Simple Voting Machine

19ECE383 VLSI Design Lab

B. Tech. ECE

Batch: C2

2023-2024 Even Semester

CB.EN.U4ECE21206	CH K VARDHAN
CB.EN.U4ECE21207	D HARSHA VARDHAN
CB.EN.U4ECE21236	N PRASANNA

Faculty Incharges: Dr. Navya Mohan, Dr. Bala Tripura Sundari B

Signature of the Faculty:

Department of Electronics and Communication Engineering

Amrita School of Engineering

Amrita Vishwa Vidyapeetham

Amritanagar, Coimbatore – 641112

1) OBJECTIVE :

The aim of this project is to develop a basic voting system employing FPGA technology and Vivado. This system will enable voters to select from a set of options using buttons and will display the results using LEDs. The project involves integrating both combinational and sequential logic to construct a prototype capable of registering votes, showcasing results, and resetting for subsequent use. Furthermore, it will incorporate mechanisms to authenticate button inputs to guarantee precise vote recording and to maintain a tally of votes for each option, which will be displayed for reference.

2) SOFTWARE AND HARDWARE USED:

- Vivado software tool
- ZED Board

3) OVERVIEW OF THE SYSTEM:

System Components:

1. Voting Interface:

- Buttons: Used by voters to select their desired options.
- Debouncing Mechanism**: Ensures that each button press is registered accurately without the effects of mechanical noise.

2. Vote Processing Unit:

- Combinational Logic: Decodes button inputs and directs them to the appropriate voting counters.
- Sequential Logic: Manages the state of the voting process, including vote counting and result display.

3. Vote Counting and Tallying:

- Counters: Maintain a tally of votes for each option.
- Registers: Store the vote counts and ensure that they are updated correctly with each valid button press.

4. Result Display:

- LEDs: Indicate the number of votes received for each option, providing a visual representation of the voting outcome.

5. Control and Reset Mechanism:

- Reset Button: Resets the entire system for subsequent voting sessions.
- Control Logic: Manages the reset process and ensures that the system returns to its initial state.

System Operation:

1. Initialization:

- Upon power-up or reset, the system initializes all counters and registers to zero.
- LEDs are turned off, indicating that no votes have been cast.

2. Voting Process:

- Voters press the buttons corresponding to their preferred options.
- Each button press is debounced to ensure accurate input.
- The combinational logic decodes the button press and sends a signal to the appropriate counter.
- The counter associated with the selected option increments by one.

3. Vote Counting:

- Sequential logic updates the registers with the current vote counts.
- The updated vote counts are continuously displayed using LEDs.

4. Result Display:

- LEDs provide a real-time display of the number of votes each option has received.
- Each LED or set of LEDs corresponds to a specific voting option.

5. Resetting the System:

- Pressing the reset button clears all counters and registers.

Design Implementation

1. Button Input and Debouncing:

- Implemented using flip-flops and delay elements to filter out noise from mechanical button presses.
- Ensures that only a single, clean signal is sent to the voting logic for each button press.

2. Combinational Logic:

- Designed using basic logic gates (AND, OR, NOT) to decode button inputs.
- Routes signals to the correct vote counters based on the button pressed.

3. Sequential Logic:

- Utilizes flip-flops and state machines to manage the voting process.
- Ensures accurate counting and updating of votes.

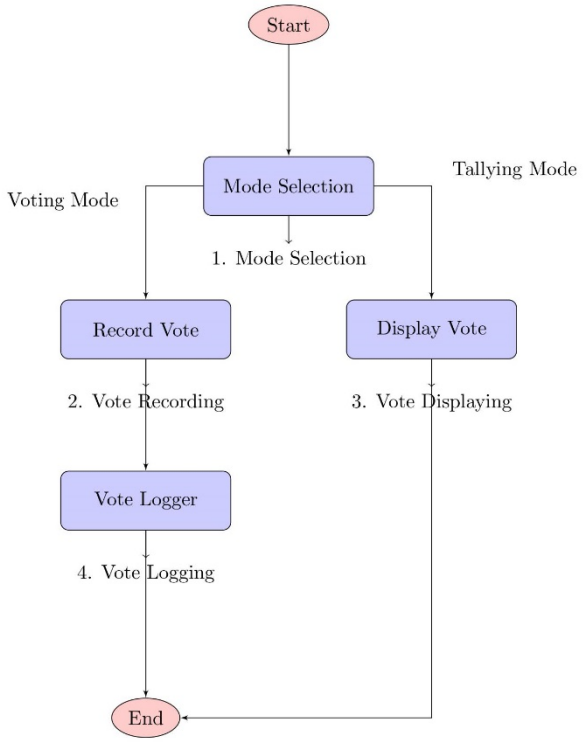
4. Vote Counters and Registers:

- Implemented as binary counters using flip-flops and adders.
- Registers hold the current vote counts and update the LED display accordingly.

5. LED Display:

- Each voting option is represented by a specific LED or a set of LEDs.
- LEDs are controlled by the outputs of the vote counters, providing a visual tally of votes.
- 6. Reset Logic:
- A dedicated reset button triggers a reset sequence.
- All counters and registers are cleared, and the system is prepared for a new voting cycle.

4) BLOCK DIAGRAM:



5) INTRODUCTION:

This system is a digital voting machine designed to streamline the process of collecting and counting votes. It leverages a combination of hardware components to ensure accurate and reliable vote counting, making it suitable for use in various voting scenarios such as meetings, polls, and small elections. The system is composed of several key components, each playing a crucial role in the overall operation:

1. **Voting Interface**:

   This component includes the physical buttons that voters use to select their preferred options. Each button is equipped with a debouncing mechanism to ensure that only valid presses are registered, eliminating issues caused by mechanical noise and ensuring the integrity of the voting process.

2. **Vote Processing Unit**: At the core of the system, the vote processing unit uses combinational logic to decode inputs from the buttons and direct them to the appropriate vote counters. Sequential logic is employed to manage the state of the voting process, including the incrementing of vote counts and the display of results.

3. **Vote Counting and Tallying**: This section features counters that tally the votes for each option. These counters work in conjunction with registers that store and update the vote counts, ensuring that every vote is accurately recorded and reflected in real-time.

4. **Result Display**: The results of the voting process are displayed using LEDs, which provide a visual representation of the number of votes each option has received. This immediate feedback allows for quick assessment of voting outcomes.

5. **Control and Reset Mechanism**: A reset button allows the system to be cleared and prepared for new voting sessions. The control logic ensures that all counters and registers are reset to their initial states, facilitating a smooth transition between voting sessions.

6) COMPLETE CODE WITH COMMENTS:

CODE FOR MODE CONTROL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Entity declaration for modeControl
entity modeControl is
    Port (
        clock           : in  STD_LOGIC;            -- Clock input
        reset           : in  STD_LOGIC;            -- Reset input
        mode            : in  STD_LOGIC;            -- Mode input: '0' for counting mode, '1' for vote display mode
        valid_vote_casted   : in  STD_LOGIC;            -- Signal indicating a valid vote has been casted
        candidate1_vote     : in  STD_LOGIC_VECTOR(7 downto 0); -- Vote count for candidate 1
        candidate2_vote     : in  STD_LOGIC_VECTOR(7 downto 0); -- Vote count for candidate 2
        candidate3_vote     : in  STD_LOGIC_VECTOR(7 downto 0); -- Vote count for candidate 3
        candidate4_vote     : in  STD_LOGIC_VECTOR(7 downto 0); -- Vote count for candidate 4
```

