CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA

Video 01 — Parte 1A: Lenguajes, Diagramas de Flujo, Algoritmos e Instrucciones en Sistemas Embebidos — Sistemas Embebidos de Tiempo Real, Jerarquía de Lenguajes, Conjunto ISA, WCET, Diagramas ISO 5807, FSM y Linker Script

Parte del Curso: Parte 1A (Tema 2A) — Lenguajes, Diagramas de Flujo, Algoritmos e Instrucciones en Sistemas Embebidos
Enfoque de Programación y Diagnóstico: Repertorio de Instrucciones ISA (Load-Store, MAC, BIC), Ciclo Fetch-Decode-Execute, Acotamiento WCET, Diagramas ISO 5807, FSM Moore/Mealy y Linker Script

$$t_exec = (N_inst \times CPI) / f_clk <= WCET_max < Ts \mid FSM\ Moore: y[k] = \lambda(s[k]) \mid FSM\ Mealy: y[k] = \lambda(s[k], x[k])$$

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Determinismo en sistemas embebidos: diferencia entre Tiempo Real Duro (Hard Real-Time) y Tiempo Real Blando; presupuesto estricto de memoria Flash y SRAM.
- Jerarquía de lenguajes en mecatrónica: Ensamblador para rutinas críticas del núcleo, C/C++ Embebido (MISRA-C) para control directo de registros sin recolector de basura, IEC 61131-3 para PLC y Python para supervisión y bancos de prueba.
- Conjunto de Instrucciones (ISA): Opcode y operandos, arquitectura Load-Store (LDR/STR), instrucción DSP MAC en un ciclo, máscaras lógicas (AND, ORR, EOR, BIC), ciclo Fetch-Decode-Execute y modos de direccionamiento (inmediato, por registro e indirecto con offset).
- Diseño de algoritmos con WCET acotado, diagramas de flujo normalizados ISO 5807, máquinas de estados finitos (FSM de Moore y de Mealy) codificadas sin bloqueos delay() y cadena de compilación cruzada gobernada por el Linker Script (.ld).

2. Parámetros Nominales de Tiempo Real, Memoria y Confiabilidad (Video 01)

Parámetro de Ingeniería	Símbolo	Valor Nominal	Función en el Sistema Embebido / Diagnóstico de Fallas
Periodo de Muestreo del Super-Loop	`Ts [ms]`	1 ms	Plazo determinista del ciclo de control cooperativo
Cota Máxima de Ejecución (WCET)	`WCET_max [µs]`	220 µs	Tiempo máximo permitido para completar todas las tareas sin violar el tiempo real
Ventana del Temporizador Vigilante	`[T_min, T_max]`	[1.5, 5] ms	Ventana válida de refresco del Windowed Watchdog por banderas de latido
Tamaño del Ring Buffer y Ventana	`N_FIFO / M_win`	FIFO = 32 B, Win = 8	Cola circular potencia de 2 para ISR y arreglo de filtrado de media móvil
Tiempo Medio Entre Fallas / Reparación	`MTBF / MTTR`	30000 h / 2 h	Indicadores de confiabilidad y mantenibilidad para seguridad funcional

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Sistemas Embebidos en Mecatrónica: Determinismo, Restricciones de Memoria y Tiempo Real

Diferencia entre Tiempo Real Duro (Hard Real-Time) y Tiempo Real Blando, Plazos Temporales y Presupuesto de Recursos

- **Turno 1 (Masculina):** A diferencia de una computadora de escritorio diseñada para maximizar el rendimiento promedio en múltiples aplicaciones abiertas, un sistema embebido mecatrónico está dedicado a gobernar una planta física bajo estrictas restricciones de tiempo, memoria y energía.
- **Turno 2 (Femenina):** La propiedad crítica de estos controladores es el determinismo temporal: en un sistema de tiempo real duro, entregar un cálculo correcto un milisegundo después del plazo límite equivale a una falla catastrófica de estabilidad o seguridad.
- **Turno 3 (Masculina):** Mientras que en un sistema de tiempo real blando un leve retraso solo degrada la fluidez visual de una pantalla, en el control de corriente de un servomotor o en el disparo de un freno de emergencia el periodo de muestreo es innegociable.
- **Turno 4 (Femenina):** Además, el ingeniero debe diseñar programas capaces de operar durante años de forma ininterrumpida dentro de unas pocas decenas de kilobytes de memoria SRAM y cientos de kilobytes de memoria Flash, sin fugas ni fragmentación de recursos.

Diapositiva 02: Lenguajes en Programación de Controladores: Ensamblador, C/C++ Embebido, IEC 61131-3 y Python

Criterios de Selección según el Nivel de Abstracción, Eficiencia de Acceso a Hardware y Seguridad Crítica (MISRA-C)

- **Turno 1 (Masculina):** La programación de controladores mecatrónicos se organiza en una jerarquía de lenguajes donde cada herramienta responde a un compromiso preciso entre control directo del silicio, productividad de desarrollo y portabilidad.
- **Turno 2 (Femenina):** En la base se encuentra el lenguaje ensamblador, indispensable en rutinas de arranque y cambios de contexto de sistemas operativos, donde necesitamos controlar cada registro del procesador y cada ciclo individual de reloj.
- **Turno 3 (Masculina):** Sobre él domina el estándar de oro de los sistemas embebidos: C y C más más bajo directrices de seguridad como MISRA C, que combinan estructuras de alto nivel con aritmética de punteros para manipular registros mapeados en memoria sin recolector de basura.
- **Turno 4 (Femenina):** En el ámbito de la automatización industrial empleamos los cinco lenguajes de la norma IEC sesenta y un mil ciento treinta y uno guion tres para P L C, mientras que Python lidera la supervisión, visión artificial y bancos de prueba automatizados.

Diapositiva 03: Conjunto de Instrucciones (ISA): Operaciones Aritméticas, Lógicas, Transferencia y Salto

Anatomía del Opcode y Operandos, Arquitectura Load-Store (LDR/STR), Instrucciones MAC y Operaciones de Bits

- **Turno 1 (Masculina):** Cualquier algoritmo escrito en lenguaje C o en diagrama de escalera termina traduciéndose al Conjunto de Instrucciones de la Arquitectura del procesador, conocido como ISA, que define el vocabulario binario que comprende la unidad central.
- **Turno 2 (Femenina):** Cada instrucción de máquina consta de un código de operación u Opcode, que especifica qué acción ejecutar, acompañado de los campos de operandos que identifican los registros de origen, el registro de destino o una constante inmediata.
- **Turno 3 (Masculina):** En arquitecturas RISC como ARM Cortex-M y RISC-V, las instrucciones se dividen en cuatro familias: transferencia de datos mediante carga LDR y almacenamiento STR, operaciones aritméticas como suma, resta y multiplicación con acumulación MAC...
- **Turno 4 (Femenina):** ...operaciones lógicas bit a bit como AND, OR, XOR y limpieza de bits BIC para gobernar periféricos, e instrucciones de salto condicional y llamada a subrutina que alteran el flujo secuencial del contador de programa.

