Automatic Environmental Control System for Food Raw Material Storage Rooms Based on Digital Multi-Sensors and Integrated Actuators

1st Mochamad Rafly Firmansyah
Department of Instrumentation
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
2042241130@student.its.ac.id
https://orcid.org/0009-0002-2456-0419

2nd Fitri Adi I., S.T., M.T.
Department of Instrumentation
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
fiskandarianto@gmail.com
https://orcid.org/0009-0001-9106-2201

3rd Ir. Dwi Oktavianto W. N., S.T., M.T.
Department of Instrumentation
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
goldcells@gmail.com
https://orcid.org/0000-0002-6781-0732

***Abstract*—Dry raw materials such as wheat flour, grains, spices, nuts, sugar, and powdered milk require stable environmental conditions during storage. Changes in temperature, humidity, airflow, and air quality can trigger mold, clumping, quality degradation, and material spoilage. Many storage rooms in the food industry still rely on manual monitoring, so responses to environmental changes are often delayed and less accurate. To maintain dry material quality consistently, an automatic environmental control system is needed that is capable of monitoring physical parameters in real-time using digital multi-sensors. The system then adjusts room conditions through actuators such as exhaust fans, air dampers, humidifiers, dehumidifiers, and cooling systems. This approach allows the storage room to remain at optimal conditions and supports the smooth running of the manufacturing process.**

***Index Terms*—Environmental Control, Multi-Sensor, FPGA, FSM, Industrial Automation.**

I. Introduction

DRY raw materials such as wheat flour, grains, spices, nuts, sugar, and powdered milk require stable environmental conditions during storage. Changes in temperature, humidity, airflow, and air quality can trigger mold, clumping, quality degradation, and material spoilage. Many storage rooms in the food industry still rely on manual monitoring, so responses to environmental changes are often delayed and less accurate.

To maintain dry material quality consistently, an automatic environmental control system is needed that is capable of monitoring physical parameters in real-time using digital multi-sensors. The system then adjusts room conditions through actuators such as exhaust fans, air dampers, humidifiers, dehumidifiers, and cooling systems. This approach allows the storage room to remain at optimal conditions and supports the smooth running of the manufacturing process.

II. Sensors in Physical Quantity Measurement

A. Temperature Sensor (DS18B20)

Physical Quantity: Measuring physical quantity in degrees (°). Reference: Based on the IEEE journal titled "Utilization of DS18B20 Temperature Sensor for Predictive Maintenance of Reciprocating Compressor". In this journal, DS18B20 is used to read compressor temperature, where the measurement result table displays the actual temperature considered as the baseline (normal condition) for machine conditions without problems, and high temperature is considered an indication of abnormality [1].

- 0 (Abnormal): $\pm 41.125°\text{C}$ – $\pm 97.587°\text{C}$
- 1 (Normal): $\pm 19.8°\text{C}$ – $\pm 20.2°\text{C}$

B. Humidity Sensor (SHT31)

Physical Quantity: Measuring relative humidity (%RH). Reference: Based on the IEEE journal "A Novel LoRaWAN-based Real-time Traffic Analysis Approach for Vehicle Congestion Estimation", the SHT31 sensor is used as a humidity sensor to measure weather parameters [2].

- 0 (Abnormal): $\geq 69\%$ Considered abnormal because it includes external factors.
- 1 (Normal): $\leq 68\%$ Not associated as a parameter that worsens the condition.

C. VOC Sensor (SGP30)

Physical Quantity: Measuring VOC (Volatile Organic Compounds) concentration in mg/m^3. Reference: Based on the IEEE journal "Indoor Air Quality Monitors using IoT Sensors and LPWAN", VOC is measured using the SGP30 sensor [3].

- 0 (Abnormal): VOC > 150
- 1 (Normal): VOC ≤ 50

D. Dust/Particulate Sensor (PMS5003)

Physical Quantity: Particulate matter concentration (PM2.5 and PM10) in μg/m^3. Reference: Based on the IEEE journal titled "Air Pollutant Detection System Utilizing an IoT-based Electronic Nose for Air Purifier", the PMS5003 sensor is used to detect particulate matter PM2.5 and PM10 [4].

