CURSO COMPLETO: SISTEMAS DE COMUNICACIÓN Y REDES INALÁMBRICAS EN MECATRÓNICA

Video 05 — Resumen Ejecutivo y Cierre del Curso de Sistemas de Comunicación y Redes Inalámbricas — Síntesis Integradora de Comunicaciones Digitales, Arquitecturas de Control, Modelo OSI, Buses Serie/Paralelo y Redes Inalámbricas

Parte del Curso: Resumen Ejecutivo y Cierre Integrador del Curso de Comunicaciones y Redes Inalámbricas
Sistemas / Protocolos Estudiados: Arquitectura Híbrida Cableada + Inalámbrica: Bus de Campo RS-485/CAN/EtherCAT + Pasarela IIoT Wi-Fi 6 / BLE / LoRaWAN / 5G URLLC

```
Integración Total: [Sensor BLE/ZigBee/LoRa] --> [Gateway SBC/PLC MQTT/OPC UA] <--> [Bus Determinista RS-485/CAN/EtherCAT]
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Síntesis de la Parte 1A: diseño de enlaces digitales respaldado por los teoremas de Nyquist y Shannon-Hartley, selección de arquitecturas de control distribuido (DCS) e integración en los 5 niveles de la pirámide de automatización CIM con métodos deterministas de acceso al medio.
- Síntesis de la Parte 1B: aplicación del modelo OSI y verificación polinómica CRC-16 en interfases serie de placa (UART, SPI, I2C), buses diferenciales de campo (RS-485 Modbus RTU, CANopen), Ethernet Industrial en tiempo real (PROFINET IRT, EtherCAT) e instrumentación paralela GPIB / IEEE-488.
- Síntesis de la Parte 2A: dimensionamiento de enlaces de radiofrecuencia mediante la ecuación de Friis y el presupuesto de enlace en dBm, desplegando redes locales de alto caudal con Wi-Fi 6 (OFDMA/TWT), Wi-Fi HaLow con MQTT y nodos sensores de bajo consumo con Bluetooth Low Energy (GATT y Mesh).
- Síntesis de la Parte 2B: selección óptima entre mallas autocurativas ZigBee 3.0 y Thread/Matter, telemetría de largo alcance LoRa / LoRaWAN, instrumentación crítica de planta con WirelessHART / ISA100.11a, localización centimétrica de robots con UWB y control inalámbrico en tiempo real con 5G Industrial URLLC.

2. Parámetros Nominales de Diseño de Red y Enlace RF (Video 05)

Parámetro de Comunicación / Radiofrecuencia	Símbolo	Valor Nominal	Función en la Red Industrial / Enlace Inalámbrico
Velocidad de Transmisión en Serie	`Baud Rate / Rb`	115200 bps	Caudal de transferencia en bus físico UART / RS-485 / CAN
Ancho de Banda de Canal	`B [kHz]`	125 kHz	Determina la capacidad de Shannon-Hartley y tasa de símbolos
Relación Señal a Ruido	`SNR [dB]`	15 dB	Margen sobre el piso de ruido térmico e interferencia industrial
Frecuencia Portadora ISM y Potencia TX	`f [MHz] / Pt [dBm]`	915 MHz / +16 dBm	Cálculo de longitud de onda λ, antena λ/4 y ecuación de Friis
Factor de Ensanchamiento LoRa y Alcance	`SF / d [m]`	SF9 / 1500 m	Compromiso entre sensibilidad de recepción y Tiempo en el Aire
Watchdog de Comunicación Determinista	`T_timeout [ms]`	30 ms	Ventana máxima antes de conmutar actuadores a estado Fail-Safe

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Síntesis de la Parte 1: Comunicaciones Digitales, Redes, Modelo OSI y Buses

De la Capacidad de Shannon y las Arquitecturas Centralizada/Jerárquica/Distribuida a los Buses Serie y Paralelo

- **Turno 1 (Masculina):** Completamos el curso de Comunicaciones y Redes Inalámbricas en Mecatrónica.
- **Turno 2 (Femenina):** En la Parte uno dominamos capacidad de Shannon, control distribuido y topologías.
- **Turno 3 (Masculina):** Aplicamos el modelo OSI, verificación ce erre ce, UART, ese pe i e i cuadrada ce.
- **Turno 4 (Femenina):** También integramos erre ese cuatro ochenta y cinco, CAN, EtherCAT y ge pe i be.

Diapositiva 02: Síntesis de la Parte 2: Ecosistema Inalámbrico Mecatrónico e Industrial

Presupuesto de Enlace RF, Wi-Fi 6/HaLow, Bluetooth BLE, ZigBee, LoRaWAN, WirelessHART, UWB y 5G URLLC

- **Turno 1 (Masculina):** En la Parte dos calculamos presupuestos de enlace con Friis en bandas ISM.
- **Turno 2 (Femenina):** Desplegamos Wi-Fi seis, Wi-Fi HaLow con eme cu te te y perfiles BLE GATT.
- **Turno 3 (Masculina):** Dominamos mallas ZigBee, modulación LoRaWAN, Thread, Matter y redes deterministas WirelessHART.
- **Turno 4 (Femenina):** Y cerramos con localización UWB y cinco G URLLC: apliquen estos códigos en planta.

4. Firmware C++/MCU: Trama Industrial, CRC-16 y FSM Fail-Safe (comunicacion_embebida_video_05.ino)

