# Electronic Voting Machine (EVM) – Specification Document

## 1. Project Overview

The Electronic Voting Machine (EVM) project is a digital system designed using Verilog HDL to simulate the functionality of a real-world voting machine. The primary goal of this project is to implement an efficient, error-free, and user-friendly design for secure voting operations. It demonstrates how digital state machines can be utilized for sequential operations involving inputs, outputs, and internal decision-making processes.

## 2. Functional Description

The EVM design operates through a series of well-defined states controlled by a finite state machine (FSM). The FSM ensures that the voting process follows a strict and logical order — from system activation to final result display. The design uses switches and buttons to simulate physical components such as EVM power switch, candidate buttons, and result display control.

The main functionalities include:

* Powering ON/OFF the EVM system.
* Accepting candidate readiness signals.
* Allowing voters to cast votes for one of three candidates.
* Displaying vote counts and the winner based on user selection.
* Handling tie conditions and invalid result detection.

## 3. System Flow

The EVM system operates in a cyclic sequence of states, ensuring the voting process adheres to the rules of valid voting. The flow of operation is as follows:

1. 1. \*\*IDLE\*\* – The system remains in the idle state until the EVM is switched ON.
2. 2. \*\*WAITING\_FOR\_CANDIDATE\*\* – After the EVM is turned ON, it waits for a candidate readiness signal. If candidate\_ready signal is not asserted till 100 clock cycles, the state goes to voting\_process \_done or if the voting\_session\_done is asserted then the state goes to voting\_process \_done
3. 3. \*\*WAITING\_FOR\_CANDIDATE\_TO\_VOTE\*\* – Once a candidate is ready, the system enables voting. The candidate presses one of the vote buttons to cast their vote by using vote\_candidate\_x , if Vote\_candidate\_x not asserted till 100 clock cycles then the state goes to waiting\_for\_candidate . once Vote\_candidate\_x is asserted the state goes to CANDIDATE\_VOTED
4. 4. \*\*CANDIDATE\_VOTED\*\* – The system records the vote, updates the corresponding count, and signals that the candidate has voted. , in this state if candidate\_ready is asserted it goes to waiting\_for\_candidate\_to\_vote. Else nothing is asserted the state goes waiting\_for\_candidate.
5. 5. \*\*VOTING\_PROCESS\_DONE\*\* – After all candidates have voted or the voting session is marked complete, the system can display the winner or individual results.
6. 6. \*\*Return to IDLE\*\* – When the EVM is switched OFF, it resets all counts and returns to the idle state.

## 4. State Machine Design

The EVM system is implemented using a finite state machine (FSM) with five primary states. Each state performs a specific set of operations depending on the input control signals. State transitions occur on the rising edge of the clock or during reset events.

The five states are:  
• IDLE  
• WAITING\_FOR\_CANDIDATE  
• WAITING\_FOR\_CANDIDATE\_TO\_VOTE  
• CANDIDATE\_VOTED  
• VOTING\_PROCESS\_DONE

## 5. Behavioral Explanation

At the behavioral level, the system uses sequential and combinational logic blocks to manage operations. The sequential block handles state transitions, vote counting, and flag management, while the combinational blocks determine next states and output signals based on the current inputs.

• \*\*Sequential Block\*\* – Triggered on the rising edge of the clock or negative edge of reset. It updates state registers, vote counters, and button flags.  
• \*\*Combinational Block (Outputs)\*\* – Controls the output LEDs, display, and invalid conditions based on the current state.  
• \*\*Combinational Block (Next State Logic)\*\* – Determines transitions between states using input signals.

## 6. Port-Level Description

Below is the detailed description of each input and output port:

| **Signal** | **Direction** | **Width** | **Description** |
| --- | --- | --- | --- |
| clk | Input | 1 | System clock |
| rst | Input | 1 | Active-low reset |
| vote\_candidate\_1/2/3 | Input | 1 each | Vote buttons for three candidates |
| switch\_on\_evm | Input | 1 | Power-on/reset control for the machine |
| candidate\_ready | Input | 1 | Indicates EVM is ready for next voter |
| voting\_session\_done | Input | 1 | Marks the end of voting session |
| display\_results | Input | 2 | Selects which candidate’s votes to show |
| display\_winner | Input | 1 | Displays only the winning candidate |
| candidate\_name | Output | 2 | Encodes candidate ID being displayed |
| invalid\_results | Output | 1 | High if tie/invalid condition detected |
| results | Output | WIDTH (default 7) | Displays vote count or winner votes |
| voting\_in\_progress | Output | 1 | High when EVM is in voting state “waiting\_for\_candidate\_to\_vote” |
| voting\_done | Output | 1 | High when in voting\_process\_done state |