- 0 (Abnormal): PM10 ≥ 25 μg/m^3, PM2.5 ≥ 10 μg/m^3
- 1 (Normal): PM10 < 25 μg/m^3, PM2.5 < 10 μg/m^3

E. Airflow Sensor (MPXV7002DP)

Physical Quantity: Measuring airflow speed in m/s. Reference: Based on the IEEE journal "Advancements in Microcontroller Technology for Wind Speed Measurement in Wind Tunnels", airflow is measured as wind speed [5].

- 0 (Abnormal): High and unstable ≥ 3.5 m/s, ≥ 4.2 m/s
- 1 (Normal): Low and stable 1.8 m/s, 2.6 m/s

F. Light Intensity Sensor (BH1750)

Physical Quantity: Measuring light intensity (illuminance) in lux. Reference: Based on the journal "Prototype Design of Automatic Light Intensity Control in Smart Green House", the BH1750 sensor is used to measure light intensity in a greenhouse system [6].

- 0 (Abnormal): Lux Condition < 500
- 1 (Normal): Lux Condition ≥ 500

III. Actuators in Physical Quantity Measurement

A. Exhaust Fan

IEEE PAPER: Prototype Design of Automatic Switching Speed of Exhaust Fan For Air Quality Control Based On IoT [7]. PHYSICAL ACTION: Regulates the exhaust of dirty air mechanically by changing the exhaust fan rotation speed based on CO_2 concentration (ppm).

- 0 (Idle/Off): No air exhaust 0 mA
- 1 (Active/On): 500 – 600 ppm – speed 1 (1.3-1.4 mA), 600 – 700 ppm – Speed 2 (1.8 mA).

B. Inline Duct Fan

IEEE PAPER: VENTI: experimental controller for inline duct fan [8]. PHYSICAL ACTION: As a ventilation actuator to regulate airflow by turning the fan on/off and adjusting fan speed.

- 0 (Idle/Off): Duct fan does not turn on because the vapor pressure value is within a threshold that does not trigger control.
- 1 (Active/On): Duct fan turns on, fan speed is active because the vapor pressure value exceeds the control threshold.

C. Humidifier

IEEE PAPER: Design and Implementation of a Wireless Control Intelligent Humidifier [9]. PHYSICAL ACTION: In this system, a relay is used to set the working gear of the humidification equipment. When the system runs normally, the humidifier gear will be set (automatically or manually) to increase room air humidity based on humidity data measured by the sensor.

- 0 (Abnormal): Humidifier Off / gear = 0
- 1 (Normal): Humidifier active (gear > 0), relay sets gear according to auto/manual mode to maintain room humidity.

D. Dehumidifier

IEEE PAPER: Automatic Usage of IoT-Based Dehumidifiers in High-Humidity Spaces [10]. PHYSICAL ACTION: The dehumidifier is activated automatically when the room humidity level is higher than the predetermined limit, and deactivated when the humidity is below that limit.

- 0 (Idle/Off): Dehumidifier Off when humidity is below threshold.
- 1 (Active/On): Dehumidifier On when humidity exceeds threshold.

E. Cooling System

IEEE PAPER: Design and Performance Evaluation of LH2 Cooling System for HTS Motor Electric Propulsion Platform [11]. PHYSICAL ACTION: The cooling system works by circulating Liquid Hydrogen (LH2) through the cooling system to lower the temperature of the HTS motor components. The cooling system keeps the temperature low during operation.

- 0 (Idle/Off): Cooling system does not circulate LH2
- 1 (Active/On): Cooling system actively circulates LH2

F. LED Light System

IEEE PAPER: A Spectrally Tunable Smart LED Lighting System With Closed-Loop Control [12]. PHYSICAL ACTION: LED works as a lighting actuator by emitting light intensity (luminous flux) which is increased or decreased through a closed-loop control system to reach the target illuminance.

