**To design and implement a Land Rover FIGO FSM**

inputs:

clk (clock signal): Provides timing for the FSM operations.

reset (reset signal): Initializes the FSM to a known state.

move\_input (binary sequence for travel plans): Controls the transitions of the FSM.

Outputs:

location (current location of the Figo land-rover): Indicates the state of the FSM.

States:

Room0: Represents the initial state of the Figo land-rover.

Room1: Represents the state when Figo is in Room1.

Room2: Represents the state when Figo is in Room2.

Room3: Represents the state when Figo is in Room3.

Room4: Represents the state when Figo is in Room4.

Room5: Represents the state when Figo is in Room5.

Room6: Represents the state when Figo is in Room6.

Room7: Represents the state when Figo is in Room7.

Assumptions:

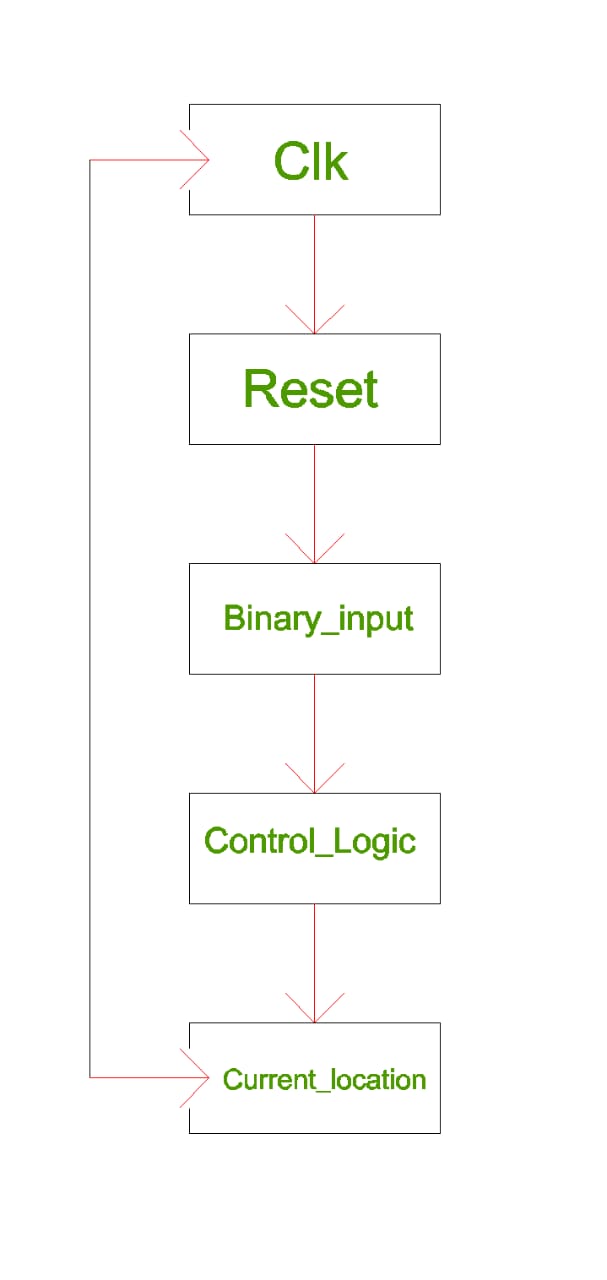
The FSM starts in Room0 as the initial state.

The FSM operates on positive-edge-triggered clock signal (clk).

The reset signal (reset) initializes the FSM to Room0.

The binary sequence for travel plans (move\_input) is provided to control Figo's movement.

BLOCK DIAGRAM:



Components of block Diagram:

clk: The clock signal used for synchronous operation.

