns | = | sixteen; | |
| 102 | else if(exit) | ns | = | fourteen; | |
| 103 | else | ns | = | fifteen; | |
| 104 | sixteen: if(enter) | ns | = | seventeen; | |
| 105 | else if(exit) | ns | = | fifteen; | |
| 106 | else | ns | = | sixteen; | |
| 107 | seventeen:if(enter) | ns | = | eighteen; | |
| 108 | else if(exit) | ns | = | sixteen; | |
| 109 | else | ns | = | seventeen; | |

demo purpose

lab design purpose

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 110 |  | eighteen:if(enter) | ns | = | nineteen; |
| 111 |  | else if(exit) | ns | = | seventeen; |
| 112 |  | else | ns | = | eighteen; |
| 113 |  | nineteen:if(enter) | ns | = | twenty; |
| 114 |  | else if(exit) | ns | = | eighteen; |
| 115 |  | else | ns | = | nineteen; |
| 116 |  | twenty:if(enter) | ns | = | twentyone; |
| 117 |  | else if(exit) | ns | = | nineteen; |
| 118 |  | else | ns | = | twenty; |
| 119 |  | twentyone:if(enter) | ns | = | twentytwo; |
| 120 |  | else if(exit) | ns | = | twenty; |
| 121 |  | else | ns | = | twentyone; |
| 122 |  | twentytwo:if(enter) | ns | = | twentythree; |
| 123 |  | else if(exit) | ns | = | twentyone; |
| 124 |  | else | ns | = | twentytwo; |
| 125 |  | twentythree:if(enter) | ns | = | twentyfour; |
| 126 |  | else if(exit) | ns | = | twentytwo; |
| 127 |  | else | ns | = | twentythree; |
| 128 |  | twentyfour:if(enter) | ns | = | twentyfive; |
| 129 |  | else if(exit) | ns | = | twentythree; |
| 130 |  | else | ns | = | twentyfour; |
| 131 |  | twentyfive:if(exit) | ns | = | twentyfour; |
| 132 |  | else | ns | = | twentyfive; |
| 133 |  | endcase |  |  |  |

134 end 135

1. // When reset is pressed, the counter will reset to state 0.
2. always @(posedge clk) begin
3. if(reset) begin
4. ps <= zero;
5. end
6. else begin
7. cout <= ps;
8. ps <= ns;
9. end
10. end
11. endmodule 147
12. // counter\_testbench tests all expected behavior that the parking lot occupancy counter system in the lab may encounter
13. module counter\_testbench();
14. logic clk, reset, enter, exit;
15. logic [4:0]cout; 152

153 counter dut (.clk(clk), .reset(reset), .enter(enter), .exit(exit), . cout(cout));

154

155 parameter CLOCK\_PERIOD = 100; 156

157 initial begin

158 clk <= 0;

1. forever #(CLOCK\_PERIOD/2) clk <= ~clk;
2. end 161
3. // 30 cars enters, the counter will reach 25 and stay there
4. // 30 cars exit(for simulation), the counter will reach 0 and stay there
5. initial begin

|  |  |  |
| --- | --- | --- |
| 165 | reset <= 1; | @(posedge clk); |
| 166 | reset <= 0; | @(posedge clk); |
| 167 | enter <= 1; | exit <= 0; repeat(30) @(posedge clk); |
| 168 | enter <= 0; | exit <= 1; repeat(30) @(posedge clk); |
| 169 | $stop; |  |
| 170 | end |  |
| 171 | endmodule |  |
| 172 |  |  |
| 173 |  |  |
| 174 |  |  |

|  |  |  |
| --- | --- | --- |
| 1 | // | Alan Li |
| 2 | // | 01/15/2022 |
| 3 | // | EE 371 |
| 4 | // | Lab #1 |
| 5 |  |  |
| 6 | // | DE1\_Soc takes 34-bit GPIO\_0 and return 1-bit enterIndicator and |

exitIndicator, 5-bit countIndicator and inputA, inputB

7 // This serves as the top-level module for the parking lot occupancy counter system

8

9 module DE1\_SoC(CLOCK\_50, GPIO\_0, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0); 10