Internal Registers , Flags

| **Signal / Variable** | **Type** | **Width** | **Description** |
| --- | --- | --- | --- |
| WIDTH | Parameter | Default = 7 | Configurable width of vote count registers & result output |
| candidate\_1\_vote\_count | reg | WIDTH bits | Holds total votes for Candidate 1 |
| candidate\_2\_vote\_count | reg | WIDTH bits | Holds total votes for Candidate 2 |
| candidate\_3\_vote\_count | reg | WIDTH bits | Holds total votes for Candidate 3 |
| current\_state, next\_state | reg | 3 bits | FSM state encoding |
| vote\_candidate\_X\_flag | reg | 1 bit each | One-shot flags to register valid vote per candidate |

invalid\_results goes **high** when a tie or invalid voting outcome is detected.  
This includes:

All candidates have the same votes  
Any **two candidates tie for the highest** number of votes

| **C1** | **C2** | **C3** | **invalid\_results** |
| --- | --- | --- | --- |
| 5 | 5 | 5 | 1 (all equal) |
| 7 | 7 | 3 | 1 (tie for top) |
| 4 | 6 | 6 | 1 (tie for top) |
| 8 | 3 | 8 | 1 (tie for top) |
| 9 | 5 | 3 | 0 (clear winner: C1) |

DISPLAY RESULTS LOGIC

| **Inputs** | **Meaning** | **Output Behavior** |
| --- | --- | --- |
| display\_winner=1 | Show final winner | candidate\_name shows winner’s ID; results shows winner’s votes |
| display\_results=2’b00 | Show Candidate 1’s votes | Candidate name = 01, results = C1 count |
| display\_results=2’b01 | Show Candidate 2’s votes | Candidate name = 10, results = C2 count |
| display\_results=2’b10 | Show Candidate 3’s votes | Candidate name = 11, results = C3 count |
| Both inactive | Default = zeros |  |

Rst and Switch\_on\_evm serve **different functional scopes**:

| **Signal** | **Type** | **When used** | **Scope** | **Description** |
| --- | --- | --- | --- | --- |
| rst | Asynchronous active-low reset | Hardware / power-on reset | Global | Clears FSM and counters at system power-up or |
| switch\_on\_evm | Functional control input | User-controlled (EVM ON/OFF switch) | Operational | Acts as power switch; when OFF → forces FSM to IDLE and clears vote counters. |

## 7. Example Operation Flow

1. EVM is in IDLE state with all counters reset.  
2. User toggles the ‘switch\_on\_evm’ signal → system enters WAITING\_FOR\_CANDIDATE.  
3. Candidate asserts ‘candidate\_ready’ → system transitions to WAITING\_FOR\_CANDIDATE\_TO\_VOTE.  
4. Candidate presses one of the vote buttons (vote\_candidate\_X) → vote is recorded.  
5. System transitions to CANDIDATE\_VOTED, updates counter.   
6. After all votes are cast, ‘voting\_session\_done’ is asserted → system transitions to VOTING\_PROCESS\_DONE.  
7. Display results using ‘display\_results’ or show winner using ‘display\_winner’.  
8. Finally, toggle ‘switch\_on\_evm’ de-asserted → system resets to IDLE.

## 8. Design Assumptions

* It is assumed that , we will be having the list of voters and their voter I’d , so we will be knowing the number of candidates ., accordingly we will be updating the parameter width., still if vote is asserted after reaching the maximum count then the counter resets to ‘d0.
* SWITCH\_ON\_EVM , it is to on and off the EVM MACHINE , if that switch is de-asserted in between the process then the count values and all the process will be reset , it is not expected to give this switch in between the process., that signals is to ON and OFF the machine ., only when the EVM is switched on all the testing the reset or any other signal will be valid .
* Candidate\_ready has the priority among candidate\_ready and voting\_session\_done , if both are asserted at the same time
* We’ve used a 10Hz clock ., and the timeout for waiting\_for\_candidate and waiting\_for\_candidate\_to\_vote will be 100 clock cycles.
* When all the vote\_cantdidate\_1/2/3 are asserted at same time we have a priority ., 1>2>3 , so it’ll consider 1 as the valid vote
  + If 1 and 2 are asserted at same time , 1 will be considered
  + If 2 and 3 are asserted at same time , 2 will be considered
  + If 1 and 3 are asserted at same time , 1 will be considered