```
        candidate1_button_press : in  STD_LOGIC;          -- Button press signal for candidate 1
        candidate2_button_press : in  STD_LOGIC;          -- Button press signal for candidate 2
        candidate3_button_press : in  STD_LOGIC;          -- Button press signal for candidate 3
        candidate4_button_press : in  STD_LOGIC;          -- Button press signal for candidate 4
        leds                    : out STD_LOGIC_VECTOR(7 downto 0) -- Output LEDs
    );
end modeControl;

architecture Behavioral of modeControl is
    -- Internal signal declaration for the counter
    signal counter : unsigned(30 downto 0) := (others => '0');
begin
    -- Process to handle the counter logic
    process(clock, reset)
    begin
        if reset = '1' then
            -- Reset the counter to zero when reset signal is high
            counter <= (others => '0');
        elsif rising_edge(clock) then
            if valid_vote_casted = '1' then
                -- Increment counter when a valid vote is casted
                counter <= counter + 1;
            elsif (counter /= 0 and counter < 100000000) then
                -- Continue incrementing counter if it is non-zero and less than a threshold
                counter <= counter + 1;
            else
                -- Reset counter if it reaches the threshold
                counter <= (others => '0');
            end if;
        end if;
    end process;

    -- Process to handle the LED output based on mode and button presses
    process(clock, reset, mode, candidate1_button_press, candidate2_button_press, candidate3_button_press,
candidate4_button_press, candidate1_vote, candidate2_vote, candidate3_vote, candidate4_vote)
```

```
    begin
        if reset = '1' then
            -- Reset LEDs to zero when reset signal is high
            leds <= (others => '0');
        elsif rising_edge(clock) then
            if mode = '0' and counter > 0 then
                -- In counting mode, if counter is non-zero, turn on all LEDs
                leds <= (others => '1');
            elsif mode = '0' then
                -- In counting mode, if counter is zero, turn off all LEDs
                leds <= (others => '0');
            elsif mode = '1' then
                -- In vote display mode, display the vote count for the pressed candidate
                if candidate1_button_press = '1' then
                    leds <= candidate1_vote;
                elsif candidate2_button_press = '1' then
                    leds <= candidate2_vote;
                elsif candidate3_button_press = '1' then
                    leds <= candidate3_vote;
                elsif candidate4_button_press = '1' then
                    leds <= candidate4_vote;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

CODE FOR VOTE LOGGER:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Entity declaration for voteLogger
entity voteLogger is
```

```
    Port (
        clock : in STD_LOGIC;                   -- Clock input
        reset : in STD_LOGIC;                   -- Reset input
        mode : in STD_LOGIC;                     -- Mode input: '0' for logging mode, '1' for other mode (logging
disabled)
        cand1_vote_valid : in STD_LOGIC;          -- Vote valid signal for candidate 1
        cand2_vote_valid : in STD_LOGIC;          -- Vote valid signal for candidate 2
        cand3_vote_valid : in STD_LOGIC;          -- Vote valid signal for candidate 3
        cand4_vote_valid : in STD_LOGIC;          -- Vote valid signal for candidate 4
        cand1_vote_recvd : out STD_LOGIC_VECTOR (7 downto 0); -- Output vote count for candidate 1
        cand2_vote_recvd : out STD_LOGIC_VECTOR (7 downto 0); -- Output vote count for candidate 2
        cand3_vote_recvd : out STD_LOGIC_VECTOR (7 downto 0); -- Output vote count for candidate 3
        cand4_vote_recvd : out STD_LOGIC_VECTOR (7 downto 0)  -- Output vote count for candidate 4
    );
end voteLogger;

architecture Behavioral of voteLogger is
    -- Internal signals to hold the vote counts for each candidate
    signal cand1_vote_recvd_internal : unsigned(7 downto 0) := (others => '0');
    signal cand2_vote_recvd_internal : unsigned(7 downto 0) := (others => '0');
    signal cand3_vote_recvd_internal : unsigned(7 downto 0) := (others => '0');
    signal cand4_vote_recvd_internal : unsigned(7 downto 0) := (others => '0');
begin
    -- Process to handle the vote logging
    process (clock)
    begin
        if rising_edge(clock) then
            if reset = '1' then
                -- Reset all vote counts to zero when reset signal is high
                cand1_vote_recvd_internal <= (others => '0');
                cand2_vote_recvd_internal <= (others => '0');
                cand3_vote_recvd_internal <= (others => '0');
                cand4_vote_recvd_internal <= (others => '0');
            else
                -- Increment the respective candidate's vote count if the vote is valid and mode is '0'
```