1. // GPIO\_0[5] is connected to the reset switch, GPIO\_0[6] is connected to the sensorA switch, GPIO\_0[7] is connected to the sensorB switch
2. inout logic [33:0] GPIO\_0;
3. input logic CLOCK\_50;
4. output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
5. logic enterIndicator, exitIndicator;
6. logic [4:0] countIndicator;
7. logic inputA, inputB; 18

19

1. assign GPIO\_0[26] = GPIO\_0[6]; // when switch A(sensor A) is enabled/triggered the corresponding LED turns on
2. assign GPIO\_0[27] = GPIO\_0[7]; // when switch B(sensor B) is enabled/triggered the corresponding LED turns on

22

1. // userInput inA and inB takes GPIO[6] and GPIO[7] as input parameters and return Q as inputA and inputB respectively
2. userInput inA (.clk(CLOCK\_50), .D(GPIO\_0[6]), .Q(inputA));
3. userInput inB (.clk(CLOCK\_50), .D(GPIO\_0[7]), .Q(inputB)); 26
4. // parkingLotSensors myfsm takes CLOCK\_50 as clk, GPIO[5], inputA, input B as parameters to reset, A, B
5. // and returns enter and exit as enterIndicator and exitIndicator respectively
6. parkingLotSensors myfsm (.clk(CLOCK\_50), .reset(GPIO\_0[5]), .A(inputA

), .B(inputB), .enter(enterIndicator), .exit(exitIndicator));

30

1. // counter mycounter takes CLOCK\_50 as clk, GPIO\_0[5], enterIndicator and exitIndicator as reset, enter and exit
2. // it returns enter, exit and cout as countIndicator[4:0]
3. counter mycounter (.clk(CLOCK\_50), .reset(GPIO\_0[5]), .enter( enterIndicator), .exit(exitIndicator), .cout(countIndicator[4:0]));

34

1. // hexDisplay mydisplay takes CLOCK\_50 as clk, countIndicator[4:0] as inputCount
2. // and returns HEX5, HEX4, HEX3, HEX2, HEX1, HEX0 as HEX5, HEX4, HEX3, HEX2, HEX1, HEX0 respectively
3. hexDisplay mydisplay (.clk(CLOCK\_50), .inputCount(countIndicator[4:0

]),

1. .HEX5(HEX5), .HEX4(HEX4), .HEX3(HEX3), .HEX2( HEX2), .HEX1(HEX1), .HEX0(HEX0));
2. endmodule 40

