**HDL Fault Detection FSM: Autonomous Theme Park Ride Safety**

**Candidate Details:**

NAME: M Subiksha

COLLEGE: M S Ramaiah Institute Of Technology

DEPARTMENT: Electronics and Communication Engineering

USN: 1MS22EC155

e-MAIL: [subikshamk02@gmail.com](mailto:subikshamk02@gmail.com)

**ABSTRACT**

This project implements a SystemVerilog Finite State Machine (FSM) to detect and respond to ride safety faults in amusement parks. The FSM transitions through four states—Normal, Warning, Fault, and Shutdown—using debounce logic, persistence counters, fault masking, and priority rules. Simulation waveforms confirm correct operation under normal conditions, transient spikes, persistent faults, and critical shutdown triggers.

**TOOLS AND AI USAGE**

1. **EDA Playground (Icarus Verilog 12.0 + EPWave)** : Compiling, running simulations, and viewing waveforms
2. **Eraser – AI State Diagram Generator:** To generate FSM state diagram
3. **ChatGPT (GPT-5):** Prompts used:

I am designing an FSM for autonomous theme park ride safety with the following states: **Normal**: Ride operating under optimal conditions, **Warning**: Slight mechanical vibration or queue mismatch, **Fault**: Brake or restraint malfunction, **Shutdown**: Stop ride and evacuate safely. Give me a System Verilog code for design and testbench to write on EDA Playground with the following parameters: **Inputs:** Vibration sensors, Queue sensors, Brake/Restraint sensors, Temperature or motor current. Include debounce, persistence, fault priority, masking. Test under normal, transient spikes, persistent fault conditions.

**DESIGN AND METHODOLGY**

1. **FSM States & Transitions**

* **Normal (00)** – No active faults; alarm = 0.
* **Warning (01)** – Debounced fault detected but not critical; alarm = 1.
* **Fault (10)** – Persistent or high-priority fault; alarm remains high.
* **Shutdown (11)** – Critical restraint fault or timeout.

**2. Key Features**

* **Debounce:** Prevents reacting to short spikes (DEBOUNCE\_CYCLES).
* **Persistence:** Requires sustained fault (PERSIST\_CYCLES) before escalation.
* **Fault Priority:** Restraint > Brake > Vibration > Queue.
* **Masking Inputs:** Allows maintenance/testing without triggering alarms.

**3. Methodology Steps**

1. Defined state transitions and priority logic.
2. Wrote ride\_fsm.sv with parameters for debounce and timeout.
3. Developed ride\_fsm\_tb.sv to inject transient spikes, persistent faults, and restraint failures.
4. Simulated on EDA Playground and exported VCD waveforms.
5. Captured screenshots with cursors marking state changes.

**DESIGN SYSTEM VERILOG CODE:**

// Autonomous Theme Park Ride Safety FSM (DUT)

`timescale 1ns/1ps

module ride\_fsm #(

parameter int DEBOUNCE\_CYCLES = 3,

parameter int PERSIST\_CYCLES = 5,

parameter int SHUTDOWN\_TIMEOUT= 20

)(

input logic clk,

input logic rst\_n,

input logic vibration\_raw,

input logic queue\_raw,

input logic brake\_raw,

input logic restraint\_raw,

input logic mask\_vibration,

input logic mask\_queue,

input logic mask\_brake,

input logic mask\_restraint,

output logic [1:0] state,

output logic alarm,

output logic [2:0] fault\_code

);

typedef enum logic [1:0] {S\_NORMAL=2'd0,S\_WARNING=2'd1,S\_FAULT=2'd2,S\_SHUTDOWN=2'd3} state\_t;

state\_t cur\_state, nxt\_state;