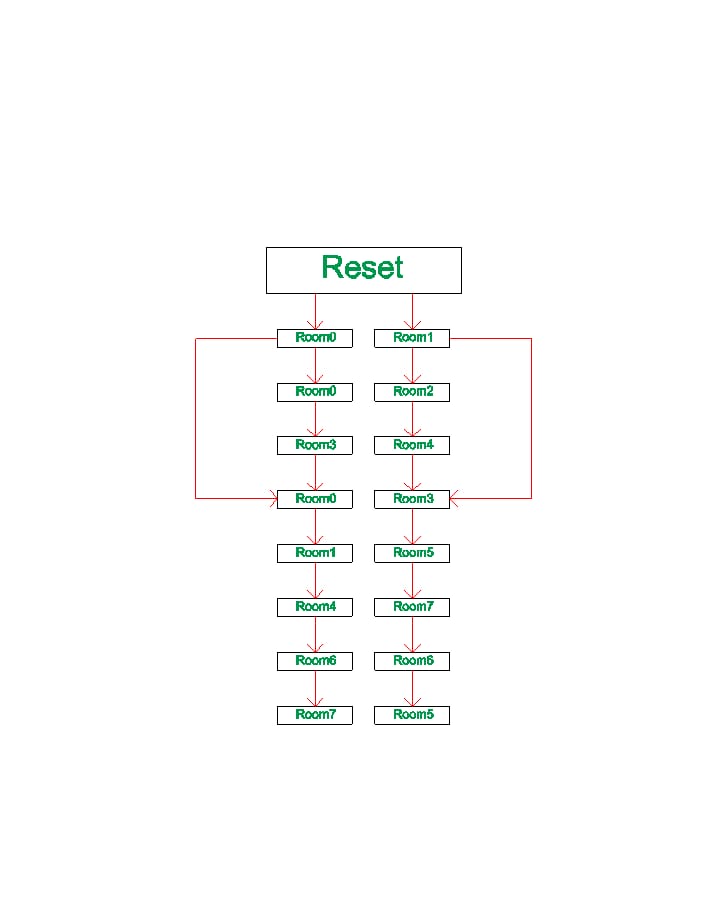
reset: The reset signal used to initialize or reset the state of the FSM.

binary\_input: The input signal used for the decision-making process in the FSM.

The "Control Logic" module represents the Verilog code implemented by the given module. It processes the inputs (clk, reset, binary\_input) and determines the next state of the FSM based on the current state and input values.

The output of the module is represented by current\_location, which is a 3-bit signal indicating the current location in the FSM.

FSM MODEL:



In this FSM model, each state is represented by a box (e.g., Room0, Room1, etc.), and the arrows indicate the transitions between states based on the input values (binary\_input). The initial state is Room0, which is set during reset.

The transitions are determined by the logic in the Verilog code. For example, if the current state is Room0 and the binary\_input is 1, the next state will be Room1. If the binary\_input is 0, the next state will remain as Room0.

The output of the FSM is represented by the current\_location signal, which corresponds to the current state of the FSM.

Approach to solve a problem:

Define the states: Identify the different locations (rooms) that need to be tracked. In the given code, the states are defined as `Room0`, `Room1`, `Room2`, `Room3`, `Room4`, `Room5`, `Room6`, and `Room7`. Assign binary values to represent each state.

Define the input signals: Identify the input signals that control the state transitions. In the given code, the input signal is `binary\_input`, which determines the next state based on its value.

Define the current state and next state variables: Declare variables to hold the current state (`current\_state`) and the next state (`next\_state`) of the FSM.

Implement the state transition logic: Use an `always` block to define the state transition logic based on the current state and input signals. Use a `case` statement to handle each state and determine the next state based on the `binary\_input` value.

Implement the current state update: Use another `always` block to update the `current\_state` variable based on the clock signal (`clk`) and reset signal (`reset`). On a positive edge of the clock or reset signal, update the `current\_state` with the `next\_state` value.

Implement the output signal: Use an `always` block to assign the output signal (`current\_location`) based on the current state. Use a `case` statement to map each state to its corresponding output value.

Compile and simulate: Use a Verilog simulator or development tool to compile and simulate the Verilog code. Provide appropriate input values to the `binary\_input`, `clk`, and `reset` signals and observe the changes in the `current\_location` output signal.

Test the FSM functionality: Test the FSM by providing different combinations of input values and verifying that the `current\_location` output signal updates correctly according to the state transitions.

.

FLOW:

1. Initialization:

- The `current\_state` and `next\_state` variables are initialized to `Room0` (000), representing the initial location.

- The `current\_location` output is set to `Room0`.

2. Clock and Reset Handling:

- On a positive edge of the `clk` signal or when `reset` is asserted, the `current\_state` is updated based on the `next\_state` value. This ensures synchronous state updates.

3. State Transition Logic:

- The `always` block sensitive to changes in `current\_state` and `binary\_input` determines the next state (`next\_state`) based on the current state and the value of `binary\_input`.