Diapositiva 04: Ciclo de Instrucción (Fetch-Decode-Execute), Modos de Direccionamiento y Ciclos de Reloj

Ejecución Segmentada en Pipeline, Direccionamiento Inmediato, por Registro e Indirecto con Desplazamiento

- **Turno 1 (Masculina):** Para ejecutar cada instrucción del programa, el procesador realiza de manera incesante un ciclo síncrono de tres fases fundamentales gobernado por el oscilador de reloj: búsqueda o Fetch, decodificación o Decode y ejecución o Execute.
- **Turno 2 (Femenina):** En la fase de búsqueda, el núcleo lee de la memoria Flash la instrucción apuntada por el Contador de Programa PC; en la decodificación interpreta el Opcode; y en la ejecución la unidad aritmético-lógica procesa los datos y actualiza las banderas de estado.
- **Turno 3 (Masculina):** Los modos de direccionamiento determinan cómo se localizan los operandos: el modo inmediato incrusta la constante dentro de la instrucción, mientras que el modo por registro opera a máxima velocidad directamente sobre el banco interno R cero a R doce.
- **Turno 4 (Femenina):** Especialmente poderoso en mecatrónica es el direccionamiento indirecto por registro con desplazamiento, el cual toma una dirección base en un puntero y suma un offset constante para acceder en un solo ciclo a los registros de cualquier periférico.

Diapositiva 05: Diseño de Algoritmos en Mecatrónica: Corrección, Complejidad Temporal y Acotamiento WCET

Finitud, Estabilidad Numérica, Prohibición de Recursión Abierta y Cálculo del Peor Tiempo de Ejecución

- **Turno 1 (Masculina):** Un algoritmo de control es una secuencia finita, ordenada y libre de ambigüedades de pasos computacionales que transforma las lecturas crudas de los sensores en comandos seguros hacia los actuadores dentro de un intervalo de tiempo conocido.
- **Turno 2 (Femenina):** En informática general suele evaluarse la complejidad asintótica promedio de un algoritmo, pero en ingeniería mecatrónica la métrica decisiva es el WCET: el Tiempo de Ejecución en el Peor Caso posible.
- **Turno 3 (Masculina):** Para garantizar que el WCET sea estrictamente menor que el periodo de muestreo del lazo de control, todo bucle dentro del algoritmo debe poseer una cota superior fija de iteraciones y jamás debemos emplear recursión abierta.
- **Turno 4 (Femenina):** Asimismo, el diseño algorítmico debe contemplar la estabilidad numérica frente al desbordamiento de enteros, la división por cero cuando una ganancia o intervalo se anula, y la pérdida de precisión al restar magnitudes casi iguales en punto flotante.

Diapositiva 06: Diagramas de Flujo Normalizados (ISO 5807): Símbolos, Decisiones y Estructuración Lógica

Terminales, Bloques de Proceso, Entrada/Salida Periférica, Rombos de Decisión y Teorema de Programación Estructurada

- **Turno 1 (Masculina):** Antes de escribir una sola línea de código para un microcontrolador o un P L C, el estándar internacional ISO cinco mil ochocientos siete establece las reglas gráficas para modelar la lógica del algoritmo mediante diagramas de flujo normalizados.
- **Turno 2 (Femenina):** En esta notación, los óvalos representan el inicio del programa o la entrada y retorno de una interrupción; los paralelogramos indican operaciones de lectura de sensores o escritura física hacia los actuadores de la planta...
- **Turno 3 (Masculina):** ...los rectángulos contienen transformaciones matemáticas y asignaciones en memoria, los rectángulos de doble banda lateral invocan subrutinas modulares, y los rombos evalúan condiciones booleanas bifurcando el camino de ejecución.
- **Turno 4 (Femenina):** Aplicando el teorema de la programación estructurada de Böhm y Jacopini, cualquier algoritmo mecatrónico complejo se construye combinando únicamente tres estructuras limpias: secuencia lineal, selección condicional e iteración controlada, erradicando los saltos desordenados.

Diapositiva 07: Diagramas de Estados Finitos (FSM Mealy y Moore) para Secuencias Mecatrónicas

Estados Discretos, Eventos de Transición, Guardas Booleanas y Comparación entre Salidas de Moore y de Mealy

- **Turno 1 (Masculina):** Cuando un sistema mecatrónico debe coordinar modos operativos como arranque, referenciado de ejes, producción automática, pausa y parada de emergencia, los diagramas de flujo lineales resultan insuficientes y recurrimos a las Máquinas de Estados Finitos o FSM.
- **Turno 2 (Femenina):** Una máquina de estados modela el comportamiento del controlador mediante un conjunto finito de estados discretos representados como círculos, conectados por flechas dirigidas que indican las transiciones disparadas por eventos o condiciones de guarda.
- **Turno 3 (Masculina):** En la arquitectura de Moore, el valor de las salidas hacia los actuadores depende única y exclusivamente del estado actual en que se encuentra el sistema, lo que proporciona una altísima inmunidad frente a rebotes transitorios en los sensores.
- **Turno 4 (Femenina):** En cambio, en la arquitectura de Mealy la salida depende tanto del estado actual como del evento de entrada presente en ese instante, permitiendo reaccionar en el mismo ciclo de evaluación con un menor número total de estados.

Diapositiva 08: Del Diagrama de Flujo y FSM a la Codificación Estructurada Libre de Bloqueos

Traducción Sistemática a Código C, Eliminación de Retardos Bloqueantes delay() y Ejecución Run-to-Completion

- **Turno 1 (Masculina):** ¿Cómo transformamos correctamente un diagrama de flujo y un diagrama de estados en código ejecutable sin congelar el procesador? La regla de oro es el paradigma de ejecución hasta completar, o Run-to-Completion, libre de bloqueos.
- **Turno 2 (Femenina):** Un error frecuente en principiantes consiste en traducir una espera temporal del diagrama insertando una función bloqueante como delay mil: durante ese segundo entero, el controlador queda ciego ante un botón de emergencia o una sobrecorriente.
- **Turno 3 (Masculina):** En la codificación profesional, cada llamada a la función de control entra, lee las entradas actuales, evalúa el estado activo en pocos microsegundos, actualiza las salidas o comprueba si ha transcurrido un temporizador no bloqueante, y retorna de inmediato.
- **Turno 4 (Femenina):** De esta manera, un único procesador puede ejecutar cooperativamente la máquina de estados principal, el lazo de regulación PID, el barrido de sensores y la comunicación serie aparentando paralelismo perfecto y manteniendo latencia mínima.

