# DIGITAL LOGIC AND DESIGN

# PROJECT

# VOTING MACHINE

**NAME : KARTHIKA P**

**ROLL NO : 2021115049**

**AIM**

To count the votes by using counter in voting machine.

**Software used**

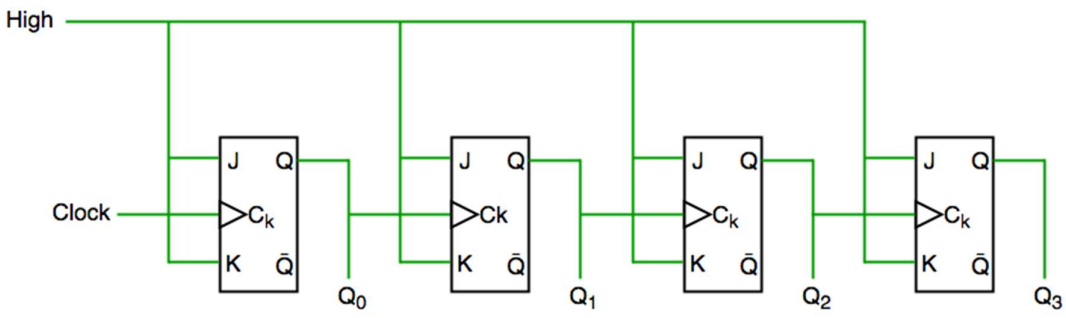
EDA playground,verilog

**BASIC DESIGN**

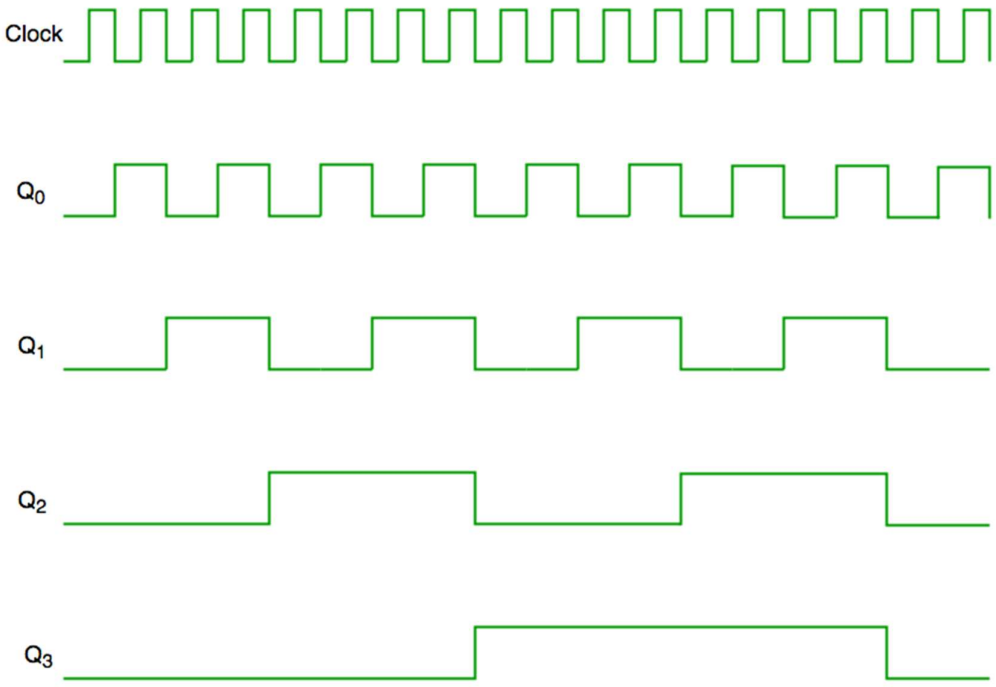
Counters

**EXPLANATION**

- Voting machine is designed using counters.
- When the voter votes for a candidate,it counts the number of votes For each candidate by using counter
- Finally it diplays the total number of votes
- Number of votes for each candidate is also counted by this voting machine
- In this input is given by pressing button