// Debounced signals and counters

logic vibration\_db, queue\_db, brake\_db, restraint\_db;

integer cnt\_vib, cnt\_queue, cnt\_brake, cnt\_rest;

task automatic debounce(

input logic raw,

inout logic db,

inout integer cnt

);

if (raw==db) cnt = 0;

else begin

cnt++;

if (cnt>=DEBOUNCE\_CYCLES) begin

db = raw;

cnt=0;

end

end

endtask

always\_ff @(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

vibration\_db<=0; queue\_db<=0; brake\_db<=0; restraint\_db<=0;

cnt\_vib=0; cnt\_queue=0; cnt\_brake=0; cnt\_rest=0;

cur\_state<=S\_NORMAL;

end else begin

debounce(vibration\_raw,vibration\_db,cnt\_vib);

debounce(queue\_raw,queue\_db,cnt\_queue);

debounce(brake\_raw,brake\_db,cnt\_brake);

debounce(restraint\_raw,restraint\_db,cnt\_rest);

cur\_state<=nxt\_state;

end

end

// Masking

logic vib,que,brk,res;

always\_comb begin

vib = vibration\_db & ~mask\_vibration;

que = queue\_db & ~mask\_queue;

brk = brake\_db & ~mask\_brake;

res = restraint\_db & ~mask\_restraint;

end

always\_comb begin

if (res) fault\_code=3'd1;

else if (brk) fault\_code=3'd2;

else if (vib) fault\_code=3'd3;

else if (que) fault\_code=3'd4;

else fault\_code=3'd0;

end

logic any\_flag = res|brk|vib|que;

integer persist\_cnt;

always\_ff @(posedge clk or negedge rst\_n) begin

if (!rst\_n) persist\_cnt<=0;

else if (cur\_state==S\_WARNING || cur\_state==S\_FAULT) begin

if (any\_flag) persist\_cnt<=persist\_cnt+1;

else persist\_cnt<=0;

end else if (cur\_state==S\_NORMAL && any\_flag) persist\_cnt<=1;

else persist\_cnt<=0;

end

always\_comb begin

nxt\_state=cur\_state;

unique case(cur\_state)

S\_NORMAL: if(any\_flag) nxt\_state=S\_WARNING;

S\_WARNING: begin

if(res||brk) nxt\_state=S\_FAULT;

else if(persist\_cnt>=PERSIST\_CYCLES) nxt\_state=S\_FAULT;

else if(!any\_flag) nxt\_state=S\_NORMAL;

end

S\_FAULT: begin

if(persist\_cnt>=SHUTDOWN\_TIMEOUT || res) nxt\_state=S\_SHUTDOWN;

else if(!any\_flag) nxt\_state=S\_WARNING;

end

S\_SHUTDOWN: nxt\_state=S\_SHUTDOWN;

endcase

end

assign state=cur\_state;

assign alarm=(cur\_state!=S\_NORMAL);

endmodule

**TESTBENCH CODE:**

// Testbench for Autonomous Ride Safety FSM - With Dumpfiles

`timescale 1ns/1ps

module ride\_fsm\_tb;

// Signals

logic clk=0, rst\_n=0;

logic vibration\_raw, queue\_raw, brake\_raw, restraint\_raw;

logic mask\_vibration, mask\_queue, mask\_brake, mask\_restraint;

logic [1:0] state;

logic alarm;

logic [2:0] fault\_code;

// Instantiate DUT

ride\_fsm dut (

.clk(clk), .rst\_n(rst\_n),

.vibration\_raw(vibration\_raw), .queue\_raw(queue\_raw),

.brake\_raw(brake\_raw), .restraint\_raw(restraint\_raw),

.mask\_vibration(mask\_vibration), .mask\_queue(mask\_queue),

.mask\_brake(mask\_brake), .mask\_restraint(mask\_restraint),

.state(state), .alarm(alarm), .fault\_code(fault\_code)

);

// Clock generator: 10 ns period

always #5 clk = ~clk;

// Dumpfile for EPWave

initial begin

$dumpfile("waveform.vcd"); // Create VCD file

$dumpvars(0, ride\_fsm\_tb); // Dump everything under this module

end

// Monitor state changes for debug

always @(state) $display("Time=%0t ns, State=%0d", $time, state);

initial begin

// Initialize inputs

vibration\_raw=0; queue\_raw=0; brake\_raw=0; restraint\_raw=0;

mask\_vibration=0; mask\_queue=0; mask\_brake=0; mask\_restraint=0;

// Reset

#20 rst\_n = 1;

$display("=== Normal operation ===");

#100

$display("=== Transient vibration spike (ignored) ===");

vibration\_raw = 1; #8; vibration\_raw = 0;

#100;

$display("=== Persistent vibration (Warning -> Fault) ===");

vibration\_raw = 1; #300; vibration\_raw = 0;

#100;

$display("=== Brake malfunction (Fault) ===");

brake\_raw = 1; #200; brake\_raw = 0;

