Password Protected System

Aim:
To design and implement a password protected system using logic gates and ICs.

Equipment and Components Required:

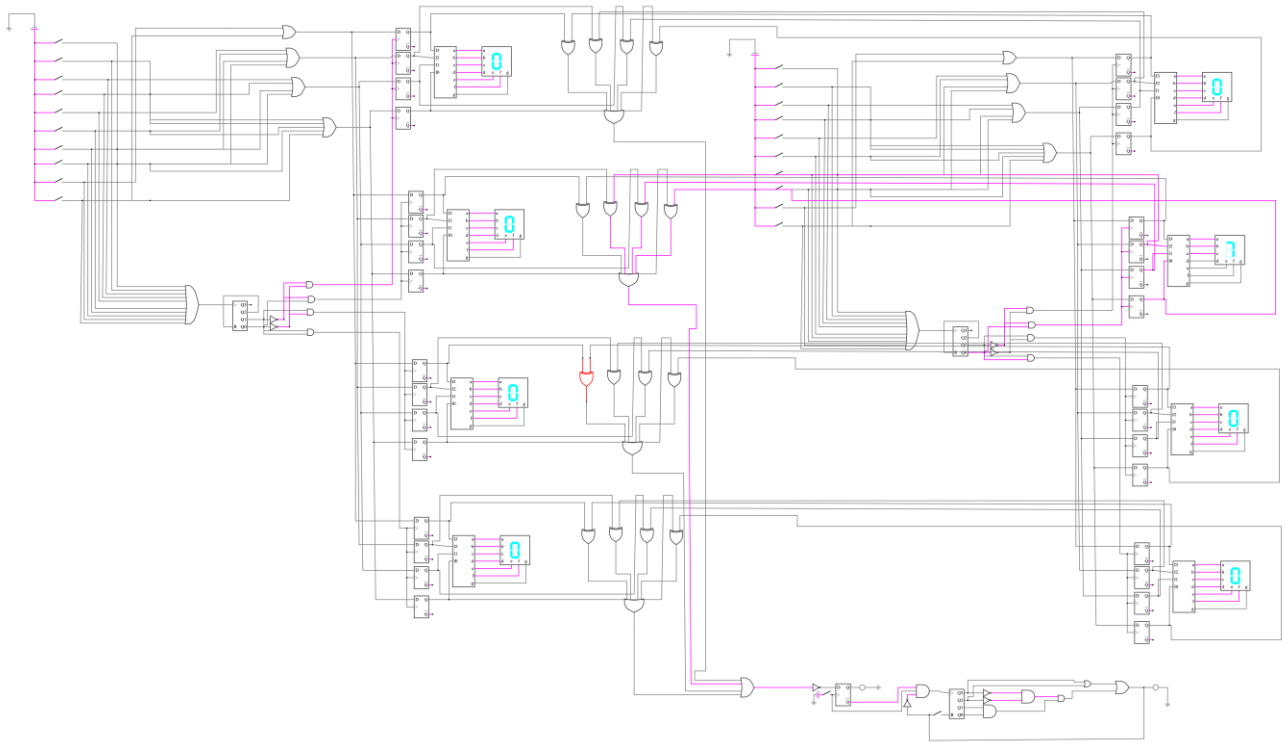
Sl. No.	Apparatus Required	Quantity
1	Digital IC Trainer Board	1
2	ICs 7404, 7408, 7432, CD4072, 74174, 74163, 7448, 7485, 7474, 7486	
3	LED	1
4	Buzzer	1
5	7 segment displays	8
6	Resistors (1k, 330 ohm)	1+56
8	Push Buttons	22
7	Connecting wires	As required

Theory:
The password protected system works on decimal to BCD conversion using k-map, which converts 4-digit decimal number to 16-bit BCD, and magnitude comparison between the password that was set and the password that was entered. The user inputs the set password using push bottoms, where each button corresponds to a particular decimal number, which is then displayed on 7-segment displays. Another set of push buttons are used to enter a password. These two 4-digit numbers are then converted to two 16-digit BCD numbers, which are compared to each other using 4 magnitude comparators, whose inputs are cascaded to compare 16 bits. The output A=B is then fed to a D-flip flop, and when another push button is pressed and if the password entered matches the set password, an LED will glow. If the password entered is wrong three times, a buzzer will buzz till a reset button is pressed.