41

42 // DE1\_SoC\_testbench tests all expected behavior that the parking lot

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 43 | occupancy counter system in the lab may encounter  module DE1\_SoC\_testbench(); | | | | | |  |
| 44 |  | | | | | |
| 45 | logic clk, reset, A, B; | | | | | |
| 46 | logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0; | | | | | |
| 47 |  | | | | | |
| 48 | DE1\_SoC dut (.CLOCK\_50(CLOCK\_50), .GPIO\_0(GPIO\_0), | | | | | |
| 49  50 | .HEX5(HEX5), .HEX4(HEX4), .HEX3(HEX3),  HEX2), .HEX1(HEX1), .HEX0(HEX0)); | | | | | | .HEX2( |
| 51 | //Set up the clock. | | | | | |  |
| 52 | parameter CLOCK\_PERIOD = 100; | | | | | |  |
| 53 |  | | | | | |  |
| 54 | initial begin | | | | | |  |
| 55 | clk <= 0; | | | | | |  |
| 56 | forever #(CLOCK\_PERIOD/2) clk <= ~clk; | | | | | |  |
| 57 |  | | | | | |  |
| 58 | end | | | | | |  |
| 59 |  | | | | | |  |
| 60 | initial begin | | | | | |  |
| 61 | reset | <= | 1; | @(posedge | clk); | // cycle through car enteri | |
| 62 | reset | <= | 0; | @(posedge | clk); |  | |
| 63 |  |  |  | @(posedge | clk); |  | |
| 64 |  |  |  | @(posedge | clk); |  | |
| 65 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 66 | {A,B} | <= | 2'b10; | @(posedge | clk); |  | |
| 67 | {A,B} | <= | 2'b11; | @(posedge | clk); |  | |
| 68 | {A,B} | <= | 2'b01; | @(posedge | clk); |  | |
| 69 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 70 |  |  |  |  |  |  | |
| 71 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 72 | {A,B} | <= | 2'b10; | @(posedge | clk); |  | |
| 73 | {A,B} | <= | 2'b11; | @(posedge | clk); |  | |
| 74 | {A,B} | <= | 2'b01; | @(posedge | clk); |  | |
| 75 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 76 |  |  |  |  |  |  | |
| 77 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 78 | {A,B} | <= | 2'b10; | @(posedge | clk); |  | |
| 79 | {A,B} | <= | 2'b11; | @(posedge | clk); |  | |
| 80 | {A,B} | <= | 2'b01; | @(posedge | clk); |  | |
| 81 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 82 |  |  |  |  |  |  | |
| 83 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 84 | {A,B} | <= | 2'b10; | @(posedge | clk); |  | |
| 85 | {A,B} | <= | 2'b11; | @(posedge | clk); |  | |
| 86 | {A,B} | <= | 2'b01; | @(posedge | clk); |  | |
| 87 | {A,B} | <= | 2'b00; | @(posedge | clk); | // 4 cars entered | |
| 88 |  |  |  |  |  |  | |
| 89 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 90 | {A,B} | <= | 2'b01; | @(posedge | clk); |  | |
| 91 | {A,B} | <= | 2'b11; | @(posedge | clk); |  | |
| 92 | {A,B} | <= | 2'b10; | @(posedge | clk); |  | |
| 93 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 94 |  |  |  |  |  |  | |
| 95 | {A,B} | <= | 2'b00; | @(posedge | clk); |  | |
| 96 | {A,B} | <= | 2'b01; | @(posedge | clk); |  | |
| 97 | {A,B} | <= | 2'b11; | @(posedge | clk); |  | |
| 98 | {A,B} | <= | 2'b10; | @(posedge | clk); |  | |

ng

99 {A,B} <= 2'b00; @(posedge clk); 100

101 {A,B} <= 2'b00; @(posedge clk);

102 {A,B} <= 2'b01; @(posedge clk);

103 {A,B} <= 2'b11; @(posedge clk);

104 {A,B} <= 2'b10; @(posedge clk);

105 {A,B} <= 2'b00; @(posedge clk); // 3 cars exiting 106

1. reset <= 1; @(posedge clk); // cycle through car exiting
2. reset <= 0; @(posedge clk);
3. @(posedge clk);
4. @(posedge clk);

111 {A,B} <= 2'b00; @(posedge clk);

112 {A,B} <= 2'b01; @(posedge clk);

113 {A,B} <= 2'b11; @(posedge clk);

114 {A,B} <= 2'b10; @(posedge clk);

115 {A,B} <= 2'b00; @(posedge clk);

1. @(posedge clk);
2. $stop;
3. end
4. endmodule 120

|  |  |  |
| --- | --- | --- |
| 1 | // | Alan Li |
| 2 | // | 01/15/2022 |
| 3 | // | EE 371 |
| 4 | // | Lab #1 |
| 5 |  |  |
| 6 | // | hexDisplay takes 5-bit inputCount as inputs and return 7-bits HEX5, |

HEX4, HEX3, HEX2, HEX1, HEX0 as output.

* 1. // Upon start, the HEX display will display "clear0", when enter signal is given, the HEX display the counter on HEX1 and HEX0. The counter will keep adding up until reach 25.
  2. // Then the hex display will diaplay "FULL25".
  3. // If the counter reach 0, the hex display will display "clear0" as there is no car in the parking lot

10

1. module hexDisplay(clk, inputCount, HEX5, HEX4, HEX3, HEX2,HEX1, HEX0);
2. input clk;
3. input logic [4:0] inputCount;
4. output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0; 15