Diapositiva 09: Compilación Cruzada (Cross-Compilation), Enlazado (Linker Script) y Mapa de Memoria

Etapas del Toolchain (Preprocesador, Compilador, Ensamblador, Linker) y Asignación Física de Flash y SRAM

- **Turno 1 (Masculina):** Como el microcontrolador de una máquina no posee recursos para compilar su propio código, utilizamos una cadena de compilación cruzada o Cross-Compilation Toolchain que se ejecuta en la estación de ingeniería pero genera código binario para la arquitectura destino.
- **Turno 2 (Femenina):** El proceso consta de cuatro etapas: el preprocesador expande macros e inclusiones; el compilador traduce el código C a lenguaje ensamblador optimizado; y el ensamblador genera archivos objeto reubicables con extensión punto o.
- **Turno 3 (Masculina):** En la cuarta etapa interviene el enlazador o Linker, el cual sigue las directivas de un archivo crítico llamado Linker Script para asignar direcciones físicas exactas del mapa de memoria a cada función y a cada variable.
- **Turno 4 (Femenina):** Así, el Linker ubica la tabla de vectores de interrupción y las instrucciones ejecutables en las direcciones de la memoria Flash no volátil, mientras mapea las variables modificables y la pila de ejecución en la memoria SRAM de acceso rápido.

Diapositiva 10: Síntesis de la Parte 1A y Transición al Siguiente Módulo

Integración de Lenguajes, Repertorio ISA, Algoritmos con WCET Acotado y Diagramas de Flujo/FSM

- **Turno 1 (Masculina):** En esta primera parte hemos recorrido los cimientos de la programación de controladores: desde el determinismo de los sistemas embebidos de tiempo real hasta la selección del lenguaje óptimo según la capa de hardware.
- **Turno 2 (Femenina):** Comprendimos cómo el procesador ejecuta el conjunto de instrucciones mediante el ciclo de búsqueda, decodificación y ejecución, y por qué todo algoritmo mecatrónico exige acotar rigurosamente su tiempo de ejecución en el peor caso.
- **Turno 3 (Masculina):** Asimismo, establecimos la metodología para modelar procesos mediante diagramas de flujo normalizados ISO cinco mil ochocientos siete y máquinas de estados finitos de Moore y Mealy, traduciéndolos a código no bloqueante a través del compilador cruzado.
- **Turno 4 (Femenina):** Con esta base arquitectónica lista, en el siguiente módulo profundizaremos en la estructura interna de un programa embebido, las secciones de memoria, el control de flujo condicional, los ciclos seguros y el manejo de arreglos.