Truth Table:

S[3:0]	C[3:0]	Enter	match_latched	success_led	wrong_count	led_out
0000	0000	1	1	1	00	0
0001	0000	1	0	0	01	0
0010	0000	1	0	0	10	1
0011	0000	1	0	0	11	1
0100	0100	1	1	1	00	0
0101	0011	1	0	0	01	0
0110	0110	1	1	1	00	0
0111	0101	1	0	0	01	0

Logic Diagram



Procedure:

1. Rig up the circuit as per the circuit diagram.
2. Give logical inputs as per the respective truth table.
3. Observe the logical output and verify with the truth table.

Verilog Program

```
`timescale 1ns/1ps

// ---------------- keyboard_to_bcd ----------------
module keyboard_to_bcd (
  input i0,i1,i2,i3,i4,i5,i6,i7,i8,i9,
  output dclk,o0,o1,o2,o3
);
  assign dclk = i0|i1|i2|i3|i4|i5|i6|i7|i8|i9;
  assign o0 = i1|i3|i5|i7|i9;
  assign o1 = i2|i3|i6|i7;
  assign o2 = i4|i5|i6|i7;
  assign o3 = i8|i9;
endmodule

// ---------------- D Flip-Flop ----------------
module dff (
  input clk,
  input rst,
  input d,
```

```
  output reg q
);
  always @(posedge clk or posedge rst)
    if (rst) q <= 1'b0;
    else q <= d;
endmodule

// ---------------- 8-bit Magnitude Comparator ----------------
module mag_comp_8 (
  input [7:0] A,
  input [7:0] B,
  output A_eq_B
);
  assign A_eq_B = (A == B);
endmodule

// ---------------- 8-bit binary to 2-digit BCD ----------------
module bin_to_bcd (
  input [7:0] bin,
  output reg [3:0] bcd1, bcd0
);
  integer i;
  reg [11:0] shift_reg;
  always @(*) begin
    bcd1 = 0; bcd0 = 0;
    shift_reg = {4'd0, bin};
    for(i=0;i<8;i=i+1) begin
      if(bcd1>=5) bcd1 = bcd1+3;
      if(bcd0>=5) bcd0 = bcd0+3;
      shift_reg = shift_reg << 1;
      {bcd1,bcd0} = shift_reg[11:4];
    end
  end
endmodule

// ---------------- 2-bit Counter ----------------
module counter_2bit_en (
  input clk,
  input rst,
  input en,
  output reg [1:0] count
);
  always @(posedge clk or posedge rst) begin
    if(rst) count <= 2'b00;
    else if(en)
      count <= (count==2'b11) ? 2'b00 : count+1'b1;
  end
endmodule

// ---------------- Top Module (8-bit version) ----------------
module top_stitched (
```

```
  input clk,
  input rst,
  input [7:0] s_key, // security side
  input [7:0] c_key, // client side
  input enter,
  output reg [1:0] wrong_count,
  output reg led_out,
  output reg match_latched,
  output reg success_led,
  output [3:0] s_bcd_tens,s_bcd_ones,
  output [3:0] c_bcd_tens,c_bcd_ones
);

  // 8 DFFs per side
  wire [7:0] s_q, c_q;
  genvar i;
  generate
    for(i=0;i<8;i=i+1) begin: S_DFFS
      dff dff_s(clk,rst,s_key[i],s_q[i]);
    end
    for(i=0;i<8;i=i+1) begin: C_DFFS
      dff dff_c(clk,rst,c_key[i],c_q[i]);
    end
  endgenerate

  // Convert to BCD
  bin_to_bcd s_bcd_conv(.bin(s_q),.bcd1(s_bcd_tens),.bcd0(s_bcd_ones));
  bin_to_bcd c_bcd_conv(.bin(c_q),.bcd1(c_bcd_tens),.bcd0(c_bcd_ones));

  // Comparator
  wire match_now;
  mag_comp_8 comp(.A(s_q),.B(c_q),.A_eq_B(match_now));

  // Latch match on Enter
  always @(posedge enter or posedge rst) begin
    if(rst) begin
      match_latched <= 0;
      success_led   <= 0;
    end else begin