```
/*
 * ============================================================================
 * CURSO COMPLETO: SISTEMAS DE COMUNICACIÓN Y REDES INALÁMBRICAS EN MECATRÓNICA
 * VIDEO 05: Video 05 — Resumen Ejecutivo y Cierre del Curso de Sistemas de Comunicación y Redes Inalámbricas
 * Archivo: comunicacion_embebida_video_05.ino
 * ----------------------------------------------------------------------------
 * Descripción Técnica:
 *   Firmware determinista en C/C++ Embebido para Microcontrolador (ESP32-S3 /
 *   STM32 / Arduino Industrial) que implementa:
 *     1. Empaquetado y análisis sintáctico de tramas binarias industriales
 *        compatibles con RS-485 (Modbus RTU) y pasarelas inalámbricas (BLE/LoRa).
 *     2. Cálculo polinómico de Verificación de Redundancia Cíclica CRC-16
 *        (polinomio reflectado 0xA001) para inmunidad ante ruido electromagnético.
 *     3. Control de dirección Half-Duplex (pin DE/RE de transceptor MAX485) y
 *        lectura de telemetría de radiofrecuencia (RSSI / SNR).
 *     4. Máquina de Estados Finitos (FSM) no bloqueante con supervisión de
 *        Timeout (30 ms) y estado seguro (Fail-Safe) ante pérdida de enlace.
 *
 * Enfoque del Módulo:
 *   Arquitectura Híbrida Cableada + Inalámbrica: Bus de Campo RS-485/CAN/EtherCAT + Pasarela IIoT Wi-Fi 6 / BLE /
LoRaWAN / 5G URLLC
 * ============================================================================
 */

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

// --- Definición de Pines de Interfase de Comunicación y Estado Seguro ---
const uint8_t PIN_RS485_DE_RE   = 4;  // Control de dirección Half-Duplex (HIGH = TX, LOW = RX)
const uint8_t PIN_CS_SPI_RADIO  = 10; // Chip Select (CS#) para transceptor SPI (LoRa SX1262 / CAN)
const uint8_t PIN_LED_ENLACE_OK = 2;  // Indicador de trama válida recibida
const uint8_t PIN_DO_FAILSAFE   = 8;  // Salida de actuador con corte seguro por Timeout

// --- Parámetros Nominales de Comunicación (Video 05) ---
const uint32_t BAUD_RATE_SERIE   = 115200UL;
const uint8_t  NODO_DIRECCION_ID = 0x01;       // Dirección de esclavo / nodo mecatrónico
const uint32_t TIMEOUT_ENLACE_MS = 30UL;      // Tiempo máximo sin trama antes de Fail-Safe
const float    FREQ_PORTADORA_MHZ = 915.0f;
const int8_t   TX_POWER_DBM       = 16;
const uint8_t  LORA_SPREADING_SF  = 9;