- A `case` statement is used to define the behavior for each state:

- For each state, the value of `binary\_input` is checked to determine the next state.

- If `binary\_input` is 1, the next state is set to a specific value depending on the current state.

- If `binary\_input` is 0, the next state is set to a different value depending on the current state.

4. Current Location Output:

- The `always` block sensitive to changes in `current\_state` assigns the `current\_location` output based on the current state.

- A `case` statement maps each state to its corresponding value for `current\_location`.

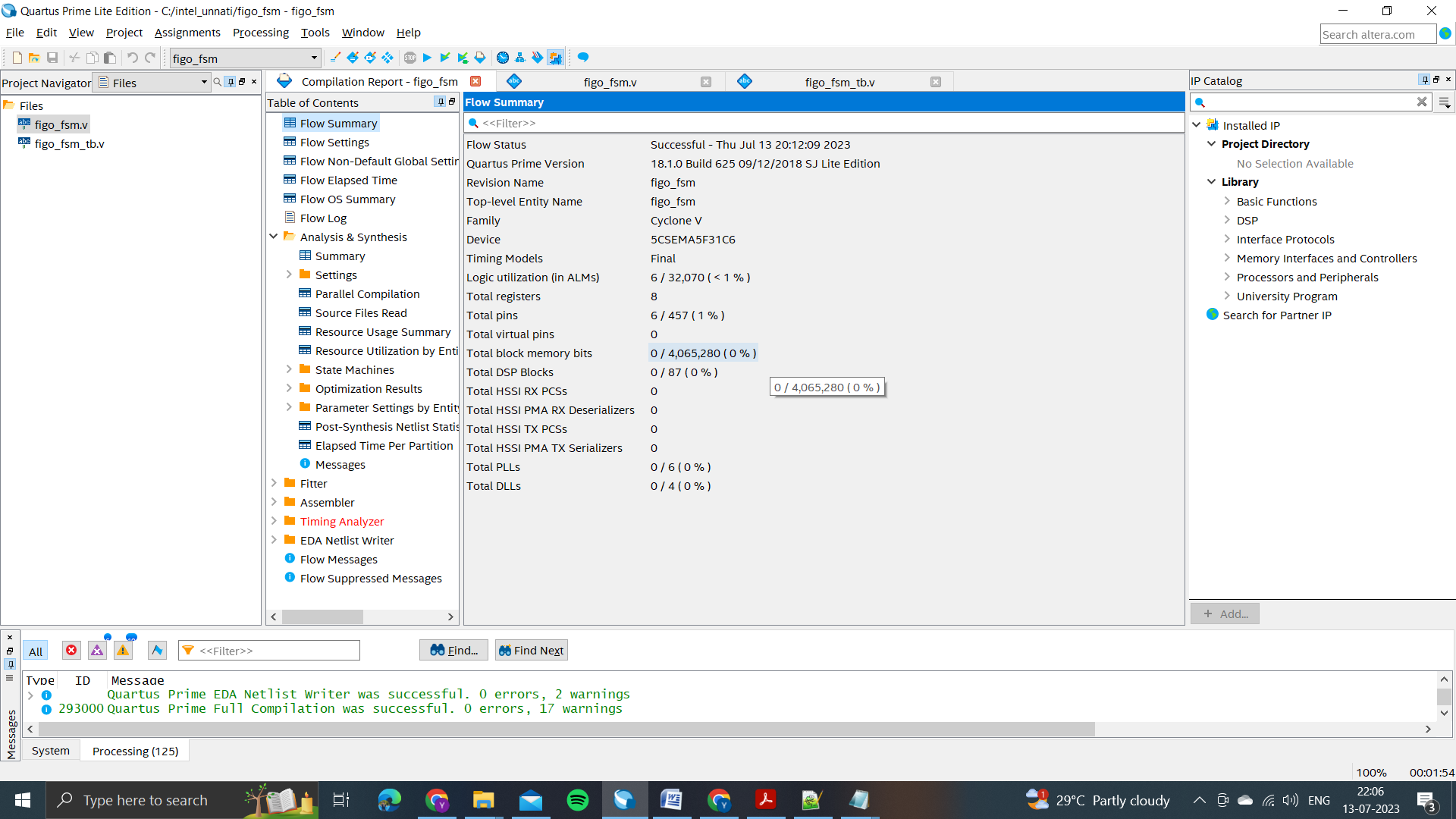
5. Simulation and Testing:

- The Verilog code is compiled and simulated using a Verilog simulator or development tool.