(a) Asynchronous counter




(b) Timing Diagram

**CODE**

**TESTBENCH.SV**

```
// Code your testbench here
// or browse Examples
`timescale 1ns/1ns                    // 1 time unit = 1 ns

module tb_voting_machine ();
reg t_clk;
reg t_rst;
reg t_candidate_1;
reg t_candidate_2;
reg t_candidate_3;
reg t_vote_over;

wire [5:0] t_result_1;
wire [5:0] t_result_2;
wire [5:0] t_result_3;

//instansiate component unit under test
voting_machine uut (
.clk(t_clk),
.rst(t_rst),
.i_candidate_1(t_candidate_1),
.i_candidate_2(t_candidate_2),
.i_candidate_3(t_candidate_3),
```

```
.i_voting_over(t_vote_over),
.o_count1(t_result_1),
.o_count2(t_result_2),
.o_count3(t_result_3)
);

//initial value of clock at 0 time
initial t_clk = 1'b1;

// clock generation
always
begin
   #5 t_clk = ~ t_clk;
end

//stimulus to design
initial
begin
   $monitor ("time = %d, rst = %b, candidate_1 = %b, candidate_2 = %b, candidate_3 = %b, vote_over = %b, result_1 = %3d, result_2 = %3d, result_3 = %3d,\n",
   $time, t_rst, t_candidate_1, t_candidate_2, t_candidate_3, t_vote_over, t_result_1, t_result_2, t_result_3,);



   t_rst = 1'b1;
   t_candidate_1 = 1'b0;
```

```
t_candidate_2 = 1'b0;
t_candidate_3 = 1'b0;
t_vote_over = 1'b0;

#20 t_rst = 1'b0;
#10 t_candidate_1 = 1'b1;          //when button for candidate 1 is pressed
#10 t_candidate_1 = 1'b0;          //button  for candidate 1 is released

#20 t_candidate_2 = 1'b1;          //when button for candidate 2 is pressed
#10 t_candidate_2 = 1'b0;          //button  for candidate 2 is released

#20 t_candidate_1 = 1'b1;          //when button for candidate 1 is pressed
#10 t_candidate_1 = 1'b0;          //button  for candidate 1 is released

#20 t_candidate_3 = 1'b1;           //when button for candidate 3 is pressed
#10 t_candidate_3 = 1'b0;           //button  for candidate 3 is released

#20 t_candidate_2 = 1'b1;
#10 t_candidate_2 = 1'b0;

#20 t_candidate_2 = 1'b1;
#10 t_candidate_2 = 1'b0;

#20 t_candidate_1 = 1'b1;
#10 t_candidate_1 = 1'b0;
```

```
   #20 t_candidate_3 = 1'b1;            //when button for candidate 3 is pressed
   #10 t_candidate_3 = 1'b0;            //button  for candidate 3 is released


   #30 t_vote_over = 1'b1;

   #50 t_rst = 1'b1;                      //reset when the voting process is over

   //use $finish for simulators other than modelsim
   #60 $stop;                           // use $stop instead of $finish to keep
modelsim simulator open
end

//.vcd file for gtk wave plot
initial
begin
   $dumpfile ("dump.vcd");
   $dumpvars;
end


endmodule
```