- 0 (Idle/Off): LED is in a condition emitting no light or below operating level.
- 1 (Active/On): LED actively turns on, emitting luminous flux as needed by the system to reach target illuminance.

IV. State Logic

A. Idle State

All sensors = 1 (normal). All actuators OFF (A1-A6 = 0). System standby, no corrective action. Example: Row 1 in truth table.

Table I. Truth Table

No	S1	S2	S3	S4	S5	S6	A1	A2	A3	A4	A5	A6	State Description
1	1	1	1	1	1	1	0	0	0	0	0	0	All normal / IDLE
2	0	1	1	1	1	1	0	0	0	0	1	0	High Temp → Cooling ON
3	1	0	1	1	1	1	0	0	1	0	0	0	High Humidity → Dehumidifier ON
4	1	1	0	1	1	1	1	0	0	0	0	0	High VOC → Exhaust Fan ON
5	1	1	1	0	1	1	1	0	0	0	0	0	High Dust → Exhaust Fan ON
6	1	1	1	1	0	1	0	0	0	1	0	0	Low Airflow → Inline Duct Fan ON
7	1	1	1	1	1	0	0	0	0	0	0	1	Low Light → LED System ON
8	0	1	0	1	1	1	1	0	0	0	1	0	High Temp + High VOC
9	0	1	1	0	1	1	1	0	0	0	1	0	High Temp + High Dust
10	0	1	1	1	0	1	0	0	0	1	1	0	High Temp + Low Airflow
11	0	1	1	1	1	0	0	0	0	0	1	1	High Temp + Low Light
12	1	0	0	1	1	1	1	0	1	0	0	0	High RH + High VOC
13	1	0	1	0	1	1	1	0	1	0	0	0	High RH + High Dust
14	1	0	1	1	0	1	0	0	1	1	0	0	High RH + Low Airflow
15	1	0	1	1	1	0	0	0	1	0	0	1	High RH + Low Light
16	1	1	0	0	1	1	1	0	0	0	0	0	High VOC + High Dust
17	1	1	0	1	0	1	1	0	0	1	0	0	High VOC + Low Airflow
18	1	1	0	1	1	0	1	0	0	0	0	1	High VOC + Low Light
19	1	1	1	0	0	1	1	0	0	1	0	0	High Dust + Low Airflow
20	1	1	1	0	1	0	1	0	0	0	0	1	High Dust + Low Light
21	1	1	1	1	0	0	0	0	0	1	0	1	Low Airflow + Low Light
22	0	0	1	1	1	1	0	0	1	0	1	0	High Temp + High RH
23	0	0	0	1	1	1	1	0	1	0	1	0	High Temp + High RH + High VOC
24	0	0	1	0	1	1	1	0	1	0	1	0	High Temp + High RH + High Dust
25	0	0	1	1	0	1	0	0	1	1	1	0	High Temp + High RH + Low Airflow
26	0	0	1	1	1	0	0	0	1	0	1	1	High Temp + High RH + Low Light
27	0	1	0	0	1	1	1	0	0	0	1	0	High Temp + High VOC + High Dust
28	0	1	0	0	0	1	1	0	0	1	1	0	High Temp + VOC + Dust + Low Airflow
29	0	1	1	0	0	0	1	0	0	1	1	1	High Temp + Dust + Airflow + Low Light
30	1	0	0	0	1	1	1	0	1	0	0	0	High RH + High VOC + High Dust
31	1	0	0	0	0	1	1	0	1	1	0	0	High RH + VOC + Dust + Low Airflow
32	0	0	0	0	0	0	1	0	1	1	1	1	All sensors abnormal → all protection actuators ON

B. Single Sensor Abnormal – Minimal Alert

Only 1 sensor = 0 (abnormal). Activation occurs on the actuator directly related to that sensor. System performs minor correction. Case example: High Temp – Cooling system ON; High VOC – Exhaust Fan ON; etc. Example: Rows 2-7 truth table.

C. Multiple Sensors Abnormal – Moderate Correction

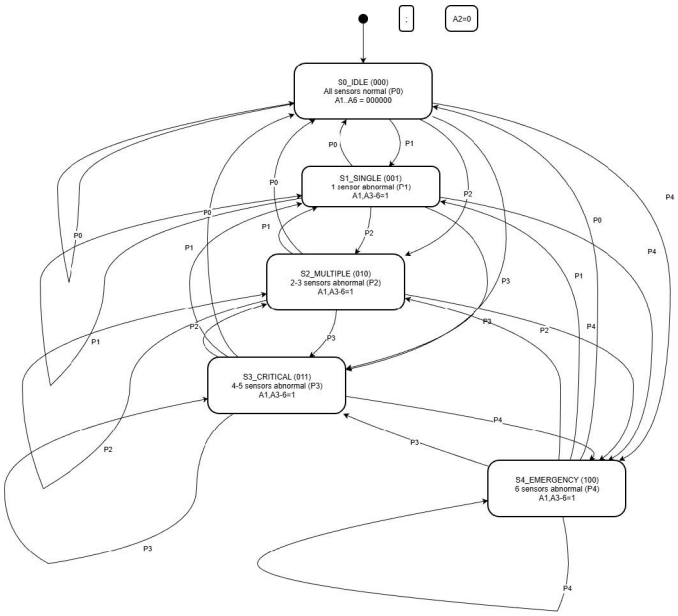
There are 2-3 abnormal sensors. Several actuators activate simultaneously to perform combined correction. Correction intensity is greater than single issue. Example: Rows 8-20 truth table.

D. Critical Condition – Many Sensors Problematic

4-5 sensors = 0 (abnormal). Almost all actuators ON. System tries to maintain room conditions within safe limits. Example: Rows 21-31 truth table.

E. Emergency Mode – All Sensors Abnormal

All S1-S6 = 0. All protection actuators activated (A1-A6 = 1). System works in emergency mode. Example: Row 32 truth table.



V. State Definitions and Transitions

A. State Definitions

- S0 – IDLE State: All sensors normal (S1–S6 = 1), all actuators OFF.
- S1 – Single Sensor Abnormal: Exactly 1 sensor value 0 (abnormal). Only the actuator related to that sensor is active.
- S2 – Multiple Sensors Abnormal (Moderate): There are 2–3 abnormal sensors. Several actuators active simultaneously.

- S3 – Critical Condition: There are 4–5 abnormal sensors. Almost all protection actuators active.
- S4 – Emergency Mode: All sensors abnormal (S1–S6 = 0). All protection actuators active.

B. Inter-State Transition Rules

From S0 (IDLE): If 1 sensor becomes 0 $\rightarrow$ S1; If 2–3 sensors $\rightarrow$ S2; If 4–5 sensors $\rightarrow$ S3; If all sensors 0 $\rightarrow$ S4. From S1, S2, S3, S4 follows the logic of the number of abnormal sensors increasing or decreasing, and reverse transition (Recovery) occurs if conditions improve.

C. Actuator Output per State (Moore Model)

- S0: A1..A6 = 0
- S1: Actuator ON only corresponding to sensor.
- S2: Combination of several actuators ON.
- S3: Majority of protection actuators ON.
- S4: All protection actuators ON (A1, A3, A4, A5, A6 = 1).

VI. State Memory Implementation

A. State Bit Quantity

To store 5 FSM states physically: $[\log_2(5)] = 3$ bits. Thus, 3 Flip-Flops (Q2,Q1,Q0) are needed. Total combination 8 states, 5 used, 3 unused.

B. Flip-Flop Type

Using D-Type Flip-Flop (D-FF) because it maps directly to FPGA architecture and has the simplest excitation equation ($D = Q_{next}$).

C. Matrix Representation

FSM uses 3 Flip-Flops, state represented by column vector $|\mathbf{Q}\rangle = Q_2Q_1Q_0$. Category of abnormal sensor count:

- P0: N = 0 (IDLE)
- P1: N = 1 (Single)
- P2: N = 2–3 (Multiple)
- P3: N = 4–5 (Critical)
- P4: N = 6 (Emergency)

Transition Matrix M:

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Relationship: $|\mathbf{Q}^+\rangle = M|\text{Input}\rangle$. Since D-FF, then $D = |\mathbf{Q}^+\rangle$.

D. Bra–Ket Representation (Dirac Notation)

FSM Basis State: $|S_0\rangle = |000\rangle$, $|S_1\rangle = |001\rangle$, $|S_2\rangle = |010\rangle$, $|S_3\rangle = |011\rangle$, $|S_4\rangle = |100\rangle$. Output Operator $\hat{O}$:

$$\hat{O} = |000000\rangle\langle S_0| + |101111\rangle(\langle S_1| + \langle S_2| + \langle S_3| + \langle S_4|)$$

Transition Operator $\hat{T}$:

$$\hat{T} = |S_0\rangle\langle P_0| + |S_1\rangle\langle P_1| + |S_2\rangle\langle P_2| + |S_3\rangle\langle P_3| + |S_4\rangle\langle P_4|$$

E. Gates Used

NOT, AND, OR, BUFFER, and Ground/Constant Gate are used to form sensor categories (P0-P4), Next State (Q2, Q1, Q0), and Actuator Outputs.

VII. Code Implementation

Listing 1. HDL Implementation for FPGA

```
// ======================================================================
// Automatic Environmental Control System - Verilog HDL for FPGA
// Mochamad Rafly Firmansyah / 2042241130
// ======================================================================
module environmental_control_fsm (
input wire clk,           // Clock signal
input wire reset,         // Reset signal (active high)
input wire S1,            // Temperature sensor (1=normal, 0=abnormal)
input wire S2,            // Humidity sensor
input wire S3,            // VOC sensor
input wire S4,            // Dust sensor
input wire S5,            // Airflow sensor
input wire S6,            // Light sensor
output reg A1,            // Exhaust Fan
output reg A2,            // Inline Duct Fan (always 0)
output reg A3,            // Humidifier
output reg A4,            // Dehumidifier
output reg A5,            // Cooling System
output reg A6,            // LED Light System
output reg [2:0] current_state  // Current FSM state
);
// State encoding
localparam S0_IDLE = 3'b000;
localparam S1_SINGLE = 3'b001;
localparam S2_MULTIPLE = 3'b010;
localparam S3_CRITICAL = 3'b011;
localparam S4_EMERGENCY = 3'b100;

reg [2:0] next_state;
reg [2:0] abnormal_count;

// Count abnormal sensors
always @(*) begin
abnormal_count = 3'b000;
if (!S1) abnormal_count = abnormal_count + 1;
if (!S2) abnormal_count = abnormal_count + 1;
if (!S3) abnormal_count = abnormal_count + 1;
if (!S4) abnormal_count = abnormal_count + 1;
if (!S5) abnormal_count = abnormal_count + 1;
if (!S6) abnormal_count = abnormal_count + 1;
end

// Next State Logic
always @(*) begin
case (abnormal_count)
3'd0: next_state = S0_IDLE;
3'd1: next_state = S1_SINGLE;
3'd2, 3'd3: next_state = S2_MULTIPLE;
3'd4, 3'd5: next_state = S3_CRITICAL;
3'd6: next_state = S4_EMERGENCY;
default: next_state = S0_IDLE;
endcase
end

// State Register
always @(posedge clk or posedge reset) begin
if (reset) current_state <= S0_IDLE;
else current_state <= next_state;
end

// Output Logic
always @(*) begin
A1 = 0; A2 = 0; A3 = 0; A4 = 0; A5 = 0; A6 = 0;
case (current_state)
S0_IDLE: begin end
S1_SINGLE: begin
if (!S1) A5 = 1;
if (!S2) A3 = 1;
if (!S3) A1 = 1;
if (!S4) A1 = 1;
if (!S5) A4 = 1;
if (!S6) A6 = 1;
end
S2_MULTIPLE: begin
if (!S1) A5 = 1;
if (!S2) A3 = 1;
if (!S3) A1 = 1;
if (!S4) A1 = 1;
if (!S5) A4 = 1;
if (!S6) A6 = 1;
end
S3_CRITICAL: begin
if (!S1) A5 = 1;
if (!S2) A3 = 1;
if (!S3) A1 = 1;
```

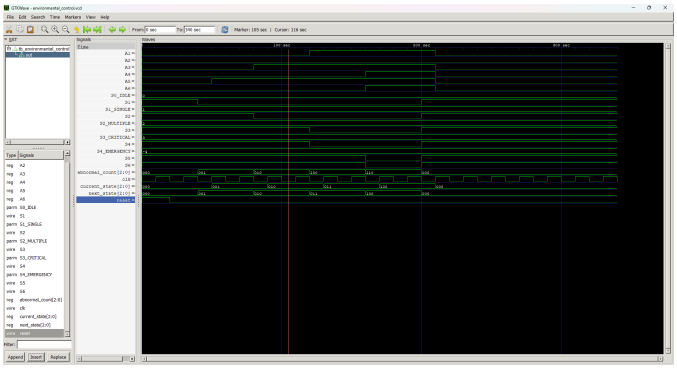
```
                if (!S4) A1 = 1;
                if (!S5) A4 = 1;
                if (!S6) A6 = 1;
                end
                S4_EMERGENCY: begin
                A1 = 1; A3 = 1; A4 = 1; A5 = 1; A6 = 1; A2 = 0;
                end
                default: begin end
                endcase
                end
                endmodule

                // TESTBENCH (Short)
                module tb_environmental_control;
                reg clk, reset;
                reg S1, S2, S3, S4, S5, S6;
                wire A1, A2, A3, A4, A5, A6;
                wire [2:0] current_state;
                environmental_control_fsm uut (
                .clk(clk), .reset(reset),
                .S1(S1), .S2(S2), .S3(S3), .S4(S4), .S5(S5), .S6(S6),
                .A1(A1), .A2(A2), .A3(A3), .A4(A4), .A5(A5), .A6(A6),
                .current_state(current_state)
                );
                initial begin
                clk = 0; forever #10 clk = ~clk;
                end
                initial begin
                $dumpfile("environmental_control.vcd");
                $dumpvars(0, tb_environmental_control);
                // Test Case 1: Reset
                reset = 1; S1=1; S2=1; S3=1; S4=1; S5=1; S6=1; #20 reset = 0;
                // Test Case 2: Single sensor abnormal
                S1=0; #40;
                // Test Case 3: Multiple
                S2=0; #40;
                // Test Case 4: Critical
                S3=0; S4=0; #40;
                // Test Case 5: Emergency
                S5=0; S6=0; #40;
                // Recovery
                S1=1; S2=1; S3=1; S4=1; S5=1; S6=1; #40;
                $finish;
                end
                endmodule
```

Listing 2. C# Microcontroller Simulation

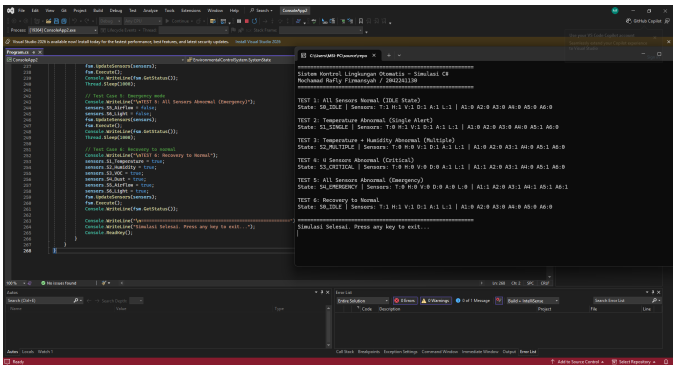
```
// ====================================================================
// Automatic Environmental Control System - C# for Microcontroller
// Mochamad Rafly Firmansyah / 2042241130
// ====================================================================
using System;
using System.Threading;

namespace EnvironmentalControlSystem {
        public enum SystemState {
                S0_IDLE = 0, S1_SINGLE = 1, S2_MULTIPLE = 2, S3_CRITICAL = 3, S4_EMERGENCY = 4
        }

        public struct SensorInputs {
                public bool S1_Temperature;
                public bool S2_Humidity;
                public bool S3_VOC;
                public bool S4_Dust;
                public bool S5_Airflow;
                public bool S6_Light;
                public int CountAbnormal() {
                        int count = 0;
                        if (!S1_Temperature) count++;
                        if (!S2_Humidity) count++;
                        if (!S3_VOC) count++;
                        if (!S4_Dust) count++;
                        if (!S5_Airflow) count++;
                        if (!S6_Light) count++;
                        return count;
                }
        }

        public struct ActuatorOutputs {
                public bool A1_ExhaustFan;
                public bool A2_InlineDuctFan;
                public bool A3_Dehumidifier;
                public bool A4_InlineDuctFan2;
```

```
                        public bool A5_CoolingSystem;
                        public bool A6_LEDSystem;
                        public void Reset() {
                                A1_ExhaustFan = false; A2_InlineDuctFan = false;
                                A3_Dehumidifier = false; A4_InlineDuctFan2 = false;
                                A5_CoolingSystem = false; A6_LEDSystem = false;
                        }
                }

                public class EnvironmentalControlFSM {
                        private SystemState currentState;
                        private SensorInputs sensors;
                        private ActuatorOutputs actuators;

                        public EnvironmentalControlFSM() { Reset(); }
                        public void Reset() {
                                currentState = SystemState.S0_IDLE; actuators.Reset();
                        }
                        public void UpdateSensors(SensorInputs newSensors) { sensors = newSensors; }

                        private SystemState DetermineNextState() {
                                int abnormalCount = sensors.CountAbnormal();
                                switch (abnormalCount) {
                                        case 0: return SystemState.S0_IDLE;
                                        case 1: return SystemState.S1_SINGLE;
                                        case 2: case 3: return SystemState.S2_MULTIPLE;
                                        case 4: case 5: return SystemState.S3_CRITICAL;
                                        case 6: return SystemState.S4_EMERGENCY;
                                        default: return SystemState.S0_IDLE;
                                }
                        }

                        private void UpdateActuators() {
                                actuators.Reset();
                                switch (currentState) {
                                        case SystemState.S0_IDLE: break;
                                        case SystemState.S1_SINGLE:
                                        if (!sensors.S1_Temperature) actuators.A5_CoolingSystem = true;
                                        if (!sensors.S2_Humidity) actuators.A3_Dehumidifier = true;
                                        if (!sensors.S3_VOC) actuators.A1_ExhaustFan = true;
                                        if (!sensors.S4_Dust) actuators.A1_ExhaustFan = true;
                                        if (!sensors.S5_Airflow) actuators.A4_InlineDuctFan2 = true;
                                        if (!sensors.S6_Light) actuators.A6_LEDSystem = true;
                                        break;
                                        case SystemState.S2_MULTIPLE:
                                        case SystemState.S3_CRITICAL:
                                        if (!sensors.S1_Temperature) actuators.A5_CoolingSystem = true;
                                        if (!sensors.S2_Humidity) actuators.A3_Dehumidifier = true;
                                        if (!sensors.S3_VOC) actuators.A1_ExhaustFan = true;
                                        if (!sensors.S4_Dust) actuators.A1_ExhaustFan = true;
                                        if (!sensors.S5_Airflow) actuators.A4_InlineDuctFan2 = true;
                                        if (!sensors.S6_Light) actuators.A6_LEDSystem = true;
                                        break;
                                        case SystemState.S4_EMERGENCY:
                                        actuators.A1_ExhaustFan = true; actuators.A3_Dehumidifier = true;
                                        actuators.A4_InlineDuctFan2 = true; actuators.A5_CoolingSystem = true;
                                        actuators.A6_LEDSystem = true;
                                        break;
                                }
                        }
                        public void Execute() {
                                currentState = DetermineNextState();
                                UpdateActuators();
                        }
                }
        }

```

VIII. Visual Output



Gambar 1. Icarus Verilog HDL Output



Gambar 2. Visual Studio C# Output

References

[1] D. Bora, D. Singh, and B. Negi, "Utilization of DS18B20 Temperature Sensor for Predictive Maintenance of Reciprocating Compressor," in *2023 International Conference on Power Energy, Environment and Intelligent Control, PEEIC 2023*, IEEE Inc., 2023, pp. 147–150. doi: 10.1109/PEEIC59336.2023.10450639.

[2] C. S. Priya and F. S. Francis, "A Novel LoRaWAN-based Real-time Traffic Analysis Approach for Vehicle Congestion Estimation," in *Winter Summit on Smart Computing and Networks, WiSSCoN 2023*, IEEE Inc., 2023. doi: 10.1109/WiSSCoN56857.2023.10133856.

[3] Jithina Jose and T.Sasipraba, "Indoor air quality monitors using IOT sensors and LPWAN," [IEEE], 2019.

[4] J. J. R. Balbin, A. J. G. De Guzman, and C. J. C. Rambuyon, "Air Pollutant Detection System Utilizing an IoT-based Electronic Nose for Air Purifier," in *Proceeding - ELTICOM 2022*, IEEE Inc., 2022, pp. 136–140. doi: 10.1109/ELTICOM57747.2022.10037913.

[5] B. Junaidin et al., "Advancements in Microcontroller Technology for Wind Speed Measurement in Wind Tunnels," in *Proceedings - IEIT 2023*, IEEE Inc., 2023, pp. 71–75. doi: 10.1109/IEIT59852.2023.10335512.

[6] S. N. Patrialova, T. Agasta, and I. N. Sari, "Prototype Design of Automatic Light Intensity Control in Smart Green House," in *ICAMIMIA 2021 - Proceeding*, IEEE Inc., 2021, pp. 41–46. doi: 10.1109/ICAMIMIA54022.2021.9807698.

[7] A. F. Adziima et al., "Prototype Design of Automatic Switching Speed of Exhaust Fan For Air Quality Control Based On IoT," in *ICAMIMIA 2021 - Proceeding*, IEEE Inc., 2021, pp. 114–119. doi: 10.1109/ICAMIMIA54022.2021.9809416.

[8] Paulo Abreu et al., "VENTI: experimental controller for inline duct fan," IEEE, 2017.

[9] S. Qiaoyun et al., "Design and Implementation of a Wireless Control Intelligent Humidifier," IEEE, Sep. 2025, pp. 1396–1400. doi: 10.1109/itaic64559.2025.11163186.

[10] A. K. Putri et al., "Automatic Usage of IoT-Based Dehumidifiers in High-Humidity Spaces," in *ICISS 2024 - Proceeding*, IEEE Inc., 2024. doi: 10.1109/ICISS62896.2024.10751296.

[11] K. Kim et al., "Design and Performance Evaluation of LH2 Cooling System for HTS Motor Electric Propulsion Platform," *IEEE Transactions on Applied Superconductivity*, vol. 35, no. 5, 2025, doi: 10.1109/TASC.2025.3529421.

[12] I. Chew et al., "A Spectrally Tunable Smart LED Lighting System With Closed-Loop Control," *IEEE Sens J*, vol. 16, no. 11, pp. 4452–4459, Jun. 2016, doi: 10.1109/JSEN.2016.2542265.