```
            if cand1_vote_valid = '1' and mode = '0' then
               cand1_vote_recvd_internal <= cand1_vote_recvd_internal + 1;
            elsif cand2_vote_valid = '1' and mode = '0' then
               cand2_vote_recvd_internal <= cand2_vote_recvd_internal + 1;
            elsif cand3_vote_valid = '1' and mode = '0' then
               cand3_vote_recvd_internal <= cand3_vote_recvd_internal + 1;
            elsif cand4_vote_valid = '1' and mode = '0' then
               cand4_vote_recvd_internal <= cand4_vote_recvd_internal + 1;
            end if;
         end if;
      end if;
   end process;

   -- Assign the internal vote count signals to the output ports
   cand1_vote_recvd <= std_logic_vector(cand1_vote_recvd_internal);
   cand2_vote_recvd <= std_logic_vector(cand2_vote_recvd_internal);
   cand3_vote_recvd <= std_logic_vector(cand3_vote_recvd_internal);
   cand4_vote_recvd <= std_logic_vector(cand4_vote_recvd_internal);
end Behavioral;
```

CODE FOR BUTTON CONTROL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for buttonControl
entity buttonControl is
   Port (
      clock      : in  std_logic;   -- Clock input
      reset      : in  std_logic;   -- Reset input
      button     : in  std_logic;   -- Button input signal
      valid_vote  : out std_logic    -- Output signal indicating a valid vote
   );
end buttonControl;
```

```
architecture Behavioral of buttonControl is
    -- Internal signal to hold the previous state of the button
    signal button_previous : std_logic := '0';
begin
    -- Process to handle button press detection
    process (clock, reset)
    begin
        if reset = '1' then
            -- Reset internal signals when reset signal is high
            button_previous <= '0';
            valid_vote <= '0';
        elsif rising_edge(clock) then
            -- Detect a button press (rising edge of the button signal)
            if button = '1' and button_previous = '0' then
                valid_vote <= '1';  -- Set valid_vote high on a button press
            else
                valid_vote <= '0';  -- Set valid_vote low if no button press
            end if;
            -- Update the previous button state
            button_previous <= button;
        end if;
    end process;
end Behavioral;
```

CODE FOR VOTING MACHINE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for votingMachine
entity votingMachine is
    Port (
        clock     : in  STD_LOGIC;    -- Clock input
        reset     : in  STD_LOGIC;    -- Reset input
        mode      : in  STD_LOGIC;    -- Mode input
```

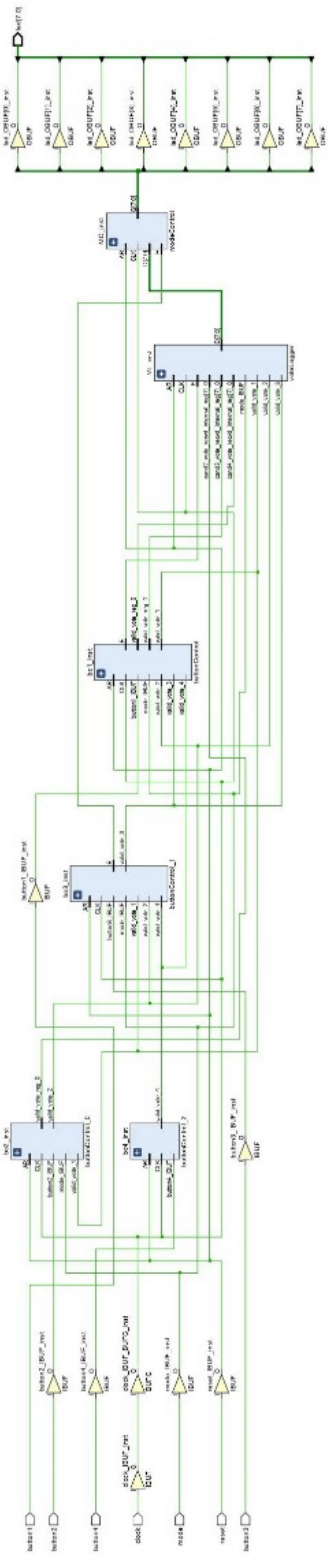
```
        button1    : in  STD_LOGIC;    -- Button input for candidate 1
        button2    : in  STD_LOGIC;    -- Button input for candidate 2
        button3    : in  STD_LOGIC;    -- Button input for candidate 3
        button4    : in  STD_LOGIC;    -- Button input for candidate 4
        led        : out STD_LOGIC_VECTOR(7 downto 0) -- LED output
    );
end votingMachine;