// --- Estructura de Trama Industrial Unificada (12 Bytes con CRC-16) ---
// [0]: SOF (0xAA) | [1]: Nodo ID | [2]: Función | [3..4]: Setpoint (uint16)
// [5..6]: Variable Proceso PV (uint16) | [7]: RSSI_dBm (int8) | [8]: EstadoFlags
// [9]: Secuencia | [10..11]: CRC-16-Modbus (Little-Endian)
struct TramaMecatronica {
  uint8_t  sof;
  uint8_t  nodoId;
  uint8_t  funcion;
  uint16_t consignaSp;
  uint16_t variablePv;
  int8_t   rssiDbm;
  uint8_t  estadoFlags;
  uint8_t  secuencia;
  uint16_t crc16;
};

enum EstadoEnlaceFSM : uint8_t {
  ENLACE_ESPERANDO_SOF = 0,
  ENLACE_RECIBIENDO_BYTES,
  ENLACE_VALIDANDO_CRC,
  ENLACE_EJECUTANDO_RESPUESTA,
  ENLACE_ERROR_FAILSAFE
};

EstadoEnlaceFSM g_estadoEnlace = ENLACE_ESPERANDO_SOF;
uint8_t         g_bufferRx[12];
```

```
uint8_t         g_indiceRx = 0;
uint32_t        g_ultimoPaqueteValidoMs = 0;
uint8_t         g_contadorSecuenciaTx = 0;
uint16_t        g_consignaActiva = 0;

// --- Cálculo Polinómico CRC-16-Modbus (Polinomio 0xA001) ---
uint16_t calcularCrc16Modbus(const uint8_t *datos, uint8_t longitud) {
  uint16_t crc = 0xFFFFU;
  for (uint8_t i = 0; i < longitud; i++) {
    crc ^= static_cast<uint16_t>(datos[i]);
    for (uint8_t bit = 0; bit < 8; bit++) {
      if ((crc & 0x0001U) != 0U) {
        crc = (crc >> 1U) ^ 0xA001U;
      } else {
        crc >>= 1U;
      }
    }
  }
  return crc;
}

// --- Transmisión Half-Duplex sobre Par Diferencial RS-485 / Enlace Serie ---
void transmitirTramaRespuesta(uint8_t funcion, uint16_t pvActual, int8_t rssiEstimado) {
  uint8_t tramaTx[12];
  tramaTx[0] = 0xAAU;
  tramaTx[1] = NODO_DIRECCION_ID;
  tramaTx[2] = funcion;
  tramaTx[3] = static_cast<uint8_t>(g_consignaActiva & 0xFFU);
  tramaTx[4] = static_cast<uint8_t>((g_consignaActiva >> 8U) & 0xFFU);
  tramaTx[5] = static_cast<uint8_t>(pvActual & 0xFFU);
  tramaTx[6] = static_cast<uint8_t>((pvActual >> 8U) & 0xFFU);
  tramaTx[7] = static_cast<uint8_t>(rssiEstimado);
  tramaTx[8] = (g_estadoEnlace == ENLACE_ERROR_FAILSAFE) ? 0x80U : 0x01U;
  tramaTx[9] = g_contadorSecuenciaTx++;

  uint16_t crc = calcularCrc16Modbus(tramaTx, 10);
  tramaTx[10] = static_cast<uint8_t>(crc & 0xFFU);
  tramaTx[11] = static_cast<uint8_t>((crc >> 8U) & 0xFFU);

  // Conmutar transceptor diferencial a modo Transmisión (DE/RE = HIGH)
  digitalWrite(PIN_RS485_DE_RE, HIGH);
  Serial.write(tramaTx, 12);
  Serial.flush(); // Esperar vaciado completo del registro de desplazamiento UART
  digitalWrite(PIN_RS485_DE_RE, LOW);
}

void setup() {
  pinMode(PIN_RS485_DE_RE, OUTPUT);
  pinMode(PIN_CS_SPI_RADIO, OUTPUT);
  pinMode(PIN_LED_ENLACE_OK, OUTPUT);
  pinMode(PIN_DO_FAILSAFE, OUTPUT);

  digitalWrite(PIN_RS485_DE_RE, LOW);   // Modo escucha por defecto en bus Half-Duplex
  digitalWrite(PIN_CS_SPI_RADIO, HIGH); // Línea CS# inactiva en nivel alto
  digitalWrite(PIN_DO_FAILSAFE, LOW);

