Sistema de monitoreo de temperatura

Especificaciones

V1.0 — Enero 2026

Erika Garcia

Diseño Digital II

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Este documento presenta las especificaciones de un sistema digital de monitoreo de temperatura con arquitectura jerárquica de FSM, implementado en SystemVerilog para aplicaciones de control térmico en invernaderos y sistemas embebidos. EL propósito es:

- Proporcionar los requisitos del sistema.
- Describir la arquitectura implementada (procesamiento, temporización, control).
- Especificar los estados de la máquina de estados finitos (FSM) y condiciones de transición.
- Documentar el mecanismo de contador de persistencia para filtrado de transitorios.
- Proporcionar la estructura de RTL y metodología de verificación.

1.1. Alcance del diseño

El diseño abarca desde la recepción de datos de temperatura escalados de 11 bits signado hasta el control de actuadores (calefactor y ventilador) mediante una lógica de persistencia configurable y una máquina de estados finitos (FSM).

1.2. Definiciones y acrónimos

- **FSM**: Máquina de Estados Finitos (Finite State Machine).
- **RTL**: Nivel de Transferencia de Registros (Register Transfer Level).
- **SVA**: Aserciones de SystemVerilog (SystemVerilog Assertions).
- **Persistencia (N)**: Número de ciclos consecutivos requeridos para validar un cambio de estado crítico.
- **T_escalada**: Valor de temperatura multiplicado por 10 ($T \times 10$) para procesamiento de enteros.
- **DUT**: Device Under Test - Dispositivo bajo prueba.

2. Especificaciones del Sistema

2.1. Parámetros de Operación

2.1.1. Rango de Temperatura y Escalado

El sistema opera en el rango típico de sensores comerciales de temperatura de propósito general (-40.0°C a 85.0°C), con una resolución de 0.1°C, utilizando escalado $\times 10$ para convertir a valores enteros (-400 a 850).

Los umbrales implementados en el diseño están específicamente enfocados para monitoreo ambiental en invernaderos, donde temperaturas inferiores a 18.0°C o superiores a 25.9°C pueden afectar el crecimiento vegetal.

$$\text{Valor procesado} = \text{Temperatura (°C)} \times 10$$

Condición	Rango Real (°C)	Rango Escalado
BAJO	$T < 18{,}0$	$T < 180$
NORMAL	$18{,}0 \leq T \leq 25{,}9$	$180 \leq T \leq 259$
ALTO	$T > 25{,}9$	$T > 259$

Cuadro 1: Rangos operativos del sistema

2.1.2. Resolución y Precisión

- **Resolución**: 0.1°C
- **Representación**: 11 bits con signo en complemento a 2
- **Ancho de entrada**: 11 bits (rango -1024 a 1023)
- **Rango de interés**: -400 a 850

2.2. Lógica de Control

2.2.1. Estados del Sistema

Estado	Código	Calefactor	Ventilador	Alerta
NORMAL	00	0	0	0
BAJA	01	0	0	0
ALTO	10	0	0	0
ALERTA	11	1 si T< 180	1 si T> 259	1

Cuadro 2: Estados de operación del sistema

2.2.2. Condiciones de Transición

- **NORMAL a BAJA**: Cuando temperatura < 180 (18.0°C)
- **NORMAL a ALTO**: Cuando temperatura > 259 (25.9°C)
- **BAJA a NORMAL**: Cuando temperatura retorna a $[180, 259]$
- **ALTO a NORMAL**: Cuando temperatura retorna a $[180, 259]$
- **BAJO a ALERTA**: per_bajo activo (5 ciclos de frío)
- **ALTO a ALERTA**: per_alto activo (5 ciclos de calor)
- **BAJO a ALTO**: Cambio directo $T > 259$
- **ALTO a BAJO** : Cambio directo $T < 180$
- **ALERTA a NORMAL**: Cuando temperatura retorna a rango normal
- **ALERTA a BAJO**: $T < 180$ sin persistencia
- **ALERTA a ALTO**: $T > 259$ sin persistencia

2.2.3. Lógica de persistencia

- **Propósito**: Filtrar transitorios breves del sensor
- **Valor por defecto**: N = 5 ciclos de reloj
- **Contador**: Se implementa dos contadores independientes, con el objetivo de separar la persistencia en temperatura baja y alta. Incrementa mientras la temperatura está fuera de rango normal
- **Reset** : Cuando temperatura vuelve a rango normal

2.2.4. Lógica de alerta automática

Características:

- la alerta se activa cuando la condición de temperatura baja o alto mantiene persistencia durante N ciclos de reloj consecutivos, descartando eventos transitorios aislados.
- la alerta se desactiva automáticamente al volver a la temperatura normal o si la temperatura sale del rango crítico o pierde persistencia, la FSM transiciona de **ALERTA** a los estados **BAJO** o **ALTO** según corresponda, apagando los actuadores pero manteniendo el monitoreo preventivo.
- La activación selectiva de actuadores: El **calefactor** se habilita exclusivamente tras la validación de persistencia de frío ($T < 180$), mientras que el **ventilador** opera únicamente tras la validación de persistencia de calor ($T > 259$).

3. Arquitectura general

3.1. Microarquitectura

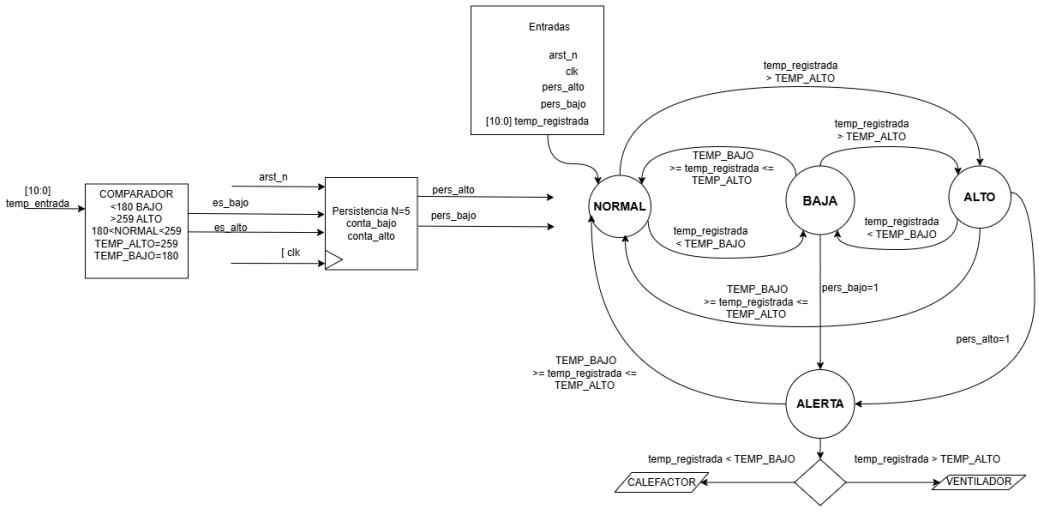

Figura 1: Microarquitectura del sistema de monitoreo de temperatura

3.2. Flujo de Datos y Control

La temperatura ingresa al **comparador_temp**, el cual genera los flags inmediatos **es_bajo** y **es_alto**. Estos alimentan al bloque **persistencia_ctr**, que genera señales de validación (**per_bajo, per_alto**) solo si los flags se mantienen estables por N ciclos. Finalmente, la FSM **estado_temp** procesa la temperatura registrada y las señales de persistencia para determinar el estado actual y activar las señales para los actuadores.

3.3. Jerarquía de diseño

El proyecto se divide en arquitectura RTL y el entorno de verificación. El sistema se organiza:

Nivel	Módulo	Función
Raíz	`monitoreo_top`	Integración
Nivel 1	`estado_temp`	Control (FSM).
Nivel 2	`persistencia_ctr`	Temporización y persistencia
Nivel 3	`comparador_temp`	Procesamiento de datos

Cuadro 3: Jerarquía y función de los módulos del sistema

3.4. Puertos del sistema

Puerto	Dirección	Ancho	Descripción
`clk`	Entrada	1 bit	Reloj del sistema (Único, síncrono al flanco de subida)
`arst_n`	Entrada	1 bit	Reset asíncrono activo en bajo
`temp_entrada`	Entrada	11 bits	Temperatura escalada del sensor (T $\times$ 10)
`alerta`	Salida	1 bit	Indicador de condición de alerta
`calefactor`	Salida	1 bit	Activación del sistema de calefacción
`ventilador`	Salida	1 bit	Activación del sistema de ventilación
`estado_actual`	Salida	2 bits	Estado actual de la FSM
`cont_bajo`	Salida	3 bits	Contador de persistencia de frío (0-5)
`cont_alto`	Salida	3 bits	Contador de persistencia de calor (0-5)
`per_bajo`	Salida	1 bit	Señal de persistencia de frío alcanzada
`per_alto`	Salida	1 bit	Señal de persistencia de calor alcanzada

Cuadro 4: Puertos del sistema

3.5. Entorno de Verificación

Compuesto por módulos dedicados a la validación y cobertura del diseño:

- **monitoreo_pkg.sv**: Package que contiene las definiciones de clases para generación de estímulos y parámetros de configuración, asegurando la consistencia de datos.
- **interface_monitoreo.sv**: Encapsula las señales del DUT, facilitando la conexión con el testbench y permitiendo el monitoreo de señales internas.
- **monitoreo_tb.sv**: Testbench principal que implementa múltiples casos de prueba, incluyendo pruebas de persistencia, transiciones, límites y valores extremos.
- **fv_monitoreo.sv**: Módulo de verificación formal que contiene SVA para validar propiedades críticas del sistema como exclusión mutua, persistencia y seguridad de actuadores.

- **cov_monitoreo.sv**: Implementa covergroups para medir la cobertura funcional, incluyendo estados de la FSM, transiciones, rangos de temperatura y valores de contadores.

3.6. Descripción de Módulos

3.6.1. Módulo: comparador_temp

Se encarga de determinar si la temperatura de entrada se encuentra por debajo del umbral de frio o por encima del umbral de calor. Implementa lógica combinacional.

Parámetros

Parámetro	Valor	Descripción
TEMP_BAJO	180	Límite superior para condición de frío (18.0°C).
TEMP_ALTO	259	Límite inferior para condición de calor (25.9°C).

Cuadro 5: Parámetros del módulo comparador_temp

Puertos

Señal	Tipo	Dirección	Ancho	Descripción
temp_entrada	logic signed	Entrada	11	Temperatura escalada ($T \times 10$).
es_bajo	logic	Salida	1	Activa cuando temp_entrada $<$ TEMP_BAJO.
es_alto	logic	Salida	1	Activa cuando temp_entrada $>$ TEMP_ALTO.

Cuadro 6: Puertos del módulo

Implementación en SystemVerilog:

```
module comparador_temp(
    input logic signed [10:0] temp_entrada,     // temperatura de entrada del sensor
    output logic es_bajo,                       // temp < 180 (para contador fr o)
    output logic es_alto                        // temp > 259 (para contador calor)
);

    // Par metros de rango ya escalado
    parameter TEMP_BAJO = 180;   // 18 C  se fija la temperatura como limite maximo
        para frio
    parameter TEMP_ALTO = 259;   // 25.9 C  se fija la temperatura como limite maximo
         para alto

    // Comparaci n de la temperatura
    assign es_bajo = (temp_entrada < TEMP_BAJO); // devuelve si la temperatura
        registrada esta en el rango bajo
    assign es_alto = (temp_entrada > TEMP_ALTO); // devuelve si la temperatura
        registrada esta en el rango alto

endmodule
```

Listing 1: Modulo comparador

3.6.2. Módulo: persistencia_ctr

El módulo `persistencia_ctr` actúa como un filtro para eliminar la sensibilidad del sistema ante ruidos eléctricos o fluctuaciones térmicas transitorias del sensor. Su arquitectura se fundamenta en el uso de dos contadores síncronos independientes, `cont_bajo` y `cont_alto`, ambos de 3 bits, controlados por un parámetro configurable $N = 5$. El funcionamiento del módulo se rige por el flanco de subida del reloj (`clk`) y un reset asíncrono (`arst_n`) que garantiza un estado inicial nulo. Mientras la señal `es_bajo` se mantenga activa, el contador respectivo incrementa su valor secuencialmente hasta alcanzar el límite N.Si `es_bajo` se desactiva (la temperatura sale de la zona de frío), el contador se resetea inmediatamente a cero. El comportamiento del contador de calor es similar al anterior.

Cuando cualquiera de ellos alcanza este valor N, se activa la señal de persistencia correspondiente (`per_bajo` o `per_alto`). Estas señales se generan de forma combinacional, lo que permite que la FSM reaccione en el mismo ciclo en que se cumple la condición de persistencia.

Puertos

Señal	Tipo	Dirección	Ancho	Descripción
clk	logic	Entrada	1	Reloj del sistema.
arst_n	logic	Entrada	1	Reset asíncrono activo en bajo.
es_bajo	logic	Entrada	1	Indicador de condición de frío.
es_alto	logic	Entrada	1	Indicador de condición de calor.
cont_bajo	logic	salida	3	Contador de ciclos en condición de frío (0–5).
cont_alto	logic	salida	3	Contador de ciclos en condición de calor (0–5).
per_bajo	logic	salida	1	Señal de persistencia de frío.
per_alto	logic	salida	1	Señal de persistencia de calor.

Cuadro 7: Puertos del módulo

Implementación en SystemVerilog:

```
module persistencia_ctr #(parameter N = 5)(
    input  logic clk,               // reloj del sistema
    input  logic arst_n,            // reset asincrono
    input  logic es_bajo,           // temp < 180
    input  logic es_alto,           // temp > 259
    output logic [2:0] cont_bajo,   // contador para la temperatura registrada en
        bajo
    output logic [2:0] cont_alto,   // contador para la temperatura registrada en
        alto
    output logic per_bajo,          // Persistencia de frio
    output logic per_alto           // Persistencia de calor
);

    always_ff (posedge clk, negedge arst_n) begin
        if (!arst_n) begin                      // reset del sistema
            cont_bajo <= 3'd0;
            cont_alto <= 3'd0;
        end else begin
            if (es_bajo) begin                 // si la persistencia esta en alto
                if (cont_bajo < N)
                    cont_bajo <= cont_bajo + 1;// se incrementa el el contador para
                        temperatura baja
            end else begin
                cont_bajo <= 3'd0;
            end

            if (es_alto) begin
```

```
25                 if (cont_alto< N)
26                     cont_alto <= cont_alto + 1;// se incrementa el el contador para
                           temperatura alta
27             end else begin
28                 cont_alto <= 3'd0;
29             end
30         end
31     end
32
33     assign per_bajo= (cont_bajo >= N-1) && es_bajo; // logica combinacional para
           asignar la condicion de persistencia en bajo
34     assign per_alto= (cont_alto >= N-1) && es_alto;// logica combinacional para
           asignar la condicion de persistencia alto
35
36 endmodule
```

Listing 2: Modulo de persistencia

3.6.3. Módulo: estado_temp (FSM)

El módulo `estado_temp` constituye la unidad de control central del sistema, implementando una Máquina de Estados Finitos (FSM) de tipo Moore-Mealy que gestiona la lógica de alerta y la activación de actuadores. La FSM está definida con cuatro estados operativos: `NORMAL` (00), `BAJO` (01), `ALTO` (10) y `ALERTA` (11). El diseño utiliza un enfoque síncrono para las transiciones de estado, regido por el flanco de subida de `clk`, y cuenta con un reset asíncrono (`arst_n`) que fuerza el retorno inmediato al estado `NORMAL`, garantizando un punto de partida seguro. La FSM permite transiciones directas entre `BAJO` y `ALTO`, si la temperatura cruza los umbrales antes de completar el ciclo de persistencia, de modo que, si en estado `ALERTA` se pierde la persistencia pero la temperatura sigue fuera de rango, el sistema retroceda a los estados intermedios (`BAJO` o `ALTO`) para re-evaluar la condición.

La lógica de salida de los actuadores se implementa de forma combinacional para asegurar una respuesta inmediata.

Puertos

Señal	Tipo	Dirección	Ancho	Descripción
clk	logic	Entrada	1	Reloj del sistema.
arst_n	logic	Entrada	1	Reset asíncrono activo en bajo.
temp_registrado	logic signed	Entrada	11	Temperatura de entrada.
per_bajo	logic	Entrada	1	Señal de persistencia de frío.
per_alto	logic	Entrada	1	Señal de persistencia de calor.
alerta	logic	Salida	1	Indicador de condición de alerta.
calefactor	logic	Salida	1	Activación del calefactor.
ventilador	logic	Salida	1	Activación del ventilador.
estado_actual	logic	Salida	2	Codificación del estado actual de la FSM.

Cuadro 8: Puertos del módulo

Implementación en SystemVerilog:

```
1     always_ff (posedge clk, negedge arst_n) begin
2         if (!arst_n) begin
3             estado      <= NORMAL;    // Para iniciar en la maquina de estado
4         end else begin
5             case (estado)
```

```
6                   NORMAL: begin
7                       if (temp_registrado < TEMP_BAJO) estado <= BAJO;
8                       else if (temp_registrado > TEMP_ALTO) estado <= ALTO;
9                   end
10                  BAJO: begin
11                      if (per_bajo) begin
12                          estado <= ALERTA;
13                      end else if (temp_registrado >= TEMP_BAJO && temp_registrado <=
                            TEMP_ALTO) begin
14                          estado <= NORMAL;
15                      end else if (temp_registrado > TEMP_ALTO) begin
16                          estado <= ALTO;
17                      end
18                  end
19                  ALTO: begin
20                      if (per_alto)begin
21                          estado <= ALERTA;
22
23                      end else if (temp_registrado >= TEMP_BAJO && temp_registrado <=
                            TEMP_ALTO) begin
24                          estado <= NORMAL;
25                      end else if (temp_registrado < TEMP_BAJO) begin
26                          estado <= BAJO;
27                      end
28                  end
29                  ALERTA: begin
30                      if (temp_registrado >= TEMP_BAJO && temp_registrado <= TEMP_ALTO)
                             begin // retorno a estado normal
31                          estado <= NORMAL;
32                      end else if (temp_registrado < TEMP_BAJO && !per_bajo) begin
33                          estado <= BAJO;
34                      end else if (temp_registrado > TEMP_ALTO && !per_alto) begin
35                          estado <= ALTO;
36                      end
37                  end
38                  default: begin      // estado por defecto
39                      estado <= NORMAL;
40                  end
41              endcase
42
43          end
44      end
```

Listing 3: Modulo de FSM

4. Entorno de verificación

El entorno de verificación está diseñado como una arquitectura modular que separa la generación de estímulos, la observación de señales y la validación del comportamiento del DUT.

1. `Package OOP (monitoreo_pkg.sv)`: Generación de estímulos mediante clases.
2. `interface (interface_monitoreo.sv)`:Encapsulación de señales.
3. `Testbench (monitoreo_tb.sv)`: Coordina la ejecución de escenarios.
4. El módulo de `aserciones (fv_monitoreo.sv)` Verificación formal SVA.
5. El módulo de `cobertura (cov_monitoreo.sv)` Medición de cobertura funcional.

4.1. Requisitos funcionales

- **Detección de umbrales**
  El sistema debe identificar condiciones de temperatura baja cuando $T < 18{,}0°$C y alta cuando $T > 25{,}9°$C, con resolución de 0.1°C (representación escalada $\times 10$).
- **Validación por persistencia**
  El sistema debe ignorar fluctuaciones térmicas cuya duración sea menor a $N = 5$ ciclos de reloj consecutivos.
- **Exclusión mutua de actuadores**
  El sistema no debe activar simultáneamente las salidas `calefactor` y `ventilador` bajo ninguna condición de operación.
- **Recuperación automática**
  El sistema debe desactivar inmediatamente los actuadores cuando la temperatura regrese al rango normal $[180, 259]$ (18.0°C a 25.9°C).
- **Reset prioritario**
  La activación de `arst_n` debe forzar el estado `NORMAL` y desactivar todos los actuadores en menos de un ciclo de reloj.
- **Filtrado de ruido**
  Eventos de temperatura fuera de rango cuya duración sea menor a N ciclos no debe provocar la transición al estado `ALERTA`.

4.2. Estrategia de verificación

4.2.1. Estados verificable

1. NORMAL $\rightarrow$ BAJO (T < 180)
2. NORMAL $\rightarrow$ ALTO (T > 259)
3. BAJO $\rightarrow$ NORMAL (T vuelve a [180,259])
4. ALTO $\rightarrow$ NORMAL (T vuelve a [180,259])
5. BAJO $\rightarrow$ ALERTA (contador $\geq$ N)

6. ALTO $\rightarrow$ ALERTA (contador $\geq$ N)
7. ALERTA (por Frío) $\rightarrow$ ALTO (Ocurre cuando $T > 259$. Se pierde la persistencia de frío y el sistema transiciona para evaluar la nueva condición de calor.)
8. ALERTA (por calor) $\rightarrow$ BAJO (Ocurre cuando $T < 180$. Se pierde la persistencia de calor y el sistema transiciona para evaluar la nueva condición de frio.)
9. ALERTA $\rightarrow$ NORMAL (T vuelve a rango normal)
10. BAJO $\rightarrow$ ALTO (cambio directo T > 259)
11. ALTO $\rightarrow$ BAJO (cambio directo T < 180)

4.3. Testbench

Basado en clases con randomización restringida (temp_bajo, temp_normal, temp_alto, temp_fuera_rango).

Parámetros globales

- `TEMP_BAJO = 180`: Límite para condición de frío (18.0°C).
- `TEMP_ALTO = 259`: Límite para condición de calor (25.9°C).
- `MIN_SENSOR = -400`: Valor mínimo del sensor (-40.0°C).
- `MAX_SENSOR = 850`: Valor máximo del sensor (85.0°C).

4.3.1. Generación de Estímulos

Se implementa una jerarquía de clases orientadas a objetos para modelar el comportamiento del sensor de temperatura y generar escenarios representativos de operación.

Clase	Restricción (Constraint)	Propósito
`temp_base`	`valor inside [MIN_SENSOR:MAX_SENSOR]`	Clase base que garantiza que los valores generados se encuentren dentro del rango físico válido del sensor.
`temp_bajo`	`valor <180`	Generación de escenarios de temperatura baja para validar detección de frío.
`temp_normal`	`valor inside [180:259]`	Generación de condiciones de operación estable dentro del rango nominal.
`temp_alto`	`valor >259`	Generación de escenarios de temperatura alta para validar detección de calor.
`temp_persistente`	`valor inside [prev-2 : prev+2]`	Simulación de estabilidad térmica (baja variación) para verificar la lógica de persistencia.
`temp_fuera_rango`	`valor inside [-1024:-401], [851:1023]`	Generación de condiciones fuera de rango para pruebas de robustez del bus de 11 bits.

Cuadro 9: Jerarquía de clases para generación de estímulos

4.4. Plan de Pruebas

Caso	Nombre	Descripción	Estímulos	Resultado
1	T_NORMAL_B	Normal básico	4 valores normales	Verificado
2	T_BAJO_INM	Frío inmediato	4 valores bajos	Verificado
3	T_ALTO_INM	Calor inmediato	4 valores altos	Verificado
4	T_BAJO_PERS	Frío persistente	5 ciclos < 180	Verificado
5	T_ALTO_PERS	Calor persistente	5 ciclos > 259	Verificado
6	T_RECU_AUTO	Recuperación BAJO→NORMAL	5 frío + 5 normal	Verificado
7	T_BAJO_ALTO	Transición BAJO→ALTO	5 frío + calor	Verificado
8	T_ALTO_BAJO	Transición ALTO→BAJO	5 calor + frío	Verificado
9	T_TRANS_ALTO	Transitorio calor	3 calor + normal	Verificado
10	T_TRANS_BAJO	Transitorio frío	3 frío + normal	Verificado
11	T_LIM_BAJO	Límite 179→180	179, 180	Verificado
12	T_LIM_ALTO	Límite 259→260	259, 260	Verificado
13	LIMITE_INFERIOR	Valores extremos negativos	-1024 a -401	Verificado
14	LIMITE_SUPERIOR	Valores extremos positivos	851 a 1023	Verificado
15	RESET_SISTEMA	Reset en diferentes estados	Reset en NORMAL y ALERTA	Verificado

Cuadro 10: Plan de pruebas implementado

4.5. SystemVerilog Assertions (SVA)

Implementadas en `fv_monitoreo.sv`

Tabla de Aserciones SVA

#	Nombre	Descripción
1	`alerta_en_reset`	Durante reset, la alerta debe estar desactivada
2	`ventilador_en_reset`	Durante reset, el ventilador debe estar apagado
3	`calefactor_en_reset`	Durante reset, el calefactor debe estar apagado
4	`persistencia_frio`	5 ciclos de frío, al siguiente ciclo se activan alerta y calefactor
5	`persistencia_calor`	5 ciclos de calor, al siguiente ciclo se activan alerta y ventilador
6	`estabilidad_normal`	En rango normal, la alerta debe ser 0 en el próximo ciclo
7	`recuperacion_normal`	En estado NORMAL, la alerta debe ser 0
8	`exclusion`	Calefactor y ventilador nunca se activan simultáneamente
9	`salida_alerta`	Si la alerta está activa mientras la temperatura del ciclo anterior estaba dentro del rango normal, entonces la alerta debe desactivarse en el siguiente ciclo
10	`seguridad_calefactor`	Calefactor solo con alerta y temperatura baja
11	`seguridad_ventilador`	Ventilador solo con alerta y temperatura alta
12	`transicion_frio_normal`	Transición de frío a normal debe llevar a estado NORMAL
13	`transicion_frio_calor`	Transición directa de frío a calor puede llevar a ALTO o ALERTA

Cuadro 11: Aserciones SystemVerilog implementadas en el módulo fv_monitoreo

4.6. Cobertura Funcional

La cobertura funcional se implementa mediante el módulo cov_monitoreo.sv, que utiliza covergroups de SystemVerilog para medir qué porcentaje de los escenarios de interés han sido ejercitados durante la simulación.

4.6.1. Cover Properties

#	Nombre	Descripción
1	exclusion_mutua_visto	Verifica que la condición prohibida (activación simultánea de ambos actuadores) sea observable para validar el chequeo de seguridad.
2	alerta_activada	Mide el número de veces que la señal de alerta fue activada durante la simulación.
3	cont_bajo_cinco_alcanzado	Registra las ocasiones en las que el contador de frío alcanzó el valor máximo de persistencia (5 ciclos).
4	cont_alto_cinco_alcanzado	Registra las ocasiones en las que el contador de calor alcanzó el valor máximo de persistencia (5 ciclos).
5	estado_alerta_alcanzado	Mide cuántas veces la FSM entró en el estado ALERTA.

Cuadro 12: Propiedades de cobertura funcional

```
‘COV(mon, exclusion_mutua_visto, 1’b1 |->, (!(ventilador && calefactor)))
‘COV(mon, alerta_activada, 1’b1 |->, (alerta == 1))
‘COV(mon, cont_bajo_cinco_alcanzado, 1’b1 |->, (cont_bajo == 5))
‘COV(mon, cont_alto_cinco_alcanzado, 1’b1 |->, (cont_alto == 5))
‘COV(mon, estado_alerta_alcanzado, 1’b1 |->, (estado_actual == 2’b11))
```

Listing 4: cover properties

4.6.2. Coverpoints

#	Coverpoint	Señal	Descripción
1	cp_temp	temp_entrada	Cobertura de los rangos de temperatura para verificar estimulación de todo el dominio de operación.
2	cp_estado	estado_actual	Cobertura de los estados de la FSM para asegurar que todos los estados sean alcanzados.
3	cp_contador_bajo	cont_bajo	Cobertura del valor del contador de persistencia de frío.
4	cp_contador_alto	cont_alto	Cobertura del valor del contador de persistencia de calor.
5	cp_actuadores	{calefactor, ventilador}	Cobertura de las combinaciones válidas de activación de actuadores.
6	cp_fsm_trans	estado_actual	Cobertura de transiciones entre estados de la FSM.

Cuadro 13: Coverpoints definidos para cobertura funcional

4.6.3. Coberturas

Se han implementado las siguientes coberturas:

```
cp_temp: coverpoint temp_entrada {
    bins min_sensor      = { MIN_SENSOR };
    bins rango_bajo      = { [MIN_SENSOR+1 : TEMP_BAJO-2] };
    bins limite_bajo     = { TEMP_BAJO - 1 };
    bins limite_normal_inf = { TEMP_BAJO };
```

```
    bins rango_normal    = { [TEMP_BAJO+1 : TEMP_ALTO-1] };
    bins limite_normal_sup = { TEMP_ALTO };
    bins limite_alto     = { TEMP_ALTO + 1 };
    bins rango_alto      = { [TEMP_ALTO+2 : MAX_SENSOR-1] };
    bins max_sensor      = { MAX_SENSOR };
}
```

Listing 5: Cobertura de rangos de temperatura

```
cp_estado: coverpoint estado_actual {
    bins NORMAL = {2'b00};
    bins BAJO   = {2'b01};
    bins ALTO   = {2'b10};
    bins ALERTA = {2'b11};
    illegal_bins otros = default;
}
```

Listing 6: Cobertura de Estados de la FSM

```
cp_actuadores: coverpoint {calefactor, ventilador} {
    bins ambos_off = {2'b00};
    bins solo_calefactor = {2'b10};
    bins solo_ventilador = {2'b01};
    illegal_bins ambos_on = {2'b11};
}
```

Listing 7: Cobertura de Actuadores

```
cp_fsm_trans: coverpoint estado_actual {
    bins normal_a_bajo   = (2'b00 => 2'b01);
    bins normal_a_alto   = (2'b00 => 2'b10);
    bins bajo_a_normal   = (2'b01 => 2'b00);
    bins alto_a_normal   = (2'b10 => 2'b00);
    bins bajo_a_alerta   = (2'b01 => 2'b11);
    bins alto_a_alerta   = (2'b10 => 2'b11);
    bins recu_auto       = (2'b11 => 2'b00);
    bins transitorio_bajo = (2'b00 => 2'b01 => 2'b00);
    bins transitorio_alto = (2'b00 => 2'b10 => 2'b00);
}
```

Listing 8: Cobertura de Estados intermedios de la FSM

5. Análisis y resultados

5.1. Métricas obtenidas

- **Overall Average Grade:** 88.92 %.
- **Cobertura de Código:** 89.91 %.
- **Cobertura Funcional (Covergroups):** 89.36 %.
- **Assertion Status Grade:** 84.62 %.

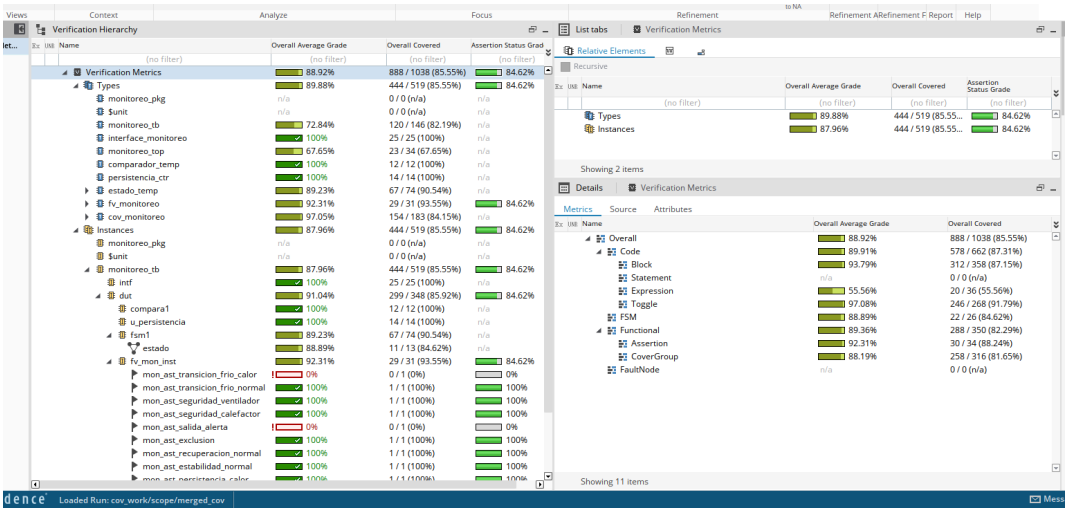

Figura 2: Cobertura del sistema

Tras la ejecución del plan de verificación y el análisis de las métricas de cobertura, se extraen las siguientes observaciones:

Tras completar el ciclo de verificación, se determinó que el 100 % de las aserciones de seguridad operativa fueron validadas sin presentar violaciones lógicas. No obstante, las propiedades `transicion_frio_a_calor` y `salida_alerta` se reportaron como 'no cubiertas'. Este escenario no indica un fallo en el diseño RTL, sino una limitación en la generación de estímulos aleatorios, los cuales no produjeron los gradientes de temperatura extremos ni los tiempos de estabilización necesarios para activar dichas métricas. La falta de cobertura en estas aserciones específicas se identifica como una oportunidad de mejora para el testbench.

6. Conclusiones

Se implementó exitosamente un diseño modular con separación clara de responsabilidades (procesamiento, temporización y control), facilitando la verificación.

Se implementó un conjunto de 13 aserciones que aseguran la integridad del sistema. El cumplimiento de aserciones críticas como la exclusión mutua garantiza que el hardware físico nunca se vea sometido a operaciones de activación simultanea de los actuadores, eliminando riesgos de seguridad. Aunque se identificaron brechas de cobertura en escenarios de transiciones específicas y en rangos de temperatura particulares, esto permitió diagnosticar la necesidad de incluir pruebas dirigidas en el plan de verificación.

Para futuras iteraciones o versiones avanzadas del sistema de monitoreo, se proponen sustituir las señales de encendido/apagado (on/off) por modulación por ancho de pulso (PWM). Esto permitiría, por ejemplo, que el ventilador gire más rápido a medida que la temperatura se aleja del rango normal, optimizando el control térmico y reduciendo el consumo energético.

7. Apéndices

Apéndice A: RTL

```
`timescale 1ns / 1ps

module comparador_temp(
    input logic signed [10:0] temp_entrada,      // temperatura de entrada del sensor
    output logic es_bajo,                        // temp < 180 (para contador fr o)
    output logic es_alto                         // temp > 259 (para contador calor)
);

    // Par metros de rango ya escalado
    parameter TEMP_BAJO = 180;   // 18 C  se fija la temperatura como limite maximo
        para frio
    parameter TEMP_ALTO = 259;   // 25.9 C  se fija la temperatura como limite maximo
         para alto

    // Comparaci n de la temperatura
    assign es_bajo = (temp_entrada < TEMP_BAJO); // devuelve si la temperatura
        registrada esta en el rango bajo
    assign es_alto = (temp_entrada > TEMP_ALTO); // devuelve si la temperatura
        registrada esta en el rango alto

endmodule
```

Listing 9: comparador_temp

```
`timescale 1ns / 1ps

module persistencia_ctr #(parameter N = 5)(
    input  logic clk,               // reloj del sistema
    input  logic arst_n,            // reset asincrono
    input  logic es_bajo,           // temp < 180
    input  logic es_alto,           // temp > 259
    output logic [2:0] cont_bajo,   // Contador para temperatura baja
    output logic [2:0] cont_alto,   // Contador para temperatura alta
    output logic per_bajo,          // Persistencia de fr o
    output logic per_alto           // Persistencia de calor
);

    always_ff (posedge clk, negedge arst_n) begin
        if (!arst_n) begin
            // Reset as ncrono: ambos contadores se inicializan a cero
            cont_bajo <= 3'd0;
            cont_alto <= 3'd0;
        end else begin
            if (es_bajo) begin
                if (cont_bajo < N)
                    cont_bajo <= cont_bajo + 1;
                // Si ya alcanz  N, mantiene el valor (no incrementa m s)
            end else begin
                // Si la condici n deja de cumplirse, resetea el contador
                cont_bajo <= 3'd0;
            end
```

```
            // Contador de calor: incrementa mientras es_alto est  activo
            if (es_alto) begin
                if (cont_alto< N)
                    cont_alto <= cont_alto + 1;
                    // Si ya alcanz  N, mantiene el valor
            end else begin
                // Si la condici n deja de cumplirse, resetea el contador
                cont_alto <= 3'd0;
            end
        end
    end
    // L gica combinacional: generaci n de se ales de persistencia
    //La se al solo estar  activa cuando AMBAS condiciones sean verdaderas.
    assign per_bajo= (cont_bajo >= N-1) && es_bajo;
    assign per_alto= (cont_alto >= N-1) && es_alto;

endmodule
```

Listing 10: Modulo persistencia_ctr

```
`timescale 1ns / 1ps

module estado_temp(
    input logic clk,                            // reloj del sistema
    input logic arst_n,                         // reset asincrono
    input logic signed [10:0] temp_registrado, // temperatura
    input logic per_bajo,                       // Persistencia de fr o (del contador
        )
    input logic per_alto,                       // Persistencia de calor (del contador
        )
    output logic alerta,                        // se alizaci n de  alerta
    output logic calefactor,                    // Se al para calefactor
    output logic ventilador,                    // Se al para ventilador
    output logic [1:0] estado_actual            // estado FSM: NORMAL,FRIO, ALTO,
        ALERTA
);

    // Par metros de temperatura escalada
    parameter TEMP_BAJO = 180; //la temperatura como limite maximo para frio
    parameter TEMP_ALTO = 259; //la temperatura como limite maximo para alto

    // Codificaci n binaria de 2 bits para 4 estados
    // NORMAL: 00 - Temperatura en rango  ptimo
    // BAJO:   01 - Temperatura inferior a TEMP_BAJO (pre-alerta)
    // ALTO:   10 - Temperatura superior a TEMP_ALTO (pre-alerta)
    // ALERTA: 11 - Condici n persistente alcanzada
    typedef enum logic [1:0] {NORMAL=2'b00, BAJO=2'b01, ALTO=2'b10, ALERTA=2'b11}
        estado_t;
    estado_t estado; // Estado actual de la FSM
    // La FSM se actualiza en cada flanco positivo de reloj
    // Reset as ncrono: fuerza el estado a NORMAL inmediatamente
    always_ff (posedge clk, negedge arst_n) begin
        if (!arst_n) begin
            estado      <= NORMAL;    // Reset: sistema comienza en estado NORMAL
        end else begin
            // Transiciones seg n estado actual
            case (estado)
                NORMAL: begin
                    if (temp_registrado < TEMP_BAJO) estado <= BAJO;   // Detecci n
                        de fr o
                    else if (temp_registrado > TEMP_ALTO) estado <= ALTO;//
                        Detecci n de calor
                    // Si temperatura normal, permanece en NORMAL
                end
                BAJO: begin // Estado BAJO: pre-alerta de fr o
```

```
                    if (per_bajo) begin   // Persistencia alcanzada: transici n a ALERTA
                        estado <= ALERTA;
                    end else if (temp_registrado >= TEMP_BAJO && temp_registrado <= TEMP_ALTO) begin
                        // Recuperaci n: temperatura vuelve a normal
                        estado <= NORMAL;
                    end else if (temp_registrado > TEMP_ALTO) begin
                        // Cambio directo de fr o a calor
                        estado <= ALTO;
                    end
                    // Si sigue en fr o sin persistencia, permanece en BAJO
                end
                ALTO: begin   // Estado ALTO: pre-alerta de calor
                    if (per_alto)begin
                        // Persistencia alcanzada: transici n a ALERTA
                        estado <= ALERTA;

                    end else if (temp_registrado >= TEMP_BAJO && temp_registrado <= TEMP_ALTO) begin
                         // Recuperaci n: temperatura vuelve a normal
                        estado <= NORMAL;
                    end else if (temp_registrado < TEMP_BAJO) begin
                        // Cambio directo de calor a fr o
                        estado <= BAJO;
                    end
                    // Si sigue en calor sin persistencia, permanece en ALTO
                end
                ALERTA: begin // Estado ALERTA: condici n cr tica
                    if (temp_registrado >= TEMP_BAJO && temp_registrado <= TEMP_ALTO)
                         begin
                        // Recuperaci n: temperatura vuelve a normal
                        estado <= NORMAL;
                    end else if (temp_registrado < TEMP_BAJO && !per_bajo) begin
                        // La temperatura sigue baja pero perdi  persistencia
                        estado <= BAJO;
                    end else if (temp_registrado > TEMP_ALTO && !per_alto) begin
                        // La temperatura sigue alta pero perdi  persistencia
                        estado <= ALTO;
                    end
                    // Si sigue con persistencia, permanece en ALERTA
                end
                default: begin      // estado por defecto
                    estado <= NORMAL;
                end
            endcase

        end
    end

    // L gica de salida (combinacional)
    // Las salidas se generan combinacionalmente para respuesta inmediata
    // Los actuadores SOLO se activan en estado ALERTA
    always_comb begin
        // Valores por defecto (actuadores apagados)
        calefactor = 1'b0;
        ventilador = 1'b0;
        // En estado ALERTA, activa actuador seg n temperatura
        case (estado)
            ALERTA: begin
                if (temp_registrado < TEMP_BAJO)
                    calefactor = 1'b1;    // Persistencia de fr o

                else if (temp_registrado > TEMP_ALTO)
                    ventilador = 1'b1;    // Persistencia de calor
```

```
            end
            default: begin // En cualquier otro estado, actuadores apagados
            calefactor = 1'b0;
            ventilador = 1'b0;
            end
        endcase
    end
    assign alerta = (estado == ALERTA); // alerta activa solo en estado ALERTA
    assign estado_actual = estado;      // verificacion del estado actual

endmodule
```

Listing 11: Modulo estado_temp

```
`timescale 1ns / 1ps

module monitoreo_top(
    input  logic clk,                          // Reloj del sistema
    input  logic arst_n,                       // Reset as ncrono (activo en bajo)
    input  logic signed [10:0] temp_entrada,   // Temperatura  del sensor
    output logic alerta,                       // Indica falla
    output logic calefactor,                   // Se al para activar el calefactor
    output logic ventilador,                   // Se al para activar el ventilador
    output logic [1:0]estado_actual,           // Estado del monitoreo
    output logic [2:0] cont_bajo,              // Contador para temperatura baja
    output logic [2:0] cont_alto               // Contador para temperatura alta


);
    logic es_bajo_int, es_alto_int;            // Indicadores internos
    logic per_bajo_int, per_alto_int;          //

    // Se instancia los modulos
   // Compara la temperatura registrada contra los l mites
    comparador_temp compara1 (
        .temp_entrada       (temp_entrada),
        .es_bajo            (es_bajo_int),
        .es_alto            (es_alto_int)
    );



    persistencia_ctr #(.N(5)) u_persistencia (
        .clk            (clk),
        .arst_n         (arst_n),
        .es_bajo        (es_bajo_int),
        .es_alto        (es_alto_int),
        .cont_bajo      (cont_bajo),
        .cont_alto      (cont_alto),
        .per_bajo       (per_bajo_int),
        .per_alto       (per_alto_int)
    );


    // Decide el estado y activa los actuadores si persiste mas 5 ciclos de reloj el
        rango de temperatura
    estado_temp fsm1 (
        .clk                (clk),
        .arst_n             (arst_n),
        .temp_registrado    (temp_entrada),
        .per_bajo           (per_bajo_int),
        .per_alto           (per_alto_int),
        .alerta             (alerta),
        .calefactor         (calefactor),
        .ventilador         (ventilador),
```

```
        .estado_actual       (estado_actual)
    );

endmodule
```

Listing 12: Modulo monitoreo$_top$

Apéndice B: Verificación

```
// fv/fv_monitoreo.sv

module fv_monitoreo (
    // Agregar todas las se ales a vigilar
    input logic clk,                              // Reloj del sistema
    input logic arst_n,                           // Reset as ncrono activo en bajo
    input logic signed [10:0] temp_entrada,       // Temperatura escalada del sensor
        (-400 a 850)
    input logic alerta,                           // Indicador de condici n de alerta
    input logic ventilador,                       // Activaci n del sistema de
        calefacci n
    input logic calefactor,                       // Activaci n del sistema de
        ventilaci n
    input logic [1:0] estado_actual,              // Estado actual de la FSM para debug
    input logic [2:0] cont_bajo,                  // Contador de persistencia de fr o
        (0-5) para debug
    input logic [2:0] cont_alto                   // Contador de persistencia de calor
        (0-5)para debug

);

    // Verificacion Reset
    `AST(mon, alerta_en_reset,     (!arst_n) |->, (alerta == 0))
    `AST(mon, ventilador_en_reset, (!arst_n) |->, (ventilador == 0))
    `AST(mon, calefactor_en_reset, (!arst_n) |->, (calefactor == 0))

    // Persistencia en frio

    `AST(mon, persistencia_frio, ((temp_entrada < 11'sd180)[*6]) |->, (alerta == 1 &&
         calefactor == 1))

    // Persistencia en calor
    `AST(mon, persistencia_calor, ((temp_entrada > 11'sd259)[*6]) |->,(alerta == 1 &&
         ventilador == 1))

    //Estabilidad en normal
    `AST(mon, estabilidad_normal, (temp_entrada >= 11'sd180 && temp_entrada <= 11'
        sd259) |=>, (alerta == 0))

    // Recuperaci n a normal
    `AST(mon, recuperacion_normal, (estado_actual == 2'b00) |->, (alerta == 0))

    // 4. Exclusi n Mutua
    `AST(mon, exclusion, 1'b1 |->, (!(calefactor && ventilador)))

    //

    `AST(mon, salida_alerta,(alerta == 1 && $past(temp_entrada) >=11'sd180 && $past(
        temp_entrada) <=11'sd259) |=>,(alerta == 0))

    // El calefactor no debe encenderse si la temperatura no es baja
    `AST(mon, seguridad_calefactor,(calefactor == 1) |->, (alerta == 1 &&
        temp_entrada <11'sd180))

```

```
45      // El ventilador no debe encenderse si la temperatura no es alta
46
47      `AST(mon, seguridad_ventilador,(ventilador == 1) |->, (alerta == 1 &&
            temp_entrada >11'sd259))
48
49      // Para transiciones en los limites para temperaturas bajas
50
51      `AST(mon, transicion_frio_normal, ($past(temp_entrada) < 11'sd180 && temp_entrada
            inside {[11'sd180:11'sd259]}) |=>, (estado_actual == 2'b00))
52
53      //`AST(mon, transicion_frio_calor,($past(temp_entrada) < 11'sd180 && temp_entrada
            > 11'sd259) |=>,(estado_actual == 2'b10))
54      `AST(mon, transicion_frio_calor,($past(temp_entrada) < 180 && temp_entrada > 259)
            |=>,(estado_actual == 2'b10 || estado_actual == 2'b11))
55
56  endmodule
57
58  // Conexion de las se ales de fv_monitoreo con las se ales internas de
        monitoreo_top
59  bind monitoreo_top fv_monitoreo fv_mon_inst (.*);
```

Listing 13: modulo fv_monitoreo

```
1   // cov_monitoreo.sv
2
3   //`include "property_defines.svh"
4   import monitoreo_pkg::*;
5   module cov_monitoreo (
6       input logic clk,                          // Reloj del sistema
7       input logic arst_n,                       // Reset as ncrono activo en bajo
8       input logic signed [10:0] temp_entrada, // Temperatura escalada del sensor (-400
            a 850)
9       input logic [1:0]  estado_actual,         // Estado actual de la FSM para debug
10      input logic [2:0]  cont_bajo,             // Contador de persistencia de fr o
            (0-5) para debug
11      input logic [2:0]  cont_alto,             // Contador de persistencia de calor
            (0-5)para debug
12      input logic alerta,                       // Indicador de condici n de alerta
13      input logic ventilador,                   // Activaci n del sistema de
            ventilaci n
14      input logic calefactor                    // Activaci n del sistema de
            calefacci n
15  );
16
17        // ============================================================
18        // 1. COVERGROUP PRINCIPAL
19        // ============================================================
20      covergroup cg_monitoreo (posedge clk);
21          option.per_instance = 1;
22          option.name = "Cobertura_Funcional_Monitoreo";
23          option.comment = "Cobertura completa del sistema de temperatura";
24
25          // ------------------------------------------------
26          // Cobertura de Rangos de Temperatura
27          // ------------------------------------------------
28          cp_temp: coverpoint temp_entrada {
29              bins min_sensor       = { MIN_SENSOR };           //-400  Valor m nimo
30              bins rango_bajo     = { [MIN_SENSOR+1 : TEMP_BAJO-2] }; //-399:178 Fr o
                    (no cr tico)
31              bins limite_bajo      = { TEMP_BAJO - 1 };        // 179 Frontera fr o/
                    normal
32              bins limite_normal_inf  = { TEMP_BAJO };            // 180 Frontera
                    normal
33              bins rango_normal     = { [TEMP_BAJO+1  : TEMP_ALTO-1] }; // 181:259
                    Normal (estable)
```

```
34              bins limite_normal_sup  = { TEMP_ALTO };            // 259 Frontera
                    normal
35              bins limite_alto      = { TEMP_ALTO + 1 };        // 260 Frontera normal/
                    calor
36              bins rango_alto       = { [TEMP_ALTO+2  : MAX_SENSOR-1] };//261:849 Calor
                     (no cr tico)
37              bins max_sensor       = { MAX_SENSOR };           // 800 Valor m ximo
38          }
39
40          // ------------------------------------------------
41          // Cobertura de Estados de la FSM
42          // ------------------------------------------------
43          cp_estado: coverpoint estado_actual {
44              bins NORMAL = {2'b00};
45              bins BAJO   = {2'b01};
46              bins ALTO   = {2'b10};
47              bins ALERTA = {2'b11};
48              illegal_bins otros = default;
49          }
50
51          // ------------------------------------------------
52          // Cobertura del Contador de Persistencia
53          // ------------------------------------------------
54          cp_contador_bajo: coverpoint cont_bajo {
55              bins cero    = {0};
56              bins uno     = {1};
57              bins dos     = {2};
58              bins tres    = {3};
59              bins cuatro  = {4};
60              bins cinco   = {5};
61              bins otros   = {[6:7]};
62          }
63          cp_contador_alto: coverpoint cont_alto {
64              bins cero    = {0};
65              bins uno     = {1};
66              bins dos     = {2};
67              bins tres    = {3};
68              bins cuatro  = {4};
69              bins cinco   = {5};
70              bins otros   = {[6:7]};
71          }
72
73          // ------------------------------------------------
74          // Cobertura de Actuadores
75          // ------------------------------------------------
76          cp_actuadores: coverpoint {calefactor, ventilador} {
77              bins ambos_off = {2'b00};
78              bins solo_calefactor = {2'b10};
79              bins solo_ventilador = {2'b01};
80              illegal_bins ambos_on = {2'b11};  // Error
81          }
82
83          // ------------------------------------------------
84          // Cobertura de transiciones de la FSM
85          // ------------------------------------------------
86          cp_fsm_trans: coverpoint estado_actual {
87              // Transiciones normales
88              bins normal_a_bajo   = (2'b00 => 2'b01);  // NORMAL a BAJO
89              bins normal_a_alto   = (2'b00 => 2'b10);  // NORMAL a ALTO
90              bins bajo_a_normal   = (2'b01 => 2'b00);  // BAJO a NORMAL
91              bins alto_a_normal   = (2'b10 => 2'b00);  // ALTO a NORMAL
92
93              // Transiciones a ALERTA (casos cr ticos)
94              bins bajo_a_alerta   = (2'b01 => 2'b11);  // BAJO a ALERTA (persistencia
                    fr o)
```

```
95              bins alto_a_alerta   = (2'b10 => 2'b11);  // ALTO a ALERTA (persistencia
                    calor)
96
97              // Recuperaci n autom tica
98              bins recu_auto       = (2'b11 => 2'b00);  // ALERTA a NORMAL
99
100             // Transitorios
101             bins transitorio_bajo = (2'b00 => 2'b01 => 2'b00);  // Ruido de fr o
102             bins transitorio_alto = (2'b00 => 2'b10 => 2'b00);  // Ruido de calor
103         }
104
105         // ------------------------------------------------
106         // COMBINACIONES
107         // ------------------------------------------------
108
109         // 1. Estado vs Temperatura
110         estado_x_temp: cross cp_estado, cp_temp {
111             // Ignoramos combinaciones imposibles
112             ignore_bins estado_alerta_con_normal =
113                 binsof(cp_estado.ALERTA) && binsof(cp_temp.rango_normal);
114         }
115
116
117         estado_x_cont_bajo: cross cp_estado, cp_contador_bajo;
118         estado_x_cont_alto: cross cp_estado, cp_contador_alto;
119
120         cont_x_temp_bajo: cross cp_contador_bajo, cp_temp;
121         cont_x_temp_alto: cross cp_contador_alto, cp_temp;
122
123     endgroup
124
125     // ============================================================
126     // 2. INSTANCIACI N
127     // ============================================================
128     cg_monitoreo cg_inst = new();
129
130     // ============================================================
131     // 3. COVER PROPERTIES
132     // ============================================================
133     // Exclusi n mutua -
134     `COV(mon, exclusion_mutua_visto, 1'b1 |->, (!(ventilador && calefactor)))
135
136     // Alerta activada
137     `COV(mon, alerta_activada, 1'b1 |->, (alerta == 1))
138
139     // Contador lleg  a 5
140     `COV(mon, cont_bajo_cinco_alcanzado, 1'b1 |->, (cont_bajo == 5))
141     `COV(mon, cont_alto_cinco_alcanzado, 1'b1 |->, (cont_alto == 5))
142
143     // Estado ALERTA alcanzado
144     `COV(mon, estado_alerta_alcanzado, 1'b1 |->, (estado_actual == 2'b11))
145
146     // ============================================================
147     // 4. REPORTE FINAL
148     // ============================================================
149     final begin
150         $display("\n=== REPORTE DE COBERTURA FUNCIONAL ===");
151         $display("Cobertura de estados: %.1f%%", cg_inst.cp_estado.get_coverage());
152         $display("Cobertura de temperatura: %.1f%%", cg_inst.cp_temp.get_coverage());
153         $display("Cobertura de contador bajo: %.1f%%", cg_inst.cp_contador_bajo.
                get_coverage());
154         $display("Cobertura de contador alto: %.1f%%", cg_inst.cp_contador_alto.
                get_coverage());
155         $display("Cobertura de actuadores: %.1f%%", cg_inst.cp_actuadores.
                get_coverage());
```

```
156         $display("Cobertura de transiciones: %.1f%%", cg_inst.cp_fsm_trans.
                get_coverage());
157         $display("Cobertura total: %.1f%%", cg_inst.get_coverage());
158         $display("====================================\n");
159     end
160
161 endmodule
162
163
164 bind monitoreo_top cov_monitoreo cov_mon_inst (.*);
```

Listing 14: cov_monitoreo.sv

```
1  interface interface_monitoreo(input logic clk, input logic arst_n);
2      logic signed [10:0] temp_entrada;  // Temperatura de entrada al DUT (11 bits con
           signo)
3      logic alerta;                      // Indicador de alerta (del DUT)
4      logic calefactor;                  // Se al de control de calefacci n (del DUT)
5      logic ventilador;                  // Se al de control de ventilaci n (del DUT)
6      logic [1:0] estado_actual;         // Estado actual de la FSM (del DUT) para
           debug
7      logic [2:0]  cont_alto;            // Contador de persistencia de fr o (del DUT)
            para debug
8      logic [2:0]  cont_bajo;            // Contador de persistencia de calor (del DUT)
            para debug
9      //
10
11     //
12     task automatic enviar_temperatura(input logic signed [10:0] valor);
13         (negedge clk);                                  // Sincronizar con flanco de
               bajada
14         temp_entrada = valor;                           // Asigna valor a la se al
15         $display("[INTERFACE] Enviando T = %0d", valor);
16         (posedge clk);                                  // Esperar al siguiente flanco
               de subida
17     endtask
18     //
19     task automatic reporte_estado();          //Muestra en consola el estado completo
            del sistema,
20         $display("[INTERFACE]  %t | cont_bajo:%0d | cont_alto:%0d | Alerta: %b | Vent
               : %b | Cal: %b | Estado: %b",
21                  $time, cont_bajo,cont_alto, alerta,ventilador, calefactor,
                        estado_actual);
22     endtask
23
24 endinterface
```

Listing 15: interface_monitoreo

```
1  package monitoreo_pkg;
2      // Par metros de l mites de alerta
3      parameter TEMP_BAJO = 180;  // 18.0 C  - Umbral para condici n de fr o
4      parameter TEMP_ALTO = 259;  // 25.9 C  - Umbral para condici n de calor
5
6      // L mites  de operaci n del sensor
7      parameter MIN_SENSOR = -400;   // -40.0 C  - M nimo del sensor
8      parameter MAX_SENSOR = 850;    // 85.0 C   - M ximo del sensor
9
10     // --- Clase padre
11     class temp_base;
12         rand logic signed [10:0] valor;
13
14         // Restricci n: valores dentro del rango f sico del sensor
15         constraint c_fisico { valor inside {[MIN_SENSOR: MAX_SENSOR ]}; }
```

```
16          // Constructor: inicializa valor a 0
17          function new();
18              this.valor = 0;
19          endfunction
20
21          // Visualizacion de dato en consola
22          virtual function void reportar();
23              $display("[monitoreo_pkg] Temperatura Generada: %0d.%0d C", valor/10, (
                    valor < 0) ? -(valor%10) : (valor%10));
24          endfunction
25      endclass
26
27      // --- Clases Hijas
28
29      class temp_bajo extends temp_base;
30          // Restricci n: valores estrictamente menores a TEMP_BAJO
31          constraint c_rango { valor < TEMP_BAJO; }
32
33          virtual function void reportar();
34              $display("[ESTADO: FRIO] registrando temperatura baja: %0d.%0d C", valor
                    /10, (valor < 0) ? -(valor%10) : (valor%10));
35          endfunction
36
37      endclass
38
39      class temp_normal extends temp_base;
40          // Restricci n: valores dentro del rango normal
41          constraint c_rango { valor inside {[TEMP_BAJO : TEMP_ALTO]}; }
42
43          virtual function void reportar();
44              $display("[ESTADO: NORMAL] registrando temperatura estable: %0d.%0d C",
                    valor/10, (valor < 0) ? -(valor%10) : (valor%10));
45          endfunction
46
47      endclass
48
49      class temp_alto extends temp_base;
50          // Restricci n: valores estrictamente mayores a TEMP_ALTO
51          constraint c_rango { valor > TEMP_ALTO; }
52
53          virtual function void reportar();
54              $display("[ESTADO: ALTO] registrando temperatura alta: %0d.%0d C", valor
                    /10, (valor < 0) ? -(valor%10) : (valor%10));
55          endfunction
56
57      endclass
58  //Simula oscilaciones de temperatura alrededor de un valor inicial
59      class temp_persistente extends temp_base;
60          // Guarda el valor anterior
61          logic signed [10:0] ultimo_valor;
62          // Margen de oscilaci n (por defecto  2 )
63          int delta = 2;
64           // Restricci n: el nuevo valor debe estar cerca del anterior
65          constraint c_cercania { valor inside {[ultimo_valor - delta : ultimo_valor +
                delta]}; }
66
67          function new(logic signed [10:0] inicio = 250);
68              super.new();
69              this.ultimo_valor = inicio;
70              this.valor = inicio;
71          endfunction
72
73          function void post_randomize();
74              this.ultimo_valor = this.valor;
75          endfunction
```

```
76
77          virtual function void reportar();
78              $display("[ESTADO: PERSISTENTE] registrando temperatura persistente: %0d
                    .%0d C", valor/10, valor%10);
79          endfunction
80      endclass
81  // Restricci n: valores fuera del rango del sensor
82      class temp_fuera_rango extends temp_base;
83          constraint c_fisico { valor inside {[-1024 : -401], [851   : 1023]};}
84
85          function new();
86              super.new();
87          endfunction
88
89
90          virtual function void reportar();
91              $display("[SENSADO EXTREMO] Valor fuera de rango detectado: %0d", valor);
92          endfunction
93      endclass
94
95  endpackage : monitoreo_pkg
```

Listing 16: monitoreo_pkg

```
1  //`define T_NORMAL_B
2  //`define T_BAJO_INM
3  //`define T_ALTO_INM
4  //`define T_NORMAL_PERS
5  //`define T_BAJO_PERS
6  //`define T_ALTO_PERS
7  //`define T_TRANS_ALTO
8  //`define T_TRANS_BAJO
9  //`define T_LIM_BAJO
10 //`define T_LIM_ALTO
11 //`define T_RECU_AUTO
12 //`define T_BAJO_ALTO
13 //`define T_ALTO_BAJO
14 `define RESET_SISTEMA
15 //`define LIMITE_INFERIOR
16 //`define LIMITE_SUPERIOR
17
18
19 `timescale 1ns / 1ps
20 import monitoreo_pkg::*;
21
22 module monitoreo_tb();
23     bit clk;         //reloj del sistema
24     bit arst_n;      // reset asincrono activo en bajo
25
26     // Instancia de la interfaz
27     interface_monitoreo intf(clk, arst_n);
28
29     // Instancia del dut
30     monitoreo_top dut (
31         .clk(intf.clk),
32         .arst_n(intf.arst_n),
33         .temp_entrada(intf.temp_entrada),
34         .alerta(intf.alerta),
35         .calefactor(intf.calefactor),
36         .ventilador(intf.ventilador),
37         .estado_actual(intf.estado_actual),
38         .cont_alto (intf.cont_alto),
39         .cont_bajo (intf.cont_bajo)
40     );
41     // Inicializacion de reloj
```

```
42      initial clk = 0;
43      always #5ns clk = ~clk;  // Periodo 10ns (100 MHz)
44
45      initial begin
46          arst_n = 0;           // Reset activo desde tiempo 0
47          intf.temp_entrada = 220;
48          #20ns;                // Espera 2 ciclos completos
49          arst_n = 1;           // Libera reset
50          $display("
                ==========================================================================================
                ");
51          $display("[TB] Reset liberado en tiempo %0t", $time);
52      end
53
54      // instanciar objetos de la clase
55      temp_bajo           t_frio;
56      temp_alto           t_calor;
57      temp_normal         t_normal;
58      temp_persistente    t_pers;
59      temp_fuera_rango    t_limite;
60
61      initial begin
62
63          wait(arst_n == 1'b1);
64          (posedge clk);
65
66          // inicializacion
67          t_frio = new();
68          t_calor =new();
69          t_normal = new();
70          t_pers = new(150);
71          t_limite = new();
72
73          $display("--- Iniciando simulacion ---");
74
75          `ifdef RESET_SISTEMA
76              repeat(10)begin
77              ejecutar_reset();
78              end
79          `endif
80
81          `ifdef LIMITE_INFERIOR
82              repeat(10000)begin
83              test_limite_inferior ();
84              end
85          `endif
86
87          `ifdef LIMITE_SUPERIOR
88              repeat(10000)begin
89              test_limite_superior ();
90              end
91          `endif
92
93  //.............................Casos operacion basica.............................
94          `ifdef T_NORMAL_B
95              repeat(10000)begin
96              test_normal();
97              end
98          `endif
99          `ifdef T_BAJO_INM
100             repeat(10000)begin
101             test_bajo ();
102             end
103         `endif
104         `ifdef T_ALTO_INM
```

```
105             repeat(1000)begin
106             test_alto ();
107             end
108         `endif
109 //.............................Casos operacion persistente
    ..............................
110         `ifdef T_NORMAL_PERS
111             repeat(10000)begin
112                 test_persistencia_normal();
113             end
114         `endif
115
116         `ifdef T_BAJO_PERS
117             repeat(10000) begin
118             test_persistencia_bajo();
119             end
120         `endif
121
122         `ifdef T_ALTO_PERS
123             repeat(10000) begin
124             test_persistencia_alto();
125             end
126         `endif
127
128 //.............................Casos recuperacion
    ....................................
129         `ifdef T_RECU_AUTO
130             repeat(10000)begin
131             test_recuperacion_bajo ();
132             end
133         `endif
134
135         `ifdef T_BAJO_ALTO
136             repeat(10000)begin
137             test_bajo_alto ();
138             end
139         `endif
140         `ifdef T_ALTO_BAJO
141             repeat(5)begin
142             test_alto_bajo ();
143             end
144         `endif
145
146 //.............................Casos transitorios
    ....................................
147         `ifdef T_TRANS_ALTO
148             repeat(10000)begin
149             test_transitorio_alto();
150             end
151         `endif
152
153         `ifdef T_TRANS_BAJO
154             repeat(10000)begin
155             test_transitorio_bajo();
156             end
157         `endif
158 //.............................Casos de limites.....................................
159         `ifdef T_LIM_BAJO
160             repeat(1)begin
161             test_limite_bajo();
162             end
163         `endif
164
165         `ifdef T_LIM_ALTO
166             repeat(1)begin
```

```
            test_limite_alto();
            end
        `endif

        repeat(5) (posedge clk);

        $display("--- Simulaci n terminada ---");
        $finish;
    end


// casos de prueba
//=============================================================================


//.............................. Tarea de Reset ....................................
    task ejecutar_reset();

    if(!t_frio.randomize()) $fatal("No se pudo generar valor");
    t_pers = new(t_frio.valor);
    repeat(5) begin
        void'(t_pers.randomize() with { valor < 180; });
        intf.enviar_temperatura(t_pers.valor);
        intf.reporte_estado();
    end
    (posedge clk);
    intf.reporte_estado();

    arst_n = 0;                  // Activar reset
    repeat(2) (posedge clk);    // Mantener 2 ciclos
    arst_n = 1;                  // Liberar reset
    (posedge clk);              // Esperar 1 ciclo para estabilizar
    intf.reporte_estado();

        /*$display("\n[TEST CASE] T_RESET - Verificando reset en diferentes estados")
            ;
        if(!t_frio.randomize()) $fatal("No se pudo generar valor");
        t_frio.reportar();
        intf.enviar_temperatura(t_frio.valor); // Usa Task de Interfaz
        intf.reporte_estado();
        // Reset en estado NORMAL
        $display("[TB] Reset en estado NORMAL");
        arst_n = 0;
        (posedge clk);
        arst_n = 1;
        (posedge clk);
        intf.reporte_estado();


        // Verificar aserciones de reset
        if (intf.alerta != 0) $error("[TB] alerta_en_reset no se cumpli ");
        if (intf.calefactor != 0) $error("[TB] calefactor_en_reset no se cumpli ");
        if (intf.ventilador != 0) $error("[TB] ventilador_en_reset no se cumpli ");
        //Forzando en alerta en temperatura baja
        test_persistencia_bajo();
        (posedge clk);

        if (intf.estado_actual != 2'b11) begin
            $error("[TB] No se alcanz  ALERTA para reset");
            return;
        end

        // Reset durante ALERTA
        $display("[TB] ..................................Reset durante ALERTA
            ..............................................");
```

```
230         arst_n = 0;
231         (posedge clk);
232         (posedge clk);
233         arst_n = 1;
234         (posedge clk);
235         intf.reporte_estado();
236         //Forzando en alerta en temperatura alta
237         test_persistencia_alto();
238         (posedge clk);
239 
240         if (intf.estado_actual != 2'b11) begin
241             $error("[TB] No se alcanz  ALERTA para reset");
242             return;
243         end
244         // Reset durante ALERTA
245         $display("[TB] ..............................Reset durante ALERTA
                .............................................");
246         arst_n = 0;
247         (posedge clk);
248         (posedge clk);
249         arst_n = 1;
250         (posedge clk);
251         intf.reporte_estado();
252         // Verificar nuevamente
253         if (intf.alerta != 0) $error("[TB] alerta_en_reset (en ALERTA) no se cumpli
                ");
254         if (intf.calefactor != 0) $error("[TB] calefactor_en_reset (en ALERTA) no se
                cumpli ");
255         if (intf.ventilador != 0) $error("[TB] ventilador_en_reset (en ALERTA) no se
                cumpli ");*/
256 
257     endtask
258 //.........................Casos operacion basica.........................
259     task test_normal ();
260         $display("[TEST CASE] T_NORMAL_B");
261         repeat(4) begin
262             if(!t_normal.randomize()) $fatal("No se pudo generar valor ");
263             t_normal.reportar();
264             intf.enviar_temperatura(t_normal.valor); // Usa Task de Interfaz
265             intf.reporte_estado();
266             $display("--- Dato ingresado al DUT exitosamente ---");
267             end
268     endtask
269 
270     task test_bajo ();
271         $display("[TEST CASE] T_BAJO_INM");
272         repeat(4) begin
273             if(!t_frio.randomize()) $fatal("No se pudo generar valor");
274             t_frio.reportar();
275             intf.enviar_temperatura(t_frio.valor); // Usa Task de Interfaz
276             intf.reporte_estado();
277             $display("--- Dato ingresado al DUT exitosamente ---");
278             end
279     endtask
280 
281     task test_alto ();
282         $display("[TEST CASE] T_ALTO_INM");
283         repeat(4) begin
284             if(!t_calor.randomize()) $fatal("No se pudo generar valor ");
285             t_calor.reportar();
286             intf.enviar_temperatura(t_calor.valor); // Usa Task de Interfaz
287             intf.reporte_estado();
288             $display("--- Dato ingresado al DUT exitosamente ---");
289             end
290     endtask
```

```
//.............................Casos operacion persistente
    ..............................
    task test_persistencia_normal();
        $display("\n[TEST CASE] T_NORMAL_PERS - Verificando alerta tras 6 ciclos...")
            ;
        if(!t_normal.randomize()) $fatal("No se pudo generar valor de frio inicial");
        t_pers = new(t_normal.valor); // Forzamos inicio en zona rango normal
        repeat(6) begin
            if(!t_pers.randomize()with { valor inside {[180 : 259]}; }) $fatal("");
            t_pers.reportar();
            intf.enviar_temperatura(t_pers.valor);
            intf.reporte_estado();
        end
    endtask

    task test_persistencia_bajo();
        $display("\n[TEST CASE] T_BAJO_PERS - Verificando alerta tras 6 ciclos...");
        if(!t_frio.randomize()) $fatal("No se pudo generar valor de frio inicial");
        t_pers = new(t_frio.valor); // Forzamos inicio en zona de fr o
        repeat(10) begin
            if(!t_pers.randomize()with { valor < 180; }) $fatal("No se pudo generar
                valor ");
            t_pers.reportar();
            intf.enviar_temperatura(t_pers.valor);
            intf.reporte_estado();
        end
    endtask

    task test_persistencia_alto();
        $display("\n[TEST CASE] T_ALTO_PERS - Verificando alerta tras 6 ciclos...");
        if(!t_calor.randomize()) $fatal("No se pudo generar valor de calor inicial");

        t_pers = new(t_calor.valor); // Forzamos inicio en zona de calor
        repeat(6) begin
            if(!t_pers.randomize() with { valor > 259; }) $fatal("No se pudo generar
                valor ");
            t_pers.reportar();
            intf.enviar_temperatura(t_pers.valor);
            intf.reporte_estado();
        end
    endtask
//.............................Casos recuperacion
    ....................................
    task test_recuperacion_bajo ();
            $display("\n[TEST CASE] T_RECU_AUTO - Forzando Alerta y luego
                Recuperaci n BAJO a NORMAL");
            // 1. Forzando persistencia
            if(!t_frio.randomize()) $fatal("No se pudo generar valor de frio inicial
                ");
            t_pers = new(t_frio.valor); // Forzamos inicio en zona de fr o
            $display("\n[TB] Temperatura %0d", t_pers.valor);
            repeat(5) begin
                void'(t_pers.randomize() with { valor < 180; });
                intf.enviar_temperatura(t_pers.valor);
                intf.reporte_estado();
            end
            (posedge clk);

            intf.reporte_estado();
            if (intf.estado_actual !== 2'b11) begin
                $error("[FAIL] No se alcanz  ALERTA");
                intf.reporte_estado();
                return;
            end

```

```
349             // 2. probando la recuperacion a normal
350             $display("[TB] Enviando temperatura normal  para recuperar...");
351             if(!t_normal.randomize()) $fatal("No se pudo generar valor de normal
                    inicial");
352             repeat(5) begin
353                 intf.enviar_temperatura(t_normal.valor);
354                 intf.reporte_estado();
355             end
356              (posedge clk);
357             // procesando el cambio
358             intf.reporte_estado();
359             if (intf.alerta == 0 && intf.estado_actual == 2'b00) begin
360                 $display("[TB] El sistema se normaliz  correctamente.");
361                 intf.reporte_estado();
362             end else begin
363                 $display("[TB] El sistema NO se recuper . Estado: %b, Alerta: %b",
364                          intf.estado_actual, intf.alerta);
365             end
366     endtask
367
368     task test_bajo_alto ();
369             $display("\n[TEST CASE] T BAJO A ALTO - Forzando Alerta y luego
                    Recuperaci n BAJO a ALTO");
370             // 1. Forzando persistencia
371             if(!t_frio.randomize()) $fatal("No se pudo generar valor de frio inicial
                    ");
372             t_pers = new(t_frio.valor); // Forzamos inicio en zona de fr o
373             $display("\n[TB] Temperatura %0d", t_pers.valor);
374             repeat(5) begin
375                 void'(t_pers.randomize() with { valor < 180; });
376                 intf.enviar_temperatura(t_pers.valor);
377                 intf.reporte_estado();
378             end
379             (posedge clk);
380
381             intf.reporte_estado();
382             if (intf.estado_actual !== 2'b11) begin
383                 $error("[FAIL] No se alcanz  ALERTA");
384                 intf.reporte_estado();
385                 return;
386             end
387
388             // 2. probando la recuperacion a alto
389             $display("[TB] Enviando temperatura alta ...");
390             if(!t_calor.randomize()) $fatal("No se pudo generar valor alto inicial");
391             repeat(2)begin
392             intf.enviar_temperatura(t_calor.valor);
393             // procesando el cambio
394             intf.reporte_estado();
395             end
396             (posedge clk);
397
398     endtask
399
400     task test_alto_bajo ();
401             $display("\n[TEST CASE] T ALTO A BAJO - Forzando Alerta y luego
                    Recuperaci n ALTO a BAJO");
402             // 1. Forzando persistencia
403             if(!t_calor.randomize()) $fatal("No se pudo generar valor de calor
                    inicial");
404             t_pers = new(t_calor.valor); // Forzamos inicio en zona de calor
405             $display("\n[TB] Temperatura %0d", t_pers.valor);
406             repeat(5) begin
407                 void'(t_pers.randomize() with { valor > 259; });
408                 intf.enviar_temperatura(t_pers.valor);
```

```
409                 intf.reporte_estado();
410             end
411             (posedge clk);
412 
413             intf.reporte_estado();
414             if (intf.estado_actual !== 2'b11) begin
415                 $error("[FAIL] No se alcanz  ALERTA");
416                 intf.reporte_estado();
417                 return;
418             end
419 
420             // 2. probando la recuperacion a bajo
421             $display("[TB] Enviando temperatura a ...");
422             if(!t_frio.randomize()) $fatal("No se pudo generar valor alto inicial");
423             intf.enviar_temperatura(t_frio.valor);
424              (posedge clk);
425             // procesando el cambio
426             intf.reporte_estado();
427 
428     endtask
429 //.............................Casos transitorio.....................................
430     task test_transitorio_alto();
431         $display("\n[TEST CASE] T_TRANS_ALTO - Calor transitorio (< N ciclos)");
432         // 1. Forzando persistencia
433         if(!t_calor.randomize()) $fatal("No se pudo generar valor de frio inicial");
434             t_pers = new(t_calor.valor); // Forzamos inicio en zona de fr o
435             repeat(3) begin
436             void'(t_pers.randomize());
437             intf.enviar_temperatura(t_pers.valor);
438             intf.reporte_estado();
439         end
440         (posedge clk);
441         // 2. Volvemos a temperatura NORMAL antes de que se cumpla la persistencia
442         $display("[TB] Enviando temperatura normal  para recuperar...");
443         if(!t_normal.randomize()) $fatal("No se pudo generar valor de frio inicial");
444         intf.enviar_temperatura(t_normal.valor);
445         (posedge clk);
446 
447         // 3. Verificaci n final
448         intf.reporte_estado();
449         if (intf.alerta == 0 && intf.estado_actual == 2'b00) begin
450             $display("[TB] El transitorio fue ignorado correctamente. Alerta: 0");
451         end else begin
452             $error("[TB]  La  alerta se dispar  o el estado no volvi  a NORMAL!");
453         end
454     endtask
455 
456     task test_transitorio_bajo();
457         $display("\n[TEST CASE] T_TRANS_BAJO - Fr o transitorio (< N ciclos)");
458         if(!t_frio.randomize()) $fatal("Randomize failed");
459         t_pers = new(t_frio.valor);
460         repeat(3) begin
461             void'(t_pers.randomize());
462             intf.enviar_temperatura(t_pers.valor);
463             intf.reporte_estado();
464         end
465         (posedge clk);
466 
467         $display("[TB] Volviendo a temperatura normal...");
468         if(!t_normal.randomize()) $fatal("No se pudo generar valor ");
469         intf.enviar_temperatura(t_normal.valor);
470         (posedge clk);
471 
472         intf.reporte_estado();
473         if (intf.alerta == 0 && intf.estado_actual == 2'b00)
```

```
            $display("[TB] Transitorio ignorado correctamente");
        else
            $error("[TB] Transitorio activ  alerta");
    endtask


//.............................Casos limite.....................................
    task test_limite_bajo();
        $display("\n Temperatura de un ciclo anterior: %0d",intf.temp_entrada);
        $display("\n[TEST CASE] T_LIM_BAJO - L mite exacto (179 a 180)");
        intf.enviar_temperatura(179);
        repeat(1) (posedge clk);
        intf.reporte_estado();
        intf.enviar_temperatura(180);
        repeat(1) (posedge clk);
        intf.reporte_estado();

        if (intf.estado_actual == 2'b00)
            $display("[TB] Transici n 179  180   correcta");
        else
            $error("[TB] Estado despu s de 180 = %b", intf.estado_actual);
    endtask


    task test_limite_alto();
        $display("\n Temperatura de un ciclo anterior: %0d",intf.temp_entrada);
        $display("\n[TEST CASE] T_LIM_ALTO - L mite exacto (259 a 260)");
        intf.enviar_temperatura(259);
        repeat(1) (posedge clk);
        intf.reporte_estado();
        intf.enviar_temperatura(260);
        repeat(1) (posedge clk);
        intf.reporte_estado();

        if (intf.estado_actual == 2'b10)
            $display("[TB] Transici n 259  260   correcta");
        else
            $error("[TB] Estado despu s de 260 = %b", intf.estado_actual);
    endtask


    //.............................Casos limite.....................................

    task test_limite_inferior ();
        $display("[TEST CASE] T_LIMITE_INFERIOR");

        if(!t_limite.randomize() with { valor inside {[-1024 : -401]}; }) $fatal("No
            se pudo generar valor ");

        t_limite.reportar();
        intf.enviar_temperatura(t_limite.valor);
        intf.reporte_estado();
        $display("--- Dato ingresado al DUT exitosamente ---");

    endtask
    task test_limite_superior ();
        $display("[TEST CASE] T_LIMITE_SUPERIOR");

        if(!t_limite.randomize() with { valor inside {[851 : 1023]}; }) $fatal("No se
             pudo generar valor ");

        t_limite.reportar();
        intf.enviar_temperatura(t_limite.valor);
        intf.reporte_estado();
        $display("--- Dato ingresado al DUT exitosamente ---");
```

```
537
538     endtask
539 //================================================================================
540
541
542     initial begin
543         $shm_open("shm_db");
544         $shm_probe("ASMTR");
545     end
546
547 endmodule
```

Listing 17: monitoreo_tb