16 // 7-bit parameter for the display to display different numbers or characters

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 17 | // |  |  | 6543210 |  |
| 18 | parameter | [6:0] | zero = | 7'b1000000, |
| 19 |  |  | one = | 7'b1111001, |
| 20 |  |  | two = | 7'b0100100, |
| 21 |  |  | three = | 7'b0110000, |
| 22 |  |  | four = | 7'b0011001, |
| 23 |  |  | five = | 7'b0010010, |
| 24 |  |  | six = | 7'b0000010, |
| 25 |  |  | seven = | 7'b1111000, |
| 26 |  |  | eight = | 7'b0000000, |
| 27 |  |  | nine = | 7'b0011000, |
| 28 |  |  | F = | 7'b0001110, | //FULL |
| 29 |  |  | U = | 7'b1000001, |  |
| 30 |  |  | L = | 7'b1000111, |  |
| 31 |  |  |  |  |  |
| 32 |  |  | C = | 7'b1000110, | //CLEAR |
| 33 |  |  | E = | 7'b0000110, |  |
| 34 |  |  | A = | 7'b0001000, |  |
| 35 |  |  | R = | 7'b1001100, |  |

36 blk = 7'b1111111;

37

38

1. // The hex are driven off upon start.
2. // The hex will display counters when enter or exit signal is given
3. // When the counter is 0, it will display "clear0". When full it will display "full25"
4. always\_comb begin
5. HEX5 = blk;
6. HEX4 = blk;
7. HEX3 = blk;
8. HEX2 = blk;
9. HEX1 = blk;
10. HEX0 = blk;
11. case(inputCount)
12. 0: begin HEX5 = C; HEX4 = L; HEX3 = E; HEX2 = A; HEX1 = R; HEX0

= zero; end

1. 1: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =

blk; HEX0 = one; end

1. 2: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = two; end
2. 3: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = three; end
3. 4: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk;HEX0 = four; end
4. 5: begin HEX5 = F; HEX4 = U; HEX3 = L; HEX2 = L;HEX1 = blk; HEX0 = five; end // for demo purpose

56

57 //5: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk;HEX0 = five; end

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 58  59 | 6:  blk;HEX0 =  7: | | begin HEX5 = blk; six; end  begin HEX5 = blk; | | | HEX4  HEX4 | =  = | blk;  blk; | HEX3  HEX3 | =  = | blk;  blk; | HEX2  HEX2 | =  = | blk;  blk; | HEX1  HEX1 | =  = |
| 60 | blk;HEX0 =  8: | | seven; end  begin HEX5 = blk; | | | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk; | HEX1 | = |
| 61 | blk;HEX0 =  9: | | eight; end  begin HEX5 = blk; | | | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk; | HEX1 | = |
|  | blk;HEX0 = | | nine; end | | |  |  |  |  |  |  |  |  |  |  |  |
| 62  63 | one; | 10: HEX0 =  11: | begin HEX5  zero; end begin HEX5 | =  = | blk;  blk; | HEX4  HEX4 | =  = | blk;  blk; | HEX3  HEX3 | =  = | blk;  blk; | HEX2  HEX2 | =  = | blk;HEX1  blk;HEX1 | | =  = |
| 64 | one; | HEX0 =  12: | one; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 65 | one; | HEX0 =  13: | two; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 66 | one; | HEX0 =  14: | three; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 67 | one; | HEX0 =  15: | four; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 68 | one; | HEX0 =  16: | five; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 69 | one; | HEX0 =  17: | six; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 70 | one; | HEX0 =  18: | seven; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 71 | one; | HEX0 =  19: | eight; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 72 | one; | HEX0 =  20: | nine; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 73 | two; | HEX0 =  21: | zero; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 74 | two; | HEX0 =  22: | one; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 75 | two; | HEX0 =  23: | two; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
| 76 | two; | HEX0 =  24: | three; end  begin HEX5 | = | blk; | HEX4 | = | blk; | HEX3 | = | blk; | HEX2 | = | blk;HEX1 | | = |
|  | two; | HEX0 = | four; end |  |  |  |  |  |  |  |  |  |  |  | |  |
| 77 | 25:  HEX0 = five; | | begin HEX5  end | = | F; HEX4 = | | U; HEX3 = L; HEX2 = L;HEX1 = two; | | | | | | | | | |