**DESIGN.SV**

```
`timescale 1ns / 1ps

module voting_machine #(
```

```
parameter idle =  2'b00,                    // states and their
corresponding numbers
parameter vote =  2'b01,
parameter hold =  2'b10,
parameter finish = 2'b11
)(
   input clk,
   input rst,                               // input High to reset
counting  (active high)
   input i_candidate_1,                     // input to vote for candidate 1
   input i_candidate_2,                     // input to vote for candidate 2
   input i_candidate_3,                     // input to vote for candidate 3
   input i_voting_over,                     // input high to get total votes
after voting is over

   output reg [31:0] o_count1,              // output for total number of
votes of candidate 1
   output reg [31:0] o_count2,              // output for total number of
votes of candidate 2
   output reg [31:0] o_count3               // output for total number of
votes of candidate 3

);

reg [31:0] r_cand1_prev;                    // store previous value of
input for candidate 1
reg [31:0] r_cand2_prev;                    // store previous value of
input for candidate 2
```

```
reg [31:0] r_cand3_prev;                    // store previous value of
input for candidate 3


reg [31:0] r_counter_1;                     // counting register
for candidate 1
reg [31:0] r_counter_2;                     // counting register
for candidate 2
reg [31:0] r_counter_3;                     // counting register
for candidate 3


reg [1:0] r_present_state, r_next_state;    // present state and next
state registers
//reg [1:0] r_state_no;                     // store state number
reg [3:0] r_hold_count;          //counter for hold state


////////////// always block that assigns next state & internal reg operations
////////////////////////////////////
always @(posedge clk or negedge rst)
        begin
                case (r_present_state)



                        idle: if (!rst)
                                // idle state operations
                                        begin
                                                r_next_state <= vote;
                                                // assign next state vote when
reset low
                                                //r_state_no <= 2'b01;
```

```
                end


            else
                begin
                //r_present_state = idle;
                // present state at the beginning
                    r_counter_1 <= 32'b0;
                        // clear counting registers
                    r_counter_2 <= 32'b0;
                    r_counter_3 <= 32'b0;
                    r_hold_count <= 4'b0000;


                    r_next_state <= idle;
                    // assign next state as idle till reset
not low
                    //r_state_no <= 2'b0;
                end



        vote: if (i_voting_over == 1'b1)
            // check if voting is over
                begin
                    r_next_state <= finish;
                // if over is high go to finish state
                    //r_state_no <= 2'b11;
                end
                                                                    //
if over is low contiue counting
```

```
                                else if (i_candidate_1 == 1'b0 && r_cand1_prev
== 1'b1)        // check falling edge of input candidate1 so only single input is
regeistered
                                        begin
                                                r_counter_1 <= r_counter_1 +
1'b1;                                           // increment counter for
candidate 1
                                                //r_counter_2 <= r_counter_2;
                                                // keep previous value of
counter
                                                //r_counter_3 <= r_counter_3;
                                                // keep previous value of
counter


                                                r_next_state <= hold;
                                                // got to hold state
                                                //r_state_no <= 2'b10;
                                        end


                                else if (i_candidate_2 == 1'b0 && r_cand2_prev
== 1'b1)                // check falling edge of input candidate2 so only single input
is regeistered
                                        begin
                                                //r_counter_1 <= r_counter_1;
                                                // keep previous value of
counter
                                                r_counter_2 <= r_counter_2 +
1'b1;                                           // increment counter for
candidate 2
```

```
                                        //r_counter_3 <= r_counter_3;
                                                // keep previous value of
counter


                                        r_next_state <= hold;
                                        // got to hold state

                                        //r_state_no <= 2'b10;

                                end


                        else if (i_candidate_3 == 1'b0 && r_cand3_prev
== 1'b1)        // check falling edge of input candidate3 so only single input
is regeistered

                                begin

                                        //r_counter_1 <= r_counter_1;
                                                // keep previous value of
counter

                                        //r_counter_2 <= r_counter_2;
                                                // keep previous value of
counter

                                        r_counter_3 <= r_counter_3 +
1'b1;                                           // increment counter for
candidate 3


                                        r_next_state <= hold;
                                        // got to hold state

                                        //r_state_no <= 2'b10;

                                end

                        else
                                                // none of the input present
or more than 1 input present at same time
```

```
                                        begin
                                                r_counter_1 <= r_counter_1;
                                                        // keep previous value of
counter
                                                r_counter_2 <= r_counter_2;

                                                r_counter_3 <= r_counter_3;


                                                r_next_state <= vote;
                                                //r_state_no <= 2'b01;
                                        end


                                hold: if (i_voting_over == 1'b1)
                                        // check if over input is high
                                        begin
                                                r_next_state <= finish;
                                                // go to finish state
                                                //r_state_no <= 2'b11;
                                        end


                                else
                                        begin
                                                if (r_hold_count != 4'b1111) begin
                                                        r_hold_count =
r_hold_count + 1'b1;
                                                end
                                                else begin
```

```
                    r_next_state <= vote;
                // if over is low go to vote state
                end
                //r_state_no <= 2'b01;
            end

        finish: if (i_voting_over == 1'b0)

            begin
                r_next_state <= idle;
                // if over is low go to idle state
                //r_state_no <= 2'b0;
            end

        else
            begin
                r_next_state <= finish;
                // remain in finish state if over is
high
                //r_state_no <= 2'b11;
            end

        default:
            begin
                r_counter_1 <= 32'b0;
                // default values for resgisters
                r_counter_2 <= 32'b0;
                r_counter_3 <= 32'b0;
```

```
                                r_hold_count <= 4'b0000;


                                r_next_state <= idle;
                                        // by default go to idle state at the
begining
                                //r_state_no <= 2'b0;
                        end
                endcase
        end


////////////// always block that performs assignment of regsiters and output
on clock signal ////////////////////////////
always @(posedge clk or negedge rst)
                                // work on positive edge of clock
        begin


    if (rst == 1'b1)
                begin
                        r_present_state <= idle;
                                // remain in idle state when reset is high



                        o_count1 <= 32'b0;
                                        // reset final output count
                        o_count2 <= 32'b0;
                        o_count3 <= 32'b0;
                        r_hold_count <= 4'b0000;
                end
```

```
            else if (rst == 1'b0 && i_voting_over == 1'b1)
                                                    // if voting process is
i.e.over is high
                    begin
                        o_count1 <= r_counter_1;
                                        // provide value of counting
registers at output
                        o_count2 <= r_counter_2;
                        o_count3 <= r_counter_3;
                    end


            else
                    begin
                        r_present_state <= r_next_state;
                                // if reset is low keep assigning next state
to present state
                        r_cand1_prev <= i_candidate_1;
                                // keep assigning input of candidate 1 to
internal register
                        r_cand2_prev <= i_candidate_2;
                        r_cand3_prev <= i_candidate_3;
                    end


        end


endmodule
```