architecture Behavioral of votingMachine is
    -- Internal signals for valid vote detection for each candidate
    signal valid_vote_1 : std_logic;
    signal valid_vote_2 : std_logic;
    signal valid_vote_3 : std_logic;
    signal valid_vote_4 : std_logic;

    -- Internal signals for vote counts for each candidate
    signal cand1_vote_recvd : std_logic_vector(7 downto 0);
    signal cand2_vote_recvd : std_logic_vector(7 downto 0);
    signal cand3_vote_recvd : std_logic_vector(7 downto 0);
    signal cand4_vote_recvd : std_logic_vector(7 downto 0);

    -- Internal signal to indicate any valid vote
    signal anyValidVote : std_logic;

    -- Component declaration for buttonControl
    component buttonControl
        Port (
            clock     : in  STD_LOGIC;
            reset     : in  STD_LOGIC;
            button    : in  STD_LOGIC;
            valid_vote : out STD_LOGIC
        );
    end component;
```

```
-- Component declaration for voteLogger
component voteLogger
    Port (
        clock            : in  STD_LOGIC;
        reset            : in  STD_LOGIC;
        mode             : in  STD_LOGIC;
        cand1_vote_valid   : in  STD_LOGIC;
        cand2_vote_valid   : in  STD_LOGIC;
        cand3_vote_valid   : in  STD_LOGIC;
        cand4_vote_valid   : in  STD_LOGIC;
        cand1_vote_recvd   : out STD_LOGIC_VECTOR(7 downto 0);
        cand2_vote_recvd   : out STD_LOGIC_VECTOR(7 downto 0);
        cand3_vote_recvd   : out STD_LOGIC_VECTOR(7 downto 0);
        cand4_vote_recvd   : out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;

-- Component declaration for modeControl
component modeControl
    Port (
        clock              : in  STD_LOGIC;
        reset              : in  STD_LOGIC;
        mode               : in  STD_LOGIC;
        valid_vote_casted    : in  STD_LOGIC;
        candidate1_vote      : in  STD_LOGIC_VECTOR(7 downto 0);
        candidate2_vote      : in  STD_LOGIC_VECTOR(7 downto 0);
        candidate3_vote      : in  STD_LOGIC_VECTOR(7 downto 0);
        candidate4_vote      : in  STD_LOGIC_VECTOR(7 downto 0);
        candidate1_button_press : in  STD_LOGIC;
        candidate2_button_press : in  STD_LOGIC;
        candidate3_button_press : in  STD_LOGIC;
        candidate4_button_press : in  STD_LOGIC;
        leds               : out STD_LOGIC_VECTOR(7 downto 0)
    );
```

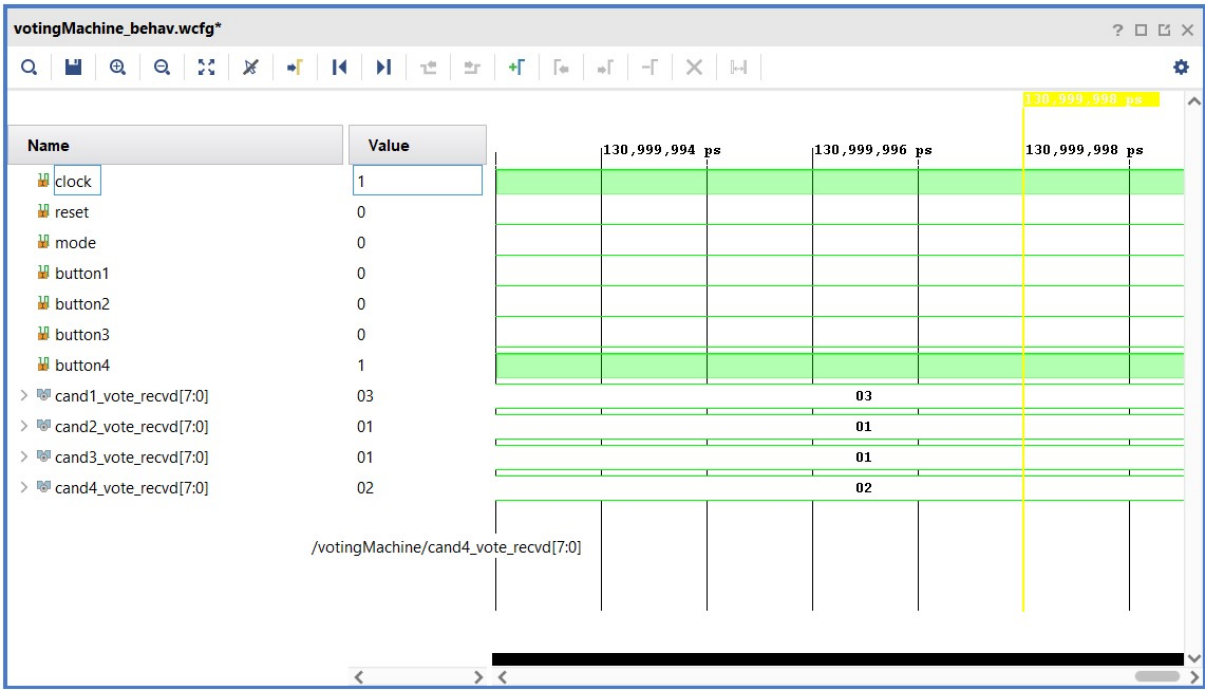
```
    end component;