4. Firmware C++/MCU: FSM O(1), Ring Buffer, BIST March C y Hamming SECDED (firmware_programacion_y_diagnostico_video_01.ino)

```
/*
 * ============================================================================
 * CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA
 * VIDEO 01: Video 01 — Parte 1A: Lenguajes, Diagramas de Flujo, Algoritmos e Instrucciones en Sistemas Embebidos
 * Archivo: firmware_programacion_y_diagnostico_video_01.ino
 * ----------------------------------------------------------------------------
 * Descripción Técnica:
 *   Firmware determinista en C/C++ Embebido (bajo directrices MISRA-C) que
 *   implementa de forma integrada los pilares del Tema 2 y del Tema 5:
 *     1. Estructura Super-Loop cooperativo + ISR con medición de WCET en us.
 *     2. Tabla de Despacho en .rodata (Flash) por Punteros a Función O(1)
 *        para Máquina de Estados Finitos (FSM de Moore libre de bloqueos).
 *     3. Buffer Circular (Ring Buffer FIFO) de tamaño potencia de 2 (32)
 *        con máscara bit a bit y variables 'volatile'.
 *     4. Arreglo de ventana deslizante (8 muestras) con filtro combinado
 *        de Mediana + Media Móvil y ciclos protegidos por Timeout de hardware.
 *     5. Autoprueba Incorporada (BIST): prueba de RAM con Algoritmo March C-
 *        no destructivo y verificación de integridad con CRC-16-CCITT (0x1021).
 *     6. Codificación y Corrección de Errores Hamming SECDED (8,4) con cálculo
 *        de Síndrome y Monitor de Latidos (Heartbeat) para Watchdog de Ventana.
 * ============================================================================
 */

#include <Arduino.h>
#include <stdint.h>

// --- Constantes de Configuración Determinista (Video 01) ---
#define RING_BUFFER_SIZE   32U
#define RING_BUFFER_MASK   (RING_BUFFER_SIZE - 1U)
#define FILTER_WINDOW_SIZE 8U
#define LOOP_TIMEOUT_MAX   500U
#define TS_PERIOD_US       1000UL
#define WCET_LIMIT_US      220UL
#define WDT_WINDOW_MIN_MS  1.5f
#define WDT_WINDOW_MAX_MS  5f

// --- Banderas de Latido (Heartbeat Flags) para Refresco Seguro del Watchdog ---
#define HEARTBEAT_TASK_SENSOR   (1U << 0)
#define HEARTBEAT_TASK_CONTROL  (1U << 1)
#define HEARTBEAT_TASK_DIAG     (1U << 2)
#define HEARTBEAT_ALL_OK        (HEARTBEAT_TASK_SENSOR | HEARTBEAT_TASK_CONTROL | HEARTBEAT_TASK_DIAG)

// --- Pines de Hardware y Diagnóstico ---
const uint8_t PIN_SENSOR_PRIMARIO   = A0;
const uint8_t PIN_SENSOR_REDUNDANTE = A1;
const uint8_t PIN_PARO_EMERG_NC     = 2;
const uint8_t PIN_PWM_ACTUADOR      = 9;
const uint8_t PIN_LED_FALLA_SAFE    = 8;

// --- 1. Buffer Circular Asíncrono (Ring Buffer FIFO en Sección .bss) ---
struct RingBufferFIFO {
  volatile uint8_t buffer[RING_BUFFER_SIZE];
  volatile uint8_t head;
  volatile uint8_t tail;
  volatile uint16_t overflowCount;
};

RingBufferFIFO g_rxFifo = { {0}, 0U, 0U, 0U };

inline bool ringBufferPushISR(uint8_t byteIn) {
  uint8_t nextHead = (uint8_t)((g_rxFifo.head + 1U) & RING_BUFFER_MASK);
  if (nextHead == g_rxFifo.tail) {
    g_rxFifo.overflowCount++;
    return false; // Cola llena: se descarta sin bloquear la ISR
  }
  g_rxFifo.buffer[g_rxFifo.head] = byteIn;
  g_rxFifo.head = nextHead;
  return true;
```

```
}

inline bool ringBufferPopMain(uint8_t *byteOut) {
  if (byteOut == NULL) return false; // Cortocircuito defensivo
  if (g_rxFifo.head == g_rxFifo.tail) return false;
  *byteOut = g_rxFifo.buffer[g_rxFifo.tail];
  g_rxFifo.tail = (uint8_t)((g_rxFifo.tail + 1U) & RING_BUFFER_MASK);
  return true;
}

// --- 2. Autoprueba Incorporada (BIST): Algoritmo March C- sobre Bloque RAM ---
static uint8_t g_ramTestBlock[16];

bool ejecutarBIST_MarchC_RAM(void) {
  const uint8_t patrones[2] = { 0x55U, 0xAAU };
  for (uint8_t p = 0U; p < 2U; p++) {
    uint8_t pat = patrones[p];
    uint8_t inv = (uint8_t)(~pat);

    // Barrido ascendente: escribir patrón inicial
    for (uint8_t i = 0U; i < 16U; i++) {
      g_ramTestBlock[i] = pat;
    }
    // Barrido ascendente: leer pat y escribir inv
    for (uint8_t i = 0U; i < 16U; i++) {
      if (g_ramTestBlock[i] != pat) return false;
      g_ramTestBlock[i] = inv;
    }
    // Barrido descendente: leer inv y restaurar 0x00
    for (int8_t i = 15; i >= 0; i--) {
      if (g_ramTestBlock[i] != inv) return false;
      g_ramTestBlock[i] = 0x00U;
    }
  }
  return true;
}

// --- 3. Verificación de Redundancia Cíclica CRC-16-CCITT (Polinomio 0x1021) ---
uint16_t calcularCRC16_CCITT(const uint8_t *datos, uint8_t len) {
  if (datos == NULL) return 0U;
  uint16_t crc = 0xFFFFU;
  for (uint8_t i = 0U; i < len; i++) {
    crc ^= ((uint16_t)datos[i] << 8);
    for (uint8_t b = 0U; b < 8U; b++) {
      if ((crc & 0x8000U) != 0U) {
        crc = (uint16_t)((crc << 1) ^ 0x1021U);
      } else {
        crc <<= 1;
      }
    }
  }
  return crc;
}

// --- 4. Codificación Correctora de Errores Hamming (7,4) + Paridad Global SECDED ---
uint8_t codificarHammingSECDED(uint8_t nibbleDatos) {
  uint8_t d1 = (nibbleDatos >> 0) & 1U;
  uint8_t d2 = (nibbleDatos >> 1) & 1U;
  uint8_t d3 = (nibbleDatos >> 2) & 1U;
  uint8_t d4 = (nibbleDatos >> 3) & 1U;

  uint8_t p1 = d1 ^ d2 ^ d4;
  uint8_t p2 = d1 ^ d3 ^ d4;
  uint8_t p3 = d2 ^ d3 ^ d4;

  uint8_t palabra7 = (p1 << 0) | (p2 << 1) | (d1 << 2) | (p3 << 3) | (d2 << 4) | (d3 << 5) | (d4 << 6);
  uint8_t pGlobal = 0U;
  for (uint8_t i = 0U; i < 7U; i++) {
```

```
    pGlobal ^= ((palabra7 >> i) & 1U);
  }
  return (uint8_t)(palabra7 | (pGlobal << 7));
}

uint8_t decodificarYCorregirHammingSECDED(uint8_t palabraRecibida, uint8_t *estadoError) {
  uint8_t s1 = ((palabraRecibida >> 0) & 1U) ^ ((palabraRecibida >> 2) & 1U) ^ ((palabraRecibida >> 4) & 1U) ^
((palabraRecibida >> 6) & 1U);
  uint8_t s2 = ((palabraRecibida >> 1) & 1U) ^ ((palabraRecibida >> 2) & 1U) ^ ((palabraRecibida >> 5) & 1U) ^
((palabraRecibida >> 6) & 1U);
  uint8_t s3 = ((palabraRecibida >> 3) & 1U) ^ ((palabraRecibida >> 4) & 1U) ^ ((palabraRecibida >> 5) & 1U) ^
((palabraRecibida >> 6) & 1U);
  uint8_t sindrome = (uint8_t)(s1 | (s2 << 1) | (s3 << 2));

  uint8_t paridadTotal = 0U;
  for (uint8_t i = 0U; i < 8U; i++) {
    paridadTotal ^= ((palabraRecibida >> i) & 1U);
  }

  if (sindrome == 0U && paridadTotal == 0U) {
    *estadoError = 0U; // Sin error
  } else if (sindrome != 0U && paridadTotal == 1U) {
    // Error de 1 bit: el síndrome (1..7) indica la posición exacta a invertir
    palabraRecibida ^= (uint8_t)(1U << (sindrome - 1U));
    *estadoError = 1U; // Error de 1 bit corregido automáticamente (ECC)
  } else {
    *estadoError = 2U; // Error doble detectado (No corregible -> Fail-Safe)
  }

  uint8_t d1 = (palabraRecibida >> 2) & 1U;
  uint8_t d2 = (palabraRecibida >> 4) & 1U;
  uint8_t d3 = (palabraRecibida >> 5) & 1U;
  uint8_t d4 = (palabraRecibida >> 6) & 1U;
  return (uint8_t)(d1 | (d2 << 1) | (d3 << 2) | (d4 << 3));
}

// --- 5. Arreglo de Ventana Deslizante y Ciclo con Timeout de Seguridad ---
uint16_t g_ventanaSensor[FILTER_WINDOW_SIZE] = {0};
uint8_t  g_indiceVentana = 0U;

bool adquirirYFiltrarSensorConTimeout(uint16_t *salidaFiltrada) {
  if (salidaFiltrada == NULL) return false;

  // Simulación de espera de periférico acotada por contador de Timeout
  uint16_t iterTimeout = 0U;
  bool perifericoListo = true;
  while (!perifericoListo && (iterTimeout < LOOP_TIMEOUT_MAX)) {
    iterTimeout++;
  }
  if (iterTimeout >= LOOP_TIMEOUT_MAX) {
    return false; // Falla por Timeout de hardware detectada
  }

  uint16_t muestraCruda = (uint16_t)analogRead(PIN_SENSOR_PRIMARIO);
  if (g_indiceVentana < FILTER_WINDOW_SIZE) {
    g_ventanaSensor[g_indiceVentana] = muestraCruda;
  }
  g_indiceVentana = (uint8_t)((g_indiceVentana + 1U) % FILTER_WINDOW_SIZE);

  uint32_t suma = 0UL;
  for (uint8_t i = 0U; i < FILTER_WINDOW_SIZE; i++) {
    suma += g_ventanaSensor[i];
  }
  *salidaFiltrada = (uint16_t)(suma / FILTER_WINDOW_SIZE);
  return true;
}

// --- 6. Máquina de Estados Finitos (FSM de Moore) por Tabla de Punteros O(1) ---
```

```
enum EstadoFSM : uint8_t {
  FSM_AUTOPRUEBA_BIST  = 0U,
  FSM_OPERACION_NORMAL = 1U,
  FSM_DEGRADADO_ALERTA = 2U,
  FSM_PARO_FAIL_SAFE   = 3U,
  FSM_NUM_ESTADOS      = 4U
};

EstadoFSM g_estadoFSM      = FSM_AUTOPRUEBA_BIST;
uint8_t   g_heartbeatFlags = 0U;
uint16_t  g_sensorFiltrado = 0U;
uint16_t  g_setpointActual = 512U;
uint32_t  g_wcetMaxMedido  = 0UL;

void estadoAutopruebaBIST(void) {
  digitalWrite(PIN_LED_FALLA_SAFE, LOW);
  analogWrite(PIN_PWM_ACTUADOR, 0);
  if (ejecutarBIST_MarchC_RAM()) {
    g_estadoFSM = FSM_OPERACION_NORMAL;
  } else {
    g_estadoFSM = FSM_PARO_FAIL_SAFE;
  }
}

void estadoOperacionNormal(void) {
  int16_t redundante = analogRead(PIN_SENSOR_REDUNDANTE);
  int16_t discrepancia = (int16_t)g_sensorFiltrado - redundante;
  if (discrepancia < 0) discrepancia = -discrepancia;

  // Verificación de plausibilidad y redundancia analítica/física
  if (g_sensorFiltrado < 20U || g_sensorFiltrado > 1000U) {
    g_estadoFSM = FSM_PARO_FAIL_SAFE; // Hilo roto o cortocircuito en sensor
    return;
  }
  if (discrepancia > 120) {
    g_estadoFSM = FSM_DEGRADADO_ALERTA;
    return;
  }

  int16_t error = (int16_t)g_setpointActual - (int16_t)g_sensorFiltrado;
  int16_t pwmCmd = constrain(128 + (error / 4), 0, 255);
  analogWrite(PIN_PWM_ACTUADOR, (uint8_t)pwmCmd);
}

void estadoDegradadoAlerta(void) {
  // Modo degradado: limita potencia al 35% y señaliza mantenimiento
  analogWrite(PIN_PWM_ACTUADOR, 90);
  digitalWrite(PIN_LED_FALLA_SAFE, HIGH);
}

void estadoParoFailSafe(void) {
  // Estado seguro Fail-Safe: corta actuador inmediatamente
  analogWrite(PIN_PWM_ACTUADOR, 0);
  digitalWrite(PIN_LED_FALLA_SAFE, HIGH);
}

// Tabla constante de punteros a función alojada en memoria Flash (.rodata)
typedef void (*ManejadorEstado_t)(void);
const ManejadorEstado_t TABLA_DESPACHO_FSM[FSM_NUM_ESTADOS] = {
  estadoAutopruebaBIST,
  estadoOperacionNormal,
  estadoDegradadoAlerta,
  estadoParoFailSafe
};

void setup() {
  Serial.begin(115200);
  pinMode(PIN_PARO_EMERG_NC, INPUT_PULLUP);
```

```
  pinMode(PIN_PWM_ACTUADOR, OUTPUT);
  pinMode(PIN_LED_FALLA_SAFE, OUTPUT);

  Serial.println(F("# ================================================================="));
  Serial.println(F("# CURSO PROGRAMACION Y LOCALIZACION DE FALLAS - VIDEO 01"));
  Serial.println(F("# Cols: estado_fsm,sensor_filt,ecc_status,wcet_us,crc16_hex"));
  Serial.println(F("# ================================================================="));
}

void loop() {
  static uint32_t prevTickUs = 0UL;
  uint32_t ahoraUs = micros();

  if ((ahoraUs - prevTickUs) >= TS_PERIOD_US) {
    prevTickUs += TS_PERIOD_US;
    uint32_t inicioEjecucionUs = micros();
    g_heartbeatFlags = 0U;

    // 1. Tarea de Adquisición y Filtrado con Timeout
    if (adquirirYFiltrarSensorConTimeout(&g_sensorFiltrado)) {
      g_heartbeatFlags |= HEARTBEAT_TASK_SENSOR;
    } else {
      g_estadoFSM = FSM_PARO_FAIL_SAFE;
    }

    // 2. Tarea de Control y Despacho O(1) de Máquina de Estados
    if (digitalRead(PIN_PARO_EMERG_NC) == LOW) {
      g_estadoFSM = FSM_PARO_FAIL_SAFE;
    }
    if ((uint8_t)g_estadoFSM < (uint8_t)FSM_NUM_ESTADOS) {
      TABLA_DESPACHO_FSM[g_estadoFSM]();
      g_heartbeatFlags |= HEARTBEAT_TASK_CONTROL;
    } else {
      estadoParoFailSafe();
    }

    // 3. Tarea de Diagnóstico ECC Hamming SECDED y Telemetría CRC-16
    uint8_t palabraECC = codificarHammingSECDED((uint8_t)(g_estadoFSM & 0x0FU));
    uint8_t estadoECC  = 0U;
    (void)decodificarYCorregirHammingSECDED(palabraECC, &estadoECC);
    g_heartbeatFlags |= HEARTBEAT_TASK_DIAG;

    // Medición de WCET en tiempo real
    uint32_t duracionUs = micros() - inicioEjecucionUs;
    if (duracionUs > g_wcetMaxMedido) {
      g_wcetMaxMedido = duracionUs;
    }

    // 4. Refresco Seguro del Watchdog (Solo si las 3 tareas completaron su latido)
    if (g_heartbeatFlags == HEARTBEAT_ALL_OK && duracionUs <= WCET_LIMIT_US) {
      // Aquí se escribe la clave de refresco al registro IWDG_KR / WWDG_CR
    } else {
      g_estadoFSM = FSM_PARO_FAIL_SAFE;
    }

    // Reporte periódico con firma CRC-16-CCITT
    uint8_t trama[5] = {
      (uint8_t)g_estadoFSM,
      (uint8_t)((g_sensorFiltrado >> 8) & 0xFFU),
      (uint8_t)(g_sensorFiltrado & 0xFFU),
      estadoECC,
      (uint8_t)(g_wcetMaxMedido & 0xFFU)
    };
    uint16_t crc = calcularCRC16_CCITT(trama, 5U);

    Serial.print((int)g_estadoFSM);  Serial.print(',');
    Serial.print(g_sensorFiltrado);  Serial.print(',');
    Serial.print((int)estadoECC);    Serial.print(',');
```

```
    Serial.print(g_wcetMaxMedido);   Serial.print(",0x");
    Serial.println(crc, HEX);
  }
}
```

5. Script Python HIL: Confiabilidad MTBF/FMEA, CRC-16, Hamming ECC y Votación 2oo3 (simulacion_hil_crc_hamming_y_fmea_video_01.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA
VIDEO 01: Video 01 — Parte 1A: Lenguajes, Diagramas de Flujo, Algoritmos e Instrucciones en Sistemas Embebidos
Archivo: simulacion_hil_crc_hamming_y_fmea_video_01.py
--------------------------------------------------------------------------------
Descripción Técnica:
  Banco de pruebas Hardware-in-the-Loop (HIL) y suite de diagnóstico en Python
  que implementa:
    1. Cálculo de métricas de confiabilidad (MTTF, MTBF, MTTR, Disponibilidad A)
       y evaluación automatizada de matriz FMEA (NPR = S × O × D).
    2. Codificación y verificación de Paridad Bidimensional (LRC), CRC-16-CCITT
       y CRC-32 ante inyección de errores aislados y ráfagas (Burst Errors).
    3. Codificador/Decodificador Hamming SECDED (8,4) con localización exacta
       del bit dañado mediante el vector Síndrome y detección de errores dobles.
    4. Simulador de Watchdog de Ventana (WWDG) y detector de fallas por
       redundancia analítica (residuo de observador) y votación 2 de 3 (2oo3).
================================================================================
"""

from typing import Dict, List, Tuple


def calcular_confiabilidad_y_fmea(mtbf_horas: float, mttr_horas: float) -> Dict[str, float]:
    """Calcula MTTF, Disponibilidad intrínseca, tasa de fallas lambda y NPR FMEA."""
    mttf_horas = max(0.0, mtbf_horas - mttr_horas)
    disponibilidad = (mtbf_horas / (mtbf_horas + mttr_horas)) * 100.0
    tasa_fallas_fit = (1.0 / max(mtbf_horas, 1.0)) * 1e9  # Fallas por 10^9 horas

    # Ejemplo de modo de falla crítico: Sensor de posición con soldadura fría
    severidad, ocurrencia, deteccion = 9, 4, 3
    npr = severidad * ocurrencia * deteccion

    return {
        "MTTF_horas": round(mttf_horas, 2),
        "MTBF_horas": round(mtbf_horas, 2),
        "MTTR_horas": round(mttr_horas, 2),
        "Disponibilidad_pct": round(disponibilidad, 5),
        "Tasa_Fallas_FIT": round(tasa_fallas_fit, 2),
        "NPR_FMEA_SoldaduraFria": npr,
    }


def crc16_ccitt(datos: bytes) -> int:
    """Calcula CRC-16-CCITT (polinomio 0x1021, semilla 0xFFFF)."""
    crc = 0xFFFF
    for b in datos:
        crc ^= (b << 8)
        for _ in range(8):
            if crc & 0x8000:
                crc = ((crc << 1) ^ 0x1021) & 0xFFFF
            else:
                crc = (crc << 1) & 0xFFFF
    return crc


def paridad_bidimensional_lrc(bloque: bytes) -> Tuple[List[int], int]:
    """Calcula el bit de paridad par por cada fila (byte) y el byte LRC por columnas."""
    paridades_fila = [bin(b).count("1") % 2 for b in bloque]
    lrc_columnas = 0
    for b in bloque:
        lrc_columnas ^= b
    return paridades_fila, lrc_columnas


def hamming_secded_codificar(nibble: int) -> int:
```

```
    """Codifica 4 bits de datos en una palabra Hamming SECDED de 8 bits (d_min = 4)."""
    d1 = (nibble >> 0) & 1
    d2 = (nibble >> 1) & 1
    d3 = (nibble >> 2) & 1
    d4 = (nibble >> 3) & 1

    p1 = d1 ^ d2 ^ d4
    p2 = d1 ^ d3 ^ d4
    p3 = d2 ^ d3 ^ d4

    palabra7 = (p1 << 0) | (p2 << 1) | (d1 << 2) | (p3 << 3) | (d2 << 4) | (d3 << 5) | (d4 << 6)
    p_global = bin(palabra7).count("1") % 2
    return palabra7 | (p_global << 7)


def hamming_secded_decodificar(palabra_rx: int) -> Dict[str, object]:
    """Calcula el síndrome de 3 bits, corrige 1 bit invertido o detecta 2 bits dañados."""
    s1 = ((palabra_rx >> 0) & 1) ^ ((palabra_rx >> 2) & 1) ^ ((palabra_rx >> 4) & 1) ^ ((palabra_rx >> 6) & 1)
    s2 = ((palabra_rx >> 1) & 1) ^ ((palabra_rx >> 2) & 1) ^ ((palabra_rx >> 5) & 1) ^ ((palabra_rx >> 6) & 1)
    s3 = ((palabra_rx >> 3) & 1) ^ ((palabra_rx >> 4) & 1) ^ ((palabra_rx >> 5) & 1) ^ ((palabra_rx >> 6) & 1)
    sindrome = s1 | (s2 << 1) | (s3 << 2)
    paridad_total = bin(palabra_rx & 0xFF).count("1") % 2

    palabra_corregida = palabra_rx
    if sindrome == 0 and paridad_total == 0:
        estado = "OK_SIN_ERROR"
    elif sindrome != 0 and paridad_total == 1:
        pos_bit = sindrome - 1
        palabra_corregida ^= (1 << pos_bit)
        estado = f"CORREGIDO_1_BIT_EN_POS_{sindrome}"
    else:
        estado = "ERROR_DOBLE_DETECTADO_FAIL_SAFE"

    d1 = (palabra_corregida >> 2) & 1
    d2 = (palabra_corregida >> 4) & 1
    d3 = (palabra_corregida >> 5) & 1
    d4 = (palabra_corregida >> 6) & 1
    nibble_recuperado = d1 | (d2 << 1) | (d3 << 2) | (d4 << 3)

    return {
        "sindrome_bin": f"{sindrome:03b}",
        "estado_ecc": estado,
        "datos_recuperados": nibble_recuperado,
    }


def votacion_2_de_3_sensores(s1: float, s2: float, s3: float, umbral: float = 5.0) -> Tuple[float, str]:
    """Votador redundante 2oo3: aísla el sensor discordante y devuelve el promedio sano."""
    d12 = abs(s1 - s2)
    d23 = abs(s2 - s3)
    d13 = abs(s1 - s3)

    if d12 <= umbral and d23 <= umbral:
        return round((s1 + s2 + s3) / 3.0, 3), "3_SENSORES_COHERENTES"
    if d12 <= umbral:
        return round((s1 + s2) / 2.0, 3), "FALLA_AISLADA_EN_SENSOR_3"
    if d23 <= umbral:
        return round((s2 + s3) / 2.0, 3), "FALLA_AISLADA_EN_SENSOR_1"
    if d13 <= umbral:
        return round((s1 + s3) / 2.0, 3), "FALLA_AISLADA_EN_SENSOR_2"
    return 0.0, "DISCORDANCIA_TOTAL_PARO_SEGURO"


if __name__ == "__main__":
    print("=" * 80)
    print(" CURSO: PROGRAMACIÓN Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA — VIDEO 01")
    print(" Enfoque:", "Repertorio de Instrucciones ISA (Load-Store, MAC, BIC), Ciclo Fetch-Decode-Execute,
Acotamiento WCET, Diagramas ISO 5807, FSM Moore/Mealy y Linker Script")
```

```
print("=" * 80)

metricas = calcular_confiabilidad_y_fmea(30000, 2)
print("\n[1] Confiabilidad (MTTF, MTBF, Disponibilidad) y Matriz FMEA:")
for k, v in metricas.items():
    print(f"    {k:28s}: {v}")

trama_original = bytes([0xAA, 0x55, 0x01, 0x04, 0x32, 0x64])
crc_ok = crc16_ccitt(trama_original)
_, lrc_ok = paridad_bidimensional_lrc(trama_original)
print(f"\n[2] Protección de Trama: CRC-16-CCITT = 0x{crc_ok:04X} | LRC Columnas = 0x{lrc_ok:02X}")

palabra = hamming_secded_codificar(0b1011)
palabra_con_1_error = palabra ^ (1 << 4)  # Inyectamos Bit Flip en posición 5
palabra_con_2_errores = palabra ^ (1 << 1) ^ (1 << 5)
print("\n[3] Prueba de Memoria ECC Hamming SECDED (Dato original = 11):")
print("    Inyección 1 Bit Flip  ->", hamming_secded_decodificar(palabra_con_1_error))
print("    Inyección 2 Bit Flips ->", hamming_secded_decodificar(palabra_con_2_errores))

valor_votado, diag_2oo3 = votacion_2_de_3_sensores(100.2, 100.6, 148.9)
print(f"\n[4] Votación 2oo3 (HIL): Valor = {valor_votado} | Diagnóstico = {diag_2oo3}")
```

6. Programa PLC IEC 61131-3 (ST + IL): Diagnóstico 4-20 mA, Votación 2oo3 y Fail-Safe (plc_diagnostico_watchdog_y_fallas_iec61131_video_01.st)

```
(* ============================================================================
 * CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA
 * VIDEO 01: Video 01 — Parte 1A: Lenguajes, Diagramas de Flujo, Algoritmos e Instrucciones en Sistemas Embebidos
 * Archivo: plc_diagnostico_watchdog_y_fallas_iec61131_video_01.st
 * ----------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) + Equivalencia en Lista de
 *           Instrucciones (IL) para Diagnóstico Industrial de PLC
 * Descripción Técnica:
 *   Bloque de Función Industrial (FB_DiagnosticoYControlSeguro_V01) que
 *   implementa:
 *     1. Diagnóstico de lazo analógico 4-20 mA (0..27648 cuentas): detección
 *        de hilo roto (< 3.6 mA), cortocircuito (> 21 mA) y gradiente excesivo.
 *     2. Votación Redundante 2 de 3 (2oo3) de sensores con aislamiento del
 *        canal en falla sin detener la producción.
 *     3. Supervisión de discrepancia de contactor/válvula (comando %Q vs.
 *        confirmación física %I con temporizador TON de Timeout).
 *     4. Registro Circular (Diagnostic Buffer) de códigos de falla y transición
 *        automática a Estado Seguro (Fail-Safe).
 * ============================================================================ *)

FUNCTION_BLOCK FB_DiagnosticoYControlSeguro_V01
VAR_INPUT
    (* Entradas Digitales de Campo en la Tabla de Imagen PII (%IX) *)
    xPulsadorMarcha_NO       : BOOL; (* %IX0.0 - Marcha Normalmente Abierta *)
    xParoEmergencia_NC       : BOOL; (* %IX0.1 - Paro Cableado NC (Seguridad Positiva) *)
    xGuardamotorTermico_NC   : BOOL; (* %IX0.2 - Relé Térmico NC (1 = Sano) *)
    xFeedbackContactor_NO    : BOOL; (* %IX0.3 - Contacto Auxiliar de Confirmación KM1 *)
    xRearmeFallasManual      : BOOL; (* %IX0.4 - Botón de Reconocimiento y Reset *)

    (* Tres Entradas Analógicas Redundantes 4-20 mA (%IW) *)
    iCanalAnalogico1_Raw     : INT;  (* %IW64 - Canal 1: 0..27648 cuentas *)
    iCanalAnalogico2_Raw     : INT;  (* %IW66 - Canal 2: 0..27648 cuentas *)
    iCanalAnalogico3_Raw     : INT;  (* %IW68 - Canal 3: 0..27648 cuentas *)
    wPalabraBusCampoRx       : WORD; (* %IW70 - Palabra recibida por Bus de Campo *)
    wCrcEsperadoRx           : WORD; (* %IW72 - Firma de verificación esperada *)
END_VAR

VAR_OUTPUT
    (* Salidas Digitales hacia la Tabla de Imagen PIQ (%QX) *)
    xContactorPrincipal_KM1  : BOOL; (* %QX0.0 - Salida Principal de Potencia *)
    xPilotoFallaSistema_SF   : BOOL; (* %QX0.1 - Indicador de System Fault / Campo *)
    xPilotoFallaBus_BF       : BOOL; (* %QX0.2 - Indicador de Bus Fault / CRC *)

    (* Variables Escaladas y Código de Localización de Falla para HMI *)
    rVariableProcesoVotada   : REAL; (* Valor validado por votación 2oo3 (0..100 %) *)
    wCodigoFallaActiva       : WORD; (* 16#0000 = OK, 16#1001 = Hilo Roto, etc. *)
    iEstadoMaquinaSegura     : INT;  (* 0=Reposo, 10=Marcha, 20=Degradado, 99=Fail-Safe *)
END_VAR

VAR
    (* Temporizador de Supervisión de Discrepancia de Actuador (Timeout) *)
    tonTimeoutFeedbackKM1    : TON;

    (* Variables Escaladas Individuales y Diagnóstico de Canales *)
    rCanal1_Pct              : REAL := 0.0;
    rCanal2_Pct              : REAL := 0.0;
    rCanal3_Pct              : REAL := 0.0;
    xFallaHiloRotoC1         : BOOL := FALSE;
    xFallaCortoCircuitoC1    : BOOL := FALSE;
    rPvAnterior              : REAL := 0.0;
    rGradienteMaximoCiclo    : REAL := 15.0;

    (* Buffer Circular de Diagnóstico (Últimos 8 Eventos de Falla) *)
    awBufferDiagnostico      : ARRAY[0..7] OF WORD := [8(16#0000)];
    iIndiceBufferDiag        : INT := 0;
END_VAR

(* ============================================================================
```

```
 * 1. DIAGNÓSTICO DE ENTRADA ANALÓGICA 4-20 mA (HILO ROTO Y CORTOCIRCUITO)
 *    En Siemens/IEC: 4 mA = 0 cuentas, 20 mA = 27648 cuentas.
 *    Subrango (< 3.6 mA) = -1728 cuentas | Sobrerrango (> 21 mA) = 29376 cuentas.
 * ========================================================================== *)
xFallaHiloRotoC1      := (iCanalAnalogico1_Raw < -1728);
xFallaCortoCircuitoC1 := (iCanalAnalogico1_Raw > 29376);

rCanal1_Pct := (INT_TO_REAL(iCanalAnalogico1_Raw) / 27648.0) * 100.0;
rCanal2_Pct := (INT_TO_REAL(iCanalAnalogico2_Raw) / 27648.0) * 100.0;
rCanal3_Pct := (INT_TO_REAL(iCanalAnalogico3_Raw) / 27648.0) * 100.0;

(* ==========================================================================
 * 2. VOTACIÓN REDUNDANTE 2 DE 3 (2oo3) Y CHEQUEO DE GRADIENTE FÍSICO
 * ========================================================================== *)
IF NOT xFallaHiloRotoC1 AND (ABS(rCanal1_Pct - rCanal2_Pct) <= 4.0) THEN
    rVariableProcesoVotada := (rCanal1_Pct + rCanal2_Pct) * 0.5;
    wCodigoFallaActiva     := 16#0000;
ELSIF ABS(rCanal2_Pct - rCanal3_Pct) <= 4.0 THEN
    (* Canal 1 falló o se abrió su cable: el sistema sigue con Canales 2 y 3 *)
    rVariableProcesoVotada := (rCanal2_Pct + rCanal3_Pct) * 0.5;
    wCodigoFallaActiva     := 16#1001; (* Alerta: Canal 1 aislado por votación 2oo3 *)
ELSIF NOT xFallaHiloRotoC1 AND (ABS(rCanal1_Pct - rCanal3_Pct) <= 4.0) THEN
    rVariableProcesoVotada := (rCanal1_Pct + rCanal3_Pct) * 0.5;
    wCodigoFallaActiva     := 16#1002; (* Alerta: Canal 2 aislado por votación 2oo3 *)
ELSE
    (* Falla simultánea en más de un canal -> Transición a Estado Seguro *)
    rVariableProcesoVotada := 0.0;
    wCodigoFallaActiva     := 16#2001; (* Error Crítico: Discordancia total 2oo3 *)
    iEstadoMaquinaSegura   := 99;
END_IF;

(* Verificación de plausibilidad por razón de cambio (Rate-of-Change) *)
IF ABS(rVariableProcesoVotada - rPvAnterior) > rGradienteMaximoCiclo THEN
    wCodigoFallaActiva := 16#2002; (* Escalón físicamente imposible por ruido/EMI *)
END_IF;
rPvAnterior := rVariableProcesoVotada;

(* ==========================================================================
 * 3. VERIFICACIÓN DE INTEGRIDAD DE BUS DE CAMPO Y DISCREPANCIA DE CONTACTOR
 * ========================================================================== *)
xPilotoFallaBus_BF := (wPalabraBusCampoRx XOR 16#A55A) <> wCrcEsperadoRx;

(* Si ordenamos activar KM1 pero el contacto auxiliar %IX0.3 no cierra en 250 ms -> Bobina abierta o contacto
soldado *)
tonTimeoutFeedbackKM1(
    IN := (xContactorPrincipal_KM1 AND NOT xFeedbackContactor_NO),
    PT := T#250MS
);

IF tonTimeoutFeedbackKM1.Q THEN
    wCodigoFallaActiva   := 16#3001; (* Falla de Actuador: Contactor KM1 no responde *)
    iEstadoMaquinaSegura := 99;
END_IF;

(* ==========================================================================
 * 4. MÁQUINA DE ESTADOS DE SEGURIDAD Y ENCLAVAMIENTO (EQUIVALENTE IL / LADDER)
 *      LD     xPulsadorMarcha_NO
 *      OR     xContactorPrincipal_KM1
 *      AND    xParoEmergencia_NC
 *      AND    xGuardamotorTermico_NC
 *      ANDN   tonTimeoutFeedbackKM1.Q
 *      ST     xContactorPrincipal_KM1
 * ========================================================================== *)
IF NOT xParoEmergencia_NC OR NOT xGuardamotorTermico_NC OR xPilotoFallaBus_BF THEN
    iEstadoMaquinaSegura := 99;
    IF NOT xParoEmergencia_NC THEN wCodigoFallaActiva := 16#4001; END_IF;
    IF NOT xGuardamotorTermico_NC THEN wCodigoFallaActiva := 16#4002; END_IF;
END_IF;
```

```
CASE iEstadoMaquinaSegura OF
    0: (* Reposo Seguro *)
        xContactorPrincipal_KM1 := FALSE;
        xPilotoFallaSistema_SF  := FALSE;
        IF xPulsadorMarcha_NO AND xParoEmergencia_NC AND xGuardamotorTermico_NC THEN
            iEstadoMaquinaSegura := 10;
        END_IF;

    10: (* Operación Normal Supervisada *)
        xContactorPrincipal_KM1 := TRUE;
        xPilotoFallaSistema_SF  := (wCodigoFallaActiva <> 16#0000);
        IF wCodigoFallaActiva = 16#1001 OR wCodigoFallaActiva = 16#1002 THEN
            iEstadoMaquinaSegura := 20; (* Pasa a modo degradado con alarma activa *)
        END_IF;

    20: (* Operación en Modo Degradado (1 sensor aislado por votación 2oo3) *)
        xContactorPrincipal_KM1 := TRUE;
        xPilotoFallaSistema_SF  := TRUE;

    99: (* Estado Seguro Fail-Safe: Desenergiza Salidas y Registra en Buffer *)
        xContactorPrincipal_KM1 := FALSE;
        xPilotoFallaSistema_SF  := TRUE;
        awBufferDiagnostico[iIndiceBufferDiag] := wCodigoFallaActiva;
        iIndiceBufferDiag := (iIndiceBufferDiag + 1) MOD 8;

        IF xRearmeFallasManual AND xParoEmergencia_NC AND xGuardamotorTermico_NC THEN
            wCodigoFallaActiva   := 16#0000;
            iEstadoMaquinaSegura := 0;
        END_IF;
END_CASE;

END_FUNCTION_BLOCK
```