1. endcase
2. end

80

81 endmodule 82

1. // counter\_testbench tests all expected behavior that the parking lot

occupancy counter system in the lab may encounter

1. module hexDisplay\_testbench();
2. logic clk;
3. logic [4:0] inputCount;
4. logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0; 88

89 hexDisplay dut (.clk(clk), .inputCount(inputCount), .HEX5(HEX5), . HEX4(HEX4), .HEX3(HEX3), .HEX2(HEX2), .HEX1(HEX1), .HEX0(HEX0));

90

91 parameter CLOCK\_PERIOD = 100; 92

93 initial begin

94 clk <= 0;

1. forever #(CLOCK\_PERIOD/2) clk <= ~clk;
2. end

97

1. initial begin
2. inputCount <= 0; @(posedge clk);
3. inputCount <= 10; @(posedge clk);
4. inputCount <= 20; @(posedge clk);
5. inputCount <= 25; @(posedge clk);
6. $stop;
7. end
8. endmodule 106

|  |  |  |
| --- | --- | --- |
| 1 | // | Alan Li |
| 2 | // | 01/15/2022 |
| 3 | // | EE 371 |
| 4 | // | Lab #1 |
| 5 |  |  |
| 6 | // | parkingLotSensors takes in 1-bit A, B and clk as inputs and return |

1-bit enter and exit as outputs.

7

1. module parkingLotSensors(clk, reset, A, B, enter, exit);
2. input logic A, B, clk, reset;
3. output logic enter, exit; 11
4. // States
5. enum {unblocked, beginIn, blockIn, almostIn, beginOut, blockOut, almostOut} ps, ns;

14

1. // The parking lot sensor can have six cases as list below. beginIn, blockin and almostIn are pairs with almostOut, blockOut and beginOut.
2. // Assume the entrance is north-south direction. If car is entering from North, it will trigger sensor A first.
3. // If car is exiting from South it will trigger sensor B first.
4. // For example, beginIn and almost out both indicate that only sensorA is blocked, they only differences is that beginIn means the car is going in the south direction and almostOut is car going north.
5. // This "case pair" is designed to tackle car changing direction issue.
6. always\_comb
7. begin
8. enter = 0;
9. exit = 0;

24

25 case(ps)

26

1. unblocked:
2. if (A & ~B) ns = beginIn; // car begin to enter
3. else if (~A & B) ns = beginOut; // car begin to exit
4. // else if (~A & ~B) ns = unblocked; // impossible
5. else ns = unblocked; // nothing happens 32
6. beginIn:
7. if (A & B) ns = blockIn; // car is halfway entering
8. else if (~A & ~B) ns = unblocked; // car just enters then back up
9. // else if (A & ~B) ns = beginIn; // impossible
10. else ns = beginIn; // car does not move

38

1. blockIn:
2. if (~A & B) ns = almostIn; // car almost enters (trigger both sensors)
3. else if (A & ~B) ns = beginIn; // car backs up
4. // else if (A & B) ns = blockIn; // impossible
5. else ns = blockIn; // car does not move

44

45 almostIn:

46 if (~A & ~B)

1. begin
2. ns = unblocked; // car enters
3. enter = 1;
4. exit = 0;
5. end
6. else if (A % B) ns = blockIn; // car backs up
7. // else if (~A & B) ns = almostIn; // impossible
8. else ns = almostIn; // car does not move

55

1. beginOut:
2. if (A & B) ns = blockOut; // car is halfway exiting
3. else if (~A & ~B) ns = unblocked; // car backs up
4. // else if (~A & B) ns = beginOut; // impossible
5. else ns = beginOut; // car does not move

61

1. blockOut:
2. if (A & ~B) ns = almostOut; // car almost exits (trigger both sensors)
3. else if (~A & B) ns = beginOut; // car backs up
4. // else if (A & B) ns = blockOut; // impossible
5. else ns = blockOut; // car does not move

67

1. almostOut:
2. if (~A & ~B) // car exits
3. begin
4. ns = unblocked;
5. enter = 0;
6. exit = 1;
7. end
8. else if (A & B) ns = blockOut; // car backs up
9. // else if (A & ~B) ns = almostOut; // impossible
10. else ns = almostOut; // car does not move