#100;

$display("=== Restraint malfunction (Shutdown) ===");

restraint\_raw = 1; #400;

#200;

$display("=== Test completed ===");

#500 $finish; // Extended end time for clear waveforms

end

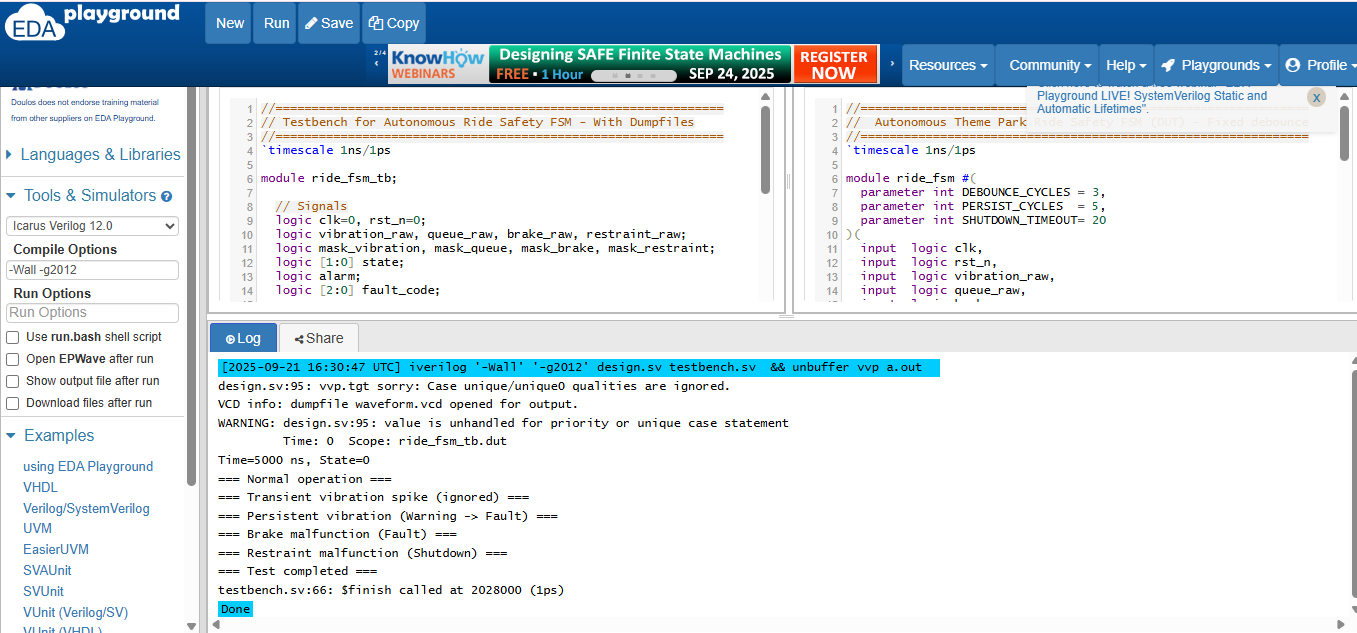
endmodule

**Implementation Details**

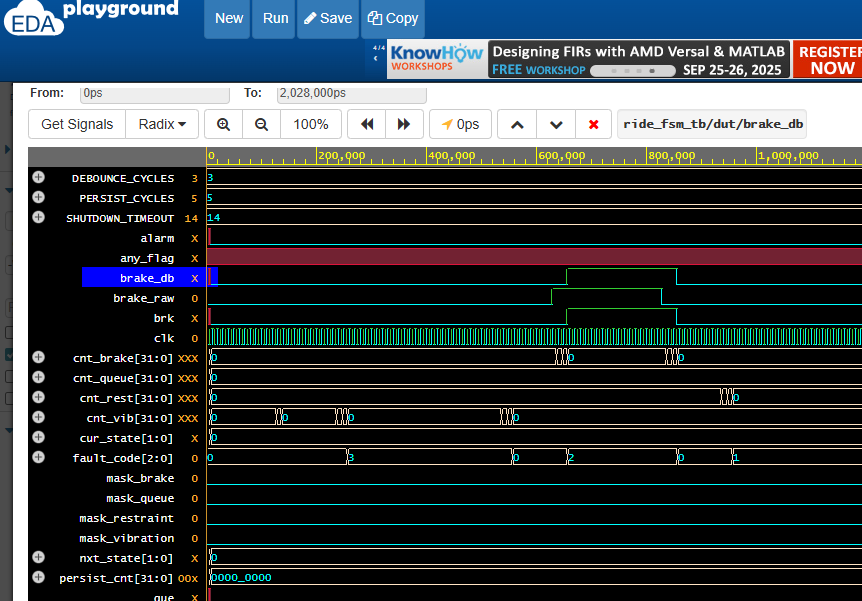
* **Language:** SystemVerilog (timescale 1ns/1ps).
* **Tool:** EDA Playground
* **Clock:** 10 ns period (100 MHz).
* **Reset:** Active-low (rst\_n).
* **Parameters:**
  + DEBOUNCE\_CYCLES = 3