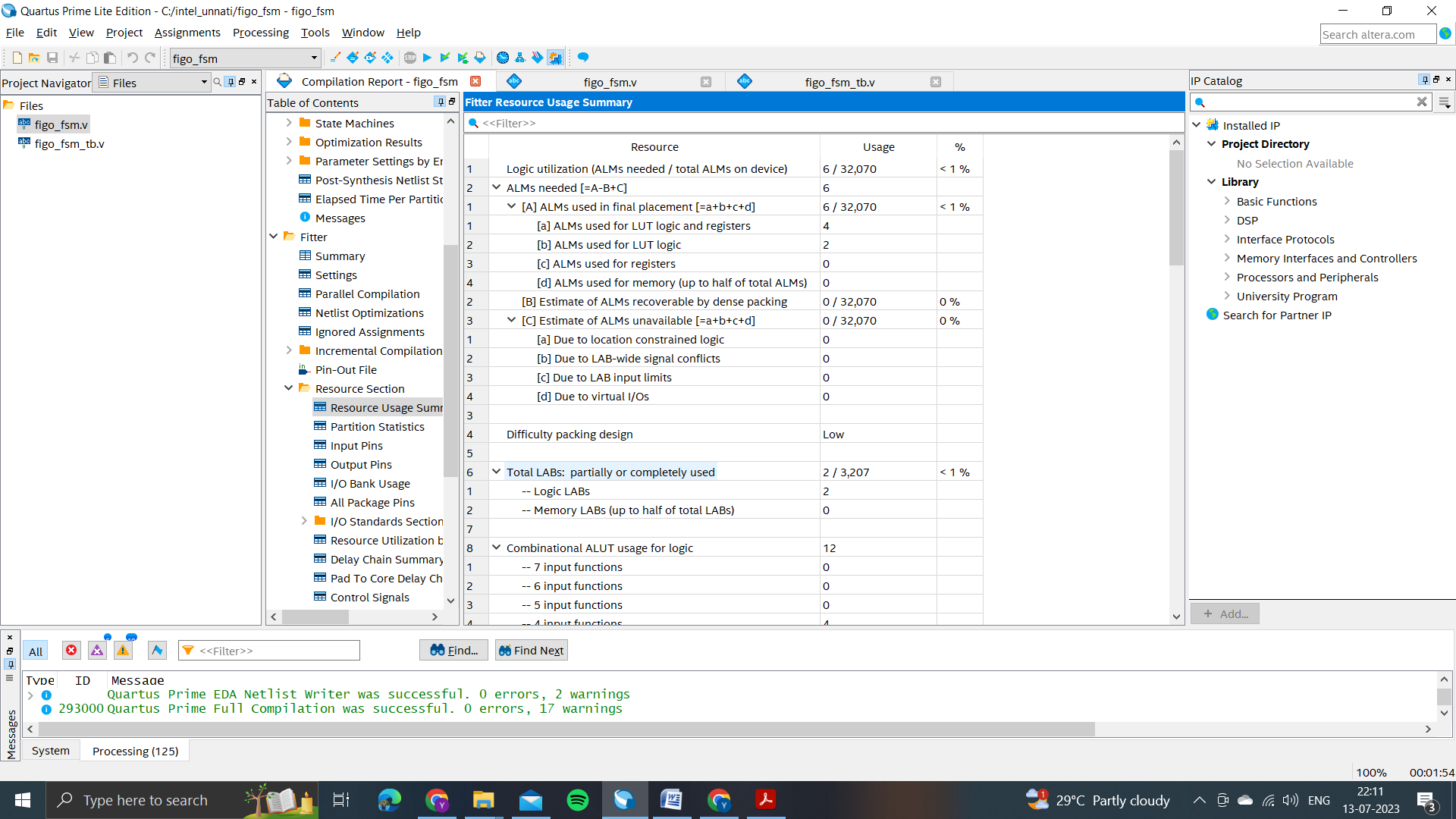
- Test vectors are provided for the `binary\_input`, `clk`, and `reset` signals to simulate different scenarios.

- The functionality of the FSM is verified by observing the changes in the `current\_location` output based on the state transitions and input values.

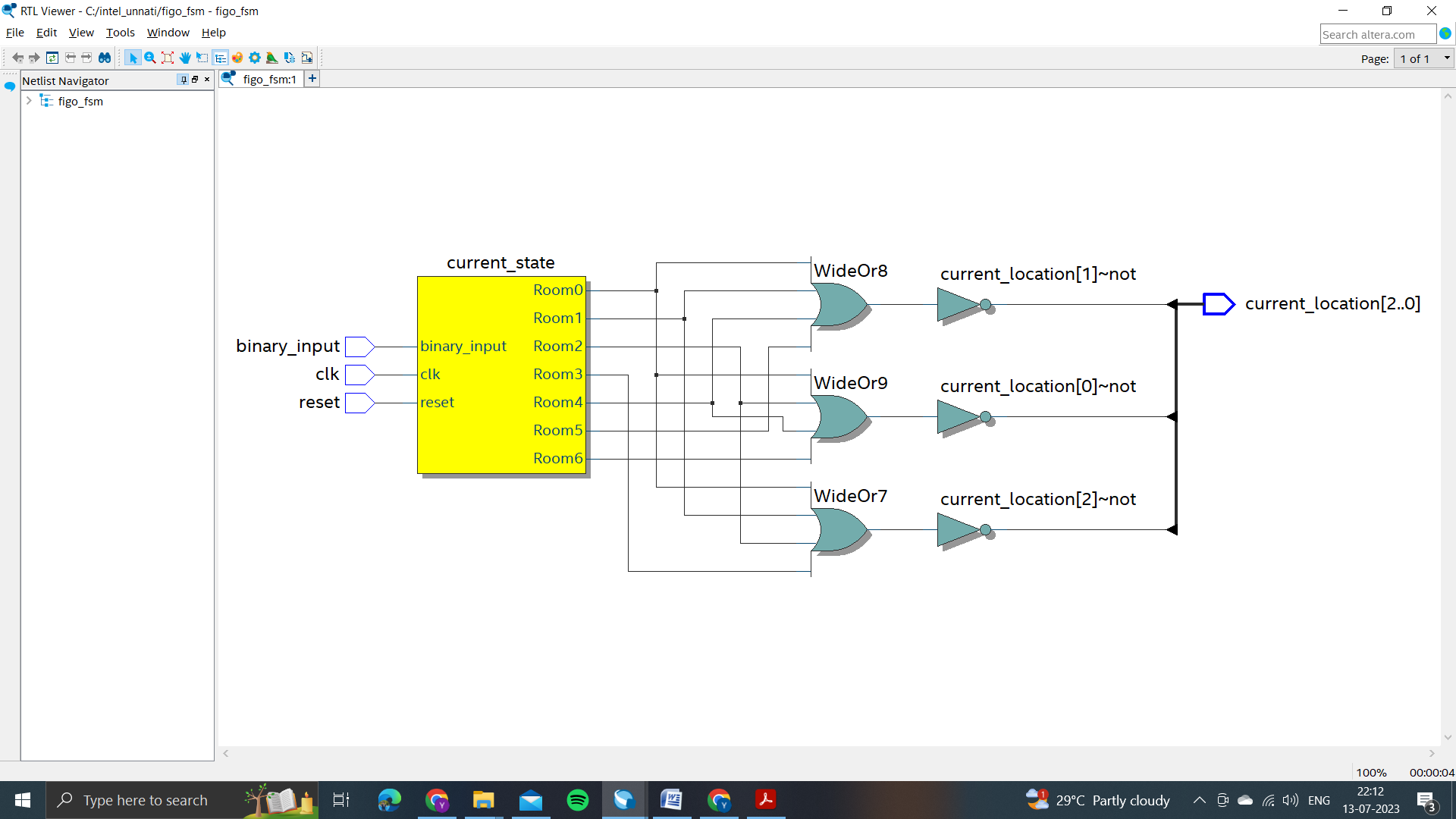
FLOW SUMMARY:



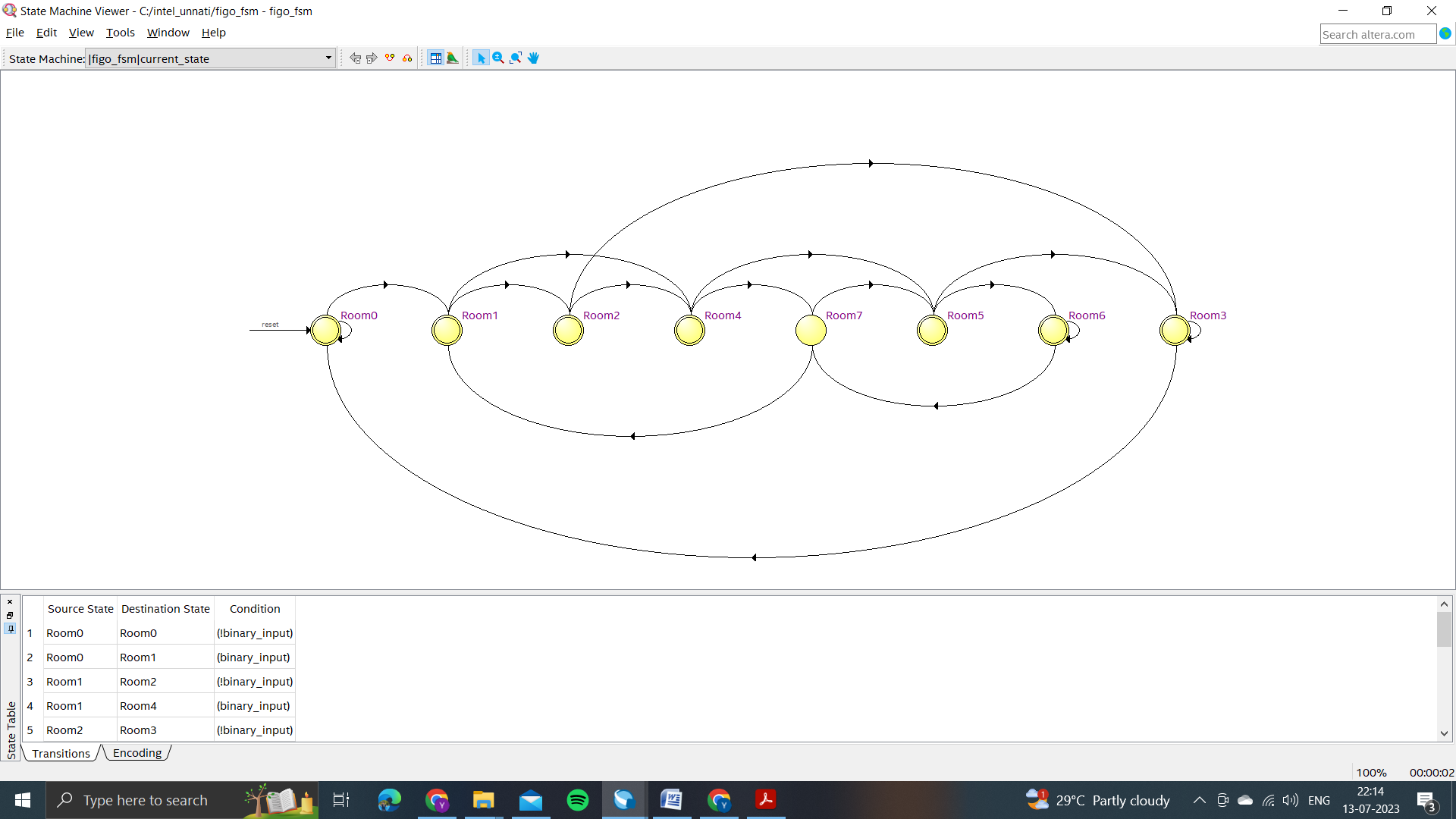
FITTER RESOURCE USAGE SUMMARY:



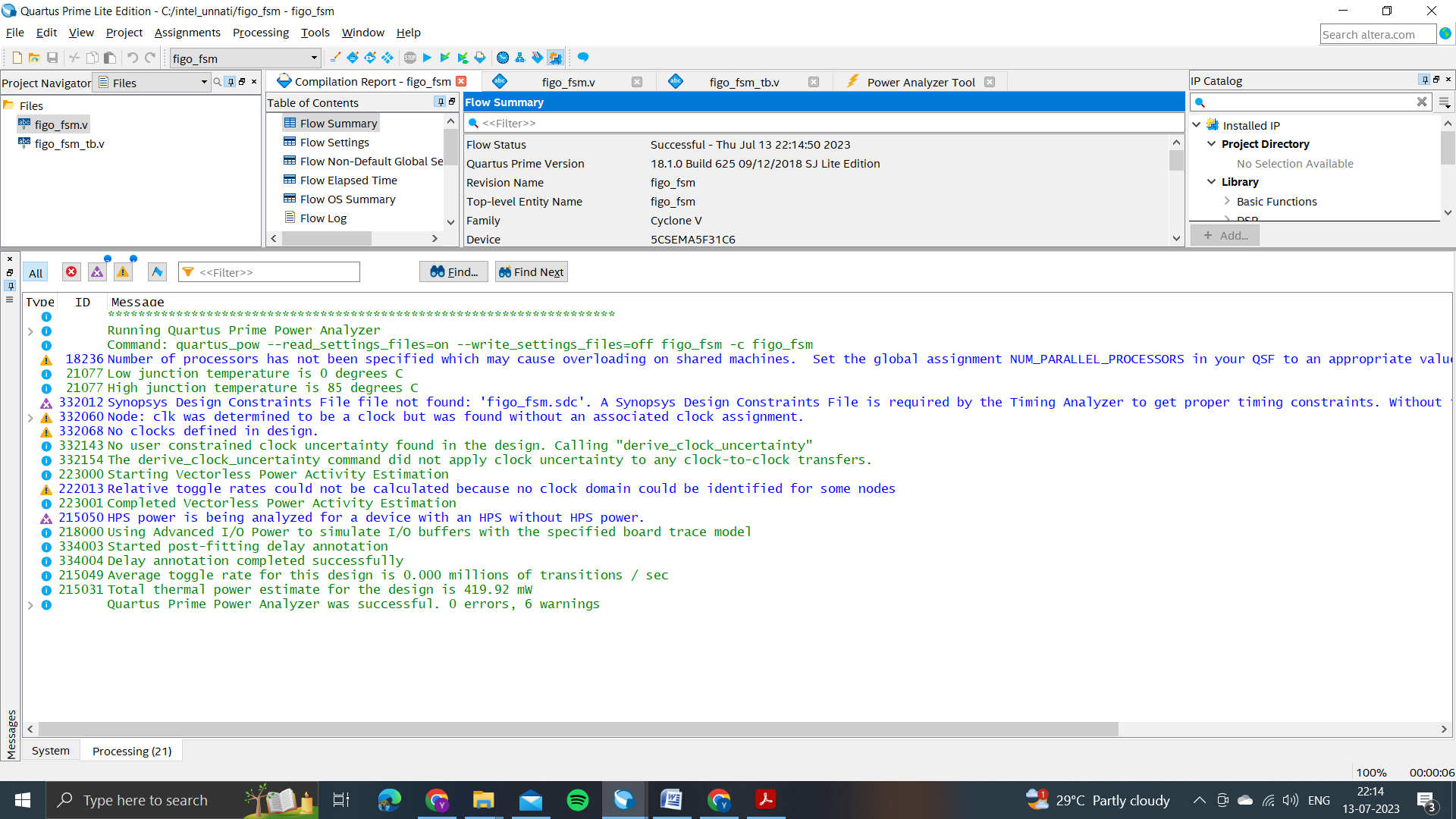
RTL VIEWER:



STATE MACHINE VIEWER:



POWER REPORT:



CONCLUSION:

The result of the project is a functional Finite State Machine (FSM) implemented in Verilog. The FSM takes inputs from clk, reset, and binary\_input signals and outputs the current\_location, which represents the current state of the FSM.

The Verilog code defines the FSM module, including the parameter declarations for the different rooms and the state variables. It uses sequential logic to update the current\_state based on the inputs and assigns the next\_state accordingly. The current\_location is updated based on the current\_state.

A test bench code was provided to test the functionality of the FSM. It includes various test cases that cover different scenarios of input values and expected outputs. The test bench stimulates the FSM with different input combinations and verifies that the current\_location output matches the expected values.

The project also involved organizing the files in a GitHub repository. The recommended folder structure includes folders for models, data, code, docs, and demo\_videos. These folders should be populated with the relevant files and documentation related to the project.

Overall, the project demonstrates the implementation of a Finite State Machine in Verilog, the testing of its functionality using a test bench, and the organization of project files in a GitHub repository. The provided documentation and video demonstrate the work done and provide a summary of the project's outcomes and results.