  Serial.begin(BAUD_RATE_SERIE);
  g_ultimoPaqueteValidoMs = millis();
}

void loop() {
  uint32_t ahoraMs = millis();

  // 1. Procesamiento No Bloqueante de Bytes Entrantes en la FSM de Protocolo
  while (Serial.available() > 0) {
    uint8_t byteLeido = static_cast<uint8_t>(Serial.read());

    if (g_estadoEnlace == ENLACE_ESPERANDO_SOF || g_estadoEnlace == ENLACE_ERROR_FAILSAFE) {
```

```
      if (byteLeido == 0xAAU) {
        g_bufferRx[0] = byteLeido;
        g_indiceRx = 1;
        g_estadoEnlace = ENLACE_RECIBIENDO_BYTES;
      }
    } else if (g_estadoEnlace == ENLACE_RECIBIENDO_BYTES) {
      g_bufferRx[g_indiceRx++] = byteLeido;
      if (g_indiceRx >= 12U) {
        g_estadoEnlace = ENLACE_VALIDANDO_CRC;
      }
    }

    if (g_estadoEnlace == ENLACE_VALIDANDO_CRC) {
      uint16_t crcRecibido = static_cast<uint16_t>(g_bufferRx[10]) |
                             (static_cast<uint16_t>(g_bufferRx[11]) << 8U);
      uint16_t crcCalculado = calcularCrc16Modbus(g_bufferRx, 10);

      if (crcRecibido == crcCalculado && g_bufferRx[1] == NODO_DIRECCION_ID) {
        g_consignaActiva = static_cast<uint16_t>(g_bufferRx[3]) |
                           (static_cast<uint16_t>(g_bufferRx[4]) << 8U);
        g_ultimoPaqueteValidoMs = ahoraMs;
        digitalWrite(PIN_LED_ENLACE_OK, HIGH);
        digitalWrite(PIN_DO_FAILSAFE, HIGH);

        uint16_t pvSensorLocal = static_cast<uint16_t>(analogRead(A0));
        int8_t rssiMedido = -68; // Ejemplo de nivel RSSI en dBm (-68 dBm)
        transmitirTramaRespuesta(g_bufferRx[2], pvSensorLocal, rssiMedido);
      }
      g_indiceRx = 0;
      g_estadoEnlace = ENLACE_ESPERANDO_SOF;
    }
  }

  // 2. Supervisión Determinista de Watchdog de Comunicación (Timeout Fail-Safe)
  if ((ahoraMs - g_ultimoPaqueteValidoMs) > TIMEOUT_ENLACE_MS) {
    g_estadoEnlace = ENLACE_ERROR_FAILSAFE;
    g_consignaActiva = 0U;
    digitalWrite(PIN_LED_ENLACE_OK, LOW);
    digitalWrite(PIN_DO_FAILSAFE, LOW); // Desenergiza actuador por pérdida de red
  }
}
```

5. Script Python SBC: Shannon-Hartley, Presupuesto de Enlace Friis y LoRa CSS (analisis_redes_y_enlace_rf_video_05.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: SISTEMAS DE COMUNICACIÓN Y REDES INALÁMBRICAS EN MECATRÓNICA
VIDEO 05: Video 05 — Resumen Ejecutivo y Cierre del Curso de Sistemas de Comunicación y Redes Inalámbricas
Archivo: analisis_redes_y_enlace_rf_video_05.py
--------------------------------------------------------------------------------
Descripción Técnica:
  Herramienta de ingeniería en Python para Computadora de Placa Única (SBC /
  Raspberry Pi / Gateway IIoT) que