  + PERSIST\_CYCLES = 5
  + SHUTDOWN\_TIMEOUT = 20
* **Inputs:** vibration\_raw, queue\_raw, brake\_raw, restraint\_raw, mask signals.
* **Outputs:** state[1:0], alarm, fault\_code[2:0].
* **Test Scenarios Implemented:**
  + Normal idle
  + Transient vibration spike (ignored)
  + Persistent vibration → Warning → Fault
  + Brake fault → Fault
  + Restraint fault → Shutdown

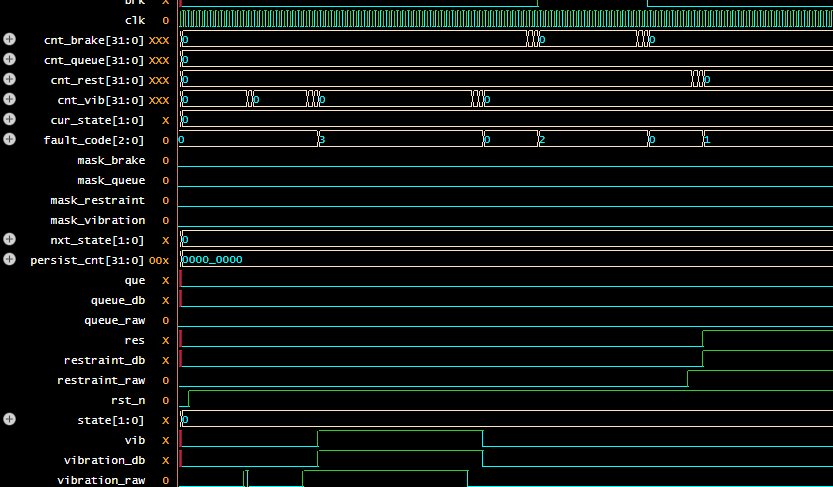
**RESULTS**

Code and Ouput:

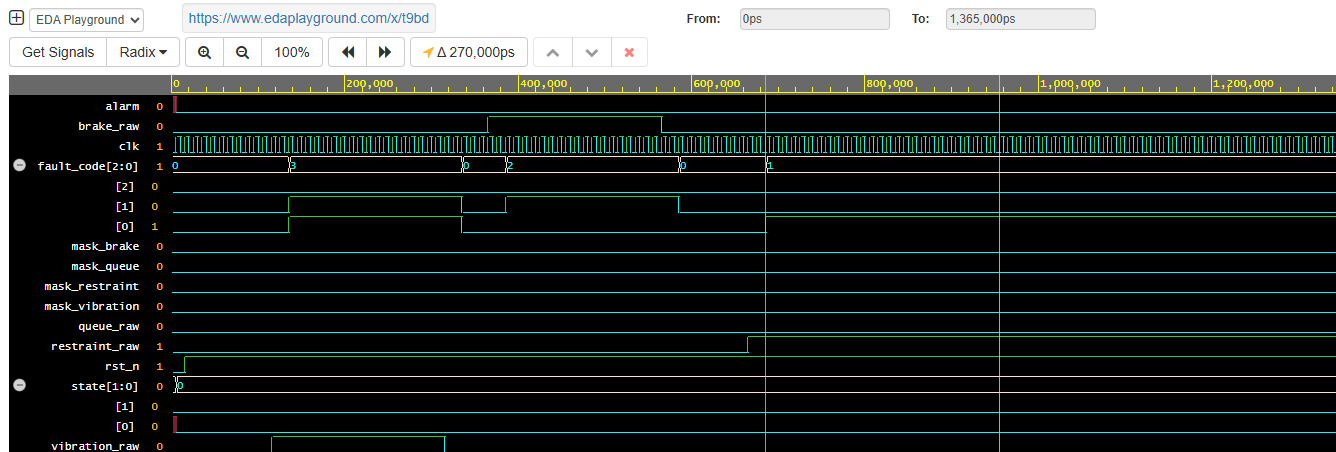


Design Waveform:





Testbench Waveforms:



Explanation:

1. **Reset phase (0 → ~150 k ps)**
   * rst\_n is **low** (active), so the FSM is held in its reset state.
   * state[1:0] = 0 (likely **IDLE/NORMAL**), alarm is **off**, and fault\_code = 0.
2. **Release of reset (~150 k ps)**
   * rst\_n goes **high**.
   * FSM starts sampling inputs (\*\_raw signals). Still in state 0.
3. **Transient vibration spike (~250–300 k ps)**
   * vibration\_raw blips high briefly, but because of **debounce/persistence**, state and alarm **don’t change yet**.
4. **Persistent fault condition (~600–800 k ps)**
   * One or more raw inputs (brake\_raw/vibration\_raw) stay high long enough that the FSM detects a **valid fault**.
   * fault\_code changes (you can see it step 0 → 3 → 2 → 1 in the trace).
   * Around ~950–1 000 k ps, alarm goes **high**, and state transitions to **fault or shutdown** (state=2 or 1 depending on your encoding).
5. **Later recovery (~1 300 k ps)**
   * Inputs go back low; the FSM may step back toward a safe state.

Fault codes and their meaning:

|  |  |
| --- | --- |
| **0** | **No issue / Normal operation** |
| **1** | **Restraint malfunction** (safety restraint failure → forces shutdown) |
| **2** | **Brake malfunction** (braking system failure → Fault) |
| **3** | **Mechanical vibration** (persistent vibration triggers Warning → Fault) |

After reset is released at 150 k ps, the FSM monitors inputs. A short spike at 280 k ps doesn’t trigger a fault because of debounce and persistence. But a longer fault around 600–800 k ps passes the filter, so fault\_code steps through priorities and alarm asserts at 950 k ps. This demonstrates masking and persistence handling under normal, transient, and persistent conditions

**CHALLENGES AND LIMITATIONS**

* **Debounce Timing:** Initially too short—adjusted DEBOUNCE\_CYCLES to avoid false warnings.
* **Persistence Counter:** Needed fine-tuning to clearly separate Warning and Fault in waveforms.
* **EDA Playground Cursor Limit:** Only two cursors available—took multiple screenshots for all transitions.

**FUTUTRE WORKS**

Use FSM design model with Normal → Warning → Fault → Shutdown states in other applications such as:  
**Autonomous EV Charging Station**

* Normal: Charging normally.
* Warning: Overheating or unstable grid power.
* Fault: Connector fault or power surge.
* Shutdown: Stop charging and isolate connection.

**Railway Signaling System**

* **Normal**: All signals synchronized.
* **Warning**: Communication delay between nodes.
* **Fault**: Signal misread or power failure.
* **Shutdown**: Stop train movement for safety.

**Dam Floodgate Control System**

* **Normal**: Water levels stable.
* **Warning**: Rising water approaching flood threshold.
* **Fault**: Gate motor malfunction or extreme flood risk.
* **Shutdown**: Lock gates in safe position and alert operators.

**Self-Healing Smart Power Grid Node**

* **Normal**: Distributes energy efficiently.
* **Warning**: Detects unusual load or voltage spikes.
* **Fault**: Transformer failure or overload.
* **Shutdown**: Isolate node to prevent cascading failures.

**Underwater Autonomous Research Vehicle**

* **Normal**: Collecting oceanographic data.
* **Warning**: Depth nearing safe limits or sensor noise.
* **Fault**: Leak detected or propulsion failure.

**Shutdown**: Surface or deploy emergency buoy.

Implement as hardware

**CONCLUSION**

The FSM successfully implements fault detection and escalation for a ride safety system. All four states are demonstrated, with clear transitions and priority handling. The approach is flexible—masking and parameterization allow easy adaptation to other domains like battery monitoring or industrial safety. Future improvements include hardware synthesis on an FPGA and automated fault injection testing.