**OUTPUT**

[2022-12-20 00:38:35 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out

testbench.sv:18: warning: Port 7 (o_count1) of voting_machine expects 32 bits, got 6.

testbench.sv:18:        : Padding 26 high bits of the port.

testbench.sv:18: warning: Port 8 (o_count2) of voting_machine expects 32 bits, got 6.

testbench.sv:18:        : Padding 26 high bits of the port.

testbench.sv:18: warning: Port 9 (o_count3) of voting_machine expects 32 bits, got 6.

testbench.sv:18:        : Padding 26 high bits of the port.

VCD info: dumpfile dump.vcd opened for output.

time =                  0, rst = 1, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 =   0, result_2 =   0, result_3 =   0,

time =                 20, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 =   0, result_2 =   0, result_3 =   0,

time =                 30, rst = 0, candidate_1 = 1, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 =   0, result_2 =   0, result_3 =   0,

time =                 40, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 =   0, result_2 =   0, result_3 =   0,

time =                 60, rst = 0, candidate_1 = 0, candidate_2 = 1, candidate_3 = 0, vote_over = 0, result_1 =   0, result_2 =   0, result_3 =   0,

time =                 70, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 =   0, result_2 =   0, result_3 =   0,

time = 90, rst = 0, candidate_1 = 1, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 100, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 120, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 1, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 130, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 150, rst = 0, candidate_1 = 0, candidate_2 = 1, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 160, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 180, rst = 0, candidate_1 = 0, candidate_2 = 1, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 190, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 210, rst = 0, candidate_1 = 1, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 220, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 240, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 1, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 250, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 0, result_1 = 0, result_2 = 0, result_3 = 0,

time = 280, rst = 0, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 1, result_1 = 3, result_2 = 3, result_3 = 2,

time = 330, rst = 1, candidate_1 = 0, candidate_2 = 0, candidate_3 = 0, vote_over = 1, result_1 = 0, result_2 = 0, result_3 = 0,

** VVP Stop(0) **

** Flushing output streams.

** Current simulation time is 390000 ticks.

**WAVEFORM**