begin
    -- Instantiate buttonControl for each candidate's button
    bc1_inst : buttonControl
        Port map (
            clock     => clock,
            reset     => reset,
            button    => button1,
            valid_vote => valid_vote_1
        );

    bc2_inst : buttonControl
        Port map (
            clock     => clock,
            reset     => reset,
            button    => button2,
            valid_vote => valid_vote_2
        );
   bc3_inst : buttonControl
        Port map (
            clock     => clock,
            reset     => reset,
            button    => button3,
            valid_vote => valid_vote_3
        );
     bc4_inst : buttonControl
        Port map (
            clock     => clock,
            reset     => reset,
            button    => button4,
            valid_vote => valid_vote_4
        );
  -- Instantiate voteLogger to log votes for each candidate
```

```
    VL_inst : voteLogger
        Port map (
            clock          => clock,
            reset          => reset,
            mode           => mode,
            cand1_vote_valid => valid_vote_1,
            cand2_vote_valid => valid_vote_2,
            cand3_vote_valid => valid_vote_3,
            cand4_vote_valid => valid_vote_4,
            cand1_vote_recvd => cand1_vote_recvd,
            cand2_vote_recvd => cand2_vote_recvd,
            cand3_vote_recvd => cand3_vote_recvd,
            cand4_vote_recvd => cand4_vote_recvd
        );
    -- Instantiate modeControl to manage LED output based on mode and votes
    MC_inst : modeControl
        Port map (
            clock              => clock,
            reset              => reset,
            mode               => mode,
            valid_vote_casted  => anyValidVote,
            candidate1_vote    => cand1_vote_recvd,
            candidate2_vote    => cand2_vote_recvd,
            candidate3_vote    => cand3_vote_recvd,
            candidate4_vote    => cand4_vote_recvd,
            candidate1_button_press => valid_vote_1,
            candidate2_button_press => valid_vote_2,
            candidate3_button_press => valid_vote_3,
            candidate4_button_press => valid_vote_4,
            leds               => led
        );
    -- OR logic to determine if any valid vote has been cast
    anyValidVote <= valid_vote_1 or valid_vote_2 or valid_vote_3 or valid_vote_4;
end Behavioral;
```

7) RTL:



8) Output Wveform:



9) Hardware Image: