CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA

Video 00 — Introducción al Curso de Programación y Localización de Fallas en Mecatrónica — Arquitectura de las 2 Grandes Partes: Programación de Controladores (Tema 2) y Localización de Fallas (Tema 5)

Parte del Curso: Introducción General al Curso de Programación y Localización de Fallas en Mecatrónica
Enfoque de Programación y Diagnóstico: Arquitectura Integral: Algoritmos Deterministas (WCET, FSM, Ring Buffer) + Diagnóstico Industrial (Watchdog, CRC, Hamming, BIST, HIL y PLC)

```
WCET_total < Ts | A = MTBF / (MTBF + MTTR) | d_min >= 2t + 1 (Hamming SECDED) | CRC-16 G(x) = x^16 + x^12 + x^5 + 1
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Visión unificada entre el Tema 2 (Lenguajes y diagramas en programación de controladores) y el Tema 5 (Localización de fallas en sistemas mecatrónicos).
- Parte 1A y 1B: Sistemas embebidos de tiempo real duro, jerarquía de lenguajes (Ensamblador, C/C++ MISRA-C, IEC 61131-3, Python), repertorio ISA, ciclo Fetch-Decode-Execute, acotamiento WCET, diagramas de flujo ISO 5807, máquinas de estados FSM (Moore y Mealy), secciones de memoria (.text, .rodata, .data, .bss, Stack), tablas de punteros a función, ciclos con timeout, arreglos de filtrado y Ring Buffers FIFO.
- Parte 2A: Métricas de confiabilidad (MTTF, MTBF, MTTR), redundancia física (2oo3) y analítica por observadores, análisis FMEA (NPR) y FTA, Watchdog Timer independiente y de ventana refrescado por banderas de latido, paridad simple/LRC, Checksum, CRC-8/16/32 y código corrector Hamming SECDED (ECC).
- Parte 2B: Física de fallas de hardware (soldadura fría, ESR de condensadores, ESD, EMI, lazos de tierra y Latch-Up), diagnóstico de microprocesadores (alimentación, reset y reloj), sonda lógica, analizador lógico y osciloscopio MSO, autoprueba BIST (CRC en Flash y algoritmo March C en RAM), depuración JTAG/SWD, validación MIL/SIL/PIL/HIL y localización de fallas en PLC.

2. Parámetros Nominales de Tiempo Real, Memoria y Confiabilidad (Video 00)

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]`	350 µs	Tiempo máximo permitido para completar todas las tareas sin violar el tiempo real
Ventana del Temporizador Vigilante	`[T_min, T_max]`	[2, 8] ms	Ventana válida de refresco del Windowed Watchdog por banderas de latido
Tamaño del Ring Buffer y Ventana	`N_FIFO / M_win`	FIFO = 64 B, Win = 8	Cola circular potencia de 2 para ISR y arreglo de filtrado de media móvil
Tiempo Medio Entre Fallas / Reparación	`MTBF / MTTR`	25000 h / 2.5 h	Indicadores de confiabilidad y mantenibilidad para seguridad funcional

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Bienvenidos al Curso de Programación y Localización de Fallas en Mecatrónica

Visión Integral desde el Diseño de Algoritmos y Sistemas Embebidos hasta el Diagnóstico Industrial

- **Turno 1 (Masculina):** Bienvenidos al Mini Curso de Programación de Controladores y Localización de Fallas en Mecatrónica en cursosmecatronica punto com.
- **Turno 2 (Femenina):** Uniremos el diseño de algoritmos en tiempo real con el diagnóstico de sistemas.
- **Turno 3 (Masculina):** Veremos lenguajes embebidos, diagramas de flujo, máquinas de estados y uso de memoria.
- **Turno 4 (Femenina):** Y aplicaremos detección de errores, prueba de hardware, emulación en circuito y simulación.

Diapositiva 02: Mapa Estructural del Curso: Programación Determinista y Diagnóstico de Fallas

Parte 1 (Lenguajes, Diagramas, Ciclos y Arreglos) y Parte 2 (Watchdog, Paridad/CRC, Hardware, HIL y PLC)

- **Turno 1 (Masculina):** En la Parte uno veremos instrucciones del procesador, control de flujo, ciclos y arreglos.
- **Turno 2 (Femenina):** En la Parte dos estudiaremos reloj vigilante, paridad, códigos cíclicos y fallas físicas.
- **Turno 3 (Masculina):** Analizaremos tarjetas con microprocesador y controladores lógicos mediante pruebas con hardware en lazo.
- **Turno 4 (Femenina):** Comencemos en el siguiente módulo de cursosmecatronica punto com con lenguajes, diagramas de flujo e instrucciones embebidas.

4. Firmware C++/MCU: FSM O(1), Ring Buffer, BIST March C y Hamming SECDED (firmware_programacion_y_diagnostico_video_00.ino)

```
/*
 * ============================================================================
 * CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA
 * VIDEO 00: Video 00 — Introducción al Curso de Programación y Localización de Fallas en Mecatrónica
 * Archivo: firmware_programacion_y_diagnostico_video_00.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 (64)
 *        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 00) ---
#define RING_BUFFER_SIZE   64U
#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      350UL
#define WDT_WINDOW_MIN_MS  2f
#define WDT_WINDOW_MAX_MS  8f

// --- 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 00"));
  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_00.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA
VIDEO 00: Video 00 — Introducción al Curso de Programación y Localización de Fallas en Mecatrónica
Archivo: simulacion_hil_crc_hamming_y_fmea_video_00.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 00")
    print(" Enfoque:", "Arquitectura Integral: Algoritmos Deterministas (WCET, FSM, Ring Buffer) + Diagnóstico
Industrial (Watchdog, CRC, Hamming, BIST, HIL y PLC)")
```

```
print("=" * 80)

metricas = calcular_confiabilidad_y_fmea(25000, 2.5)
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_00.st)

```
(* ================================================================================
 * CURSO: PROGRAMACIÓN DE CONTROLADORES Y LOCALIZACIÓN DE FALLAS EN MECATRÓNICA
 * VIDEO 00: Video 00 — Introducción al Curso de Programación y Localización de Fallas en Mecatrónica
 * Archivo: plc_diagnostico_watchdog_y_fallas_iec61131_video_00.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_V00) 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_V00
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
```