      match_latched <= match_now;
      success_led   <= match_now;
    end
  end

  // Wrong-entry counter
  wire incr = enter & (~match_latched) & (~led_out);
  wire counter_rst = rst | match_latched;
  wire [1:0] cnt;
  counter_2bit_en counter(clk,counter_rst,incr,cnt);

  always @(posedge clk or posedge rst) begin
```

```
    if(rst) led_out <= 0;
    else if(match_latched) led_out <= 0;
    else if(cnt==2'b10 & incr) led_out <= 1;
    else led_out <= led_out;

    wrong_count <= cnt;
  end
endmodule

// ---------------- Testbench ----------------
module tb_top_stitched;
  reg clk,rst,enter;
  reg [7:0] s_key, c_key;
  wire [1:0] wrong_count;
  wire led_out, match_latched, success_led;
  wire [3:0] s_bcd_tens,s_bcd_ones;
  wire [3:0] c_bcd_tens,c_bcd_ones;

  top_stitched
DUT(clk,rst,s_key,c_key,enter,wrong_count,led_out,match_latched,success_led,
              s_bcd_tens,s_bcd_ones,c_bcd_tens,c_bcd_ones);

  initial clk=0;
  always #5 clk=~clk;

  initial begin
    rst=1; enter=0;
    s_key = 8'b0;
    c_key = 8'b0;
    #20 rst=0;

    // Test Case 1: matching keys
    $display("Test Case 1: Matching keys");
    s_key = 8'b00010010; c_key = 8'b00010010;
    #10 enter=1; #10 enter=0; #10;
    $display("Match=%b, Success LED=%b, Wrong LED=%b, Count=%b",
           match_latched, success_led, led_out, wrong_count);

    // Test Case 2: wrong entry
    $display("Test Case 2: Wrong entry");
    s_key = 8'b00100100; c_key = 8'b00010010;
    #10 enter=1; #10 enter=0; #10;
    $display("Match=%b, Success LED=%b, Wrong LED=%b, Count=%b",
           match_latched, success_led, led_out, wrong_count);

    // Test Case 3: multiple wrong entries
    $display("Test Case 3: Multiple wrong entries");
    repeat(3) begin
      s_key = 8'b00100100; c_key = 8'b00010010;
      #10 enter=1; #10 enter=0; #10;
      $display("Match=%b, Success LED=%b, Wrong LED=%b, Count=%b",
```

```
            match_latched, success_led, led_out, wrong_count);
    end

    // Test Case 4: correct entry clears LED
    $display("Test Case 4: Correct entry clears LED");
    s_key = 8'b00010010; c_key = 8'b00010010;
    #10 enter=1; #10 enter=0; #10;
    $display("Match=%b, Success LED=%b, Wrong LED=%b, Count=%b",
            match_latched, success_led, led_out, wrong_count);

    $display("Simulation finished.");
    $stop;
  end
endmodule
```

Simulation Output

Log | Share

```
[2025-10-02 08:57:12 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Test Case 1: Matching keys
Match=1, Success LED=1, Wrong LED=0, Count=00
Test Case 2: Wrong entry
Match=0, Success LED=0, Wrong LED=0, Count=01
Test Case 3: Multiple wrong entries
Match=0, Success LED=0, Wrong LED=0, Count=10
Match=0, Success LED=0, Wrong LED=1, Count=11
Match=0, Success LED=0, Wrong LED=1, Count=11
Test Case 4: Correct entry clears LED
Match=1, Success LED=1, Wrong LED=0, Count=00
Simulation finished.
```

Result:

Thus, the password protected system was designed and implemented successfully and the output was verified.