78

79 /\*

1. default:
2. begin
3. enter = 0;
4. exit = 0;
5. end

85 \*/

1. endcase
2. end

88

89

1. // if reset, the parking lot sensor goes to unblocked case
2. always\_ff @(posedge clk)
3. begin
4. if (reset)
5. ps <= unblocked;
6. else ps <= ns;
7. end
8. endmodule 98
9. // parkingLotSensor\_testbench tests all expected behavior that the parking lot occupancy counter system in the lab may encounter
10. module parkingLotSensor\_testbench();
11. logic clk, reset, A, B, enter, exit; 102

103 parkingLotSensors dut (.clk(clk), .reset(reset), .A(A), .B(B), .enter (enter), .exit(exit));

104

105 parameter CLOCK\_PERIOD = 100; 106

107 initial begin

108 clk <= 0;

1. forever #(CLOCK\_PERIOD/2) clk <= ~clk;
2. end 111
3. // I have already simulate the case where car enters and exits without changing directions in DE1\_SoC
4. // For this testbench I will simulate the case where the car changes direction multiple times

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 114 | initial begin |  |  |  | | | | |
| 115 | reset <= 1; | @(posedge | clk); |
| 116 | reset <= 0; | @(posedge | clk); |
| 117 | {A, B} <= 2'b00; | @(posedge | clk); |
| 118 | {A, B} <= 2'b10; | @(posedge | clk); |
| 119 | {A, B} <= 2'b11; | @(posedge | clk); |
| 120 | {A, B} <= 2'b01; | @(posedge | clk); |
| 121 | {A, B} <= 2'b11; | @(posedge | clk); |
| 122 | {A, B} <= 2'b10; | @(posedge | clk); |
| 123 | {A, B} <= 2'b11; | @(posedge | clk); |
| 124 | {A, B} <= 2'b01; | @(posedge | clk); |
| 125 | {A, B} <= 2'b00; | @(posedge | clk); | // | simulation | for | car | enters |
| 126 |  |  |  |  |  |  |  |  |
| 127 | {A, B} <= 2'b00; | @(posedge | clk); |  |  |  |  |  |
| 128 | {A, B} <= 2'b01; | @(posedge | clk); |  |  |  |  |  |
| 129 | {A, B} <= 2'b11; | @(posedge | clk); |  |  |  |  |  |
| 130 | {A, B} <= 2'b10; | @(posedge | clk); |  |  |  |  |  |
| 131 | {A, B} <= 2'b11; | @(posedge | clk); |  |  |  |  |  |
| 132 | {A, B} <= 2'b01; | @(posedge | clk); |  |  |  |  |  |
| 133 | {A, B} <= 2'b11; | @(posedge | clk); |  |  |  |  |  |
| 134 | {A, B} <= 2'b10; | @(posedge | clk); |  |  |  |  |  |
| 135 | {A, B} <= 2'b00; | @(posedge | clk); | // | simulation | for | car | exits |
| 136 | $stop; |  |  |  |  |  |  |  |
| 137 | end |  |  |  |  |  |  |  |
| 138 | endmodule |  |  |  |  |  |  |  |
| 139 |  |  |  |  |  |  |  |  |
| 140 |  |  |  |  |  |  |  |  |
| 141 |  |  |  |  |  |  |  |  |
| 142 |  |  |  |  |  |  |  |  |

Date: January 15, 2022 userInput.sv Project: DE1\_SoC

|  |  |  |
| --- | --- | --- |
| 1 | // | Alan Li |
| 2 | // | 01/15/2022 |
| 3 | // | EE 371 |
| 4 | // | Lab #1 |
| 5 |  |  |
| 6 | // | userInput take 1-bit D as user input and return Q as output |
| 7 | // | This module utilize two DFF to reduce the possibility of |

metastability by adding latency

8

9 module userInput(clk, D, Q);

1. input clk, D;
2. output logic Q;
3. logic temp; 13
4. always\_ff @(posedge clk) begin
5. temp <= D;
6. Q <= temp;
7. end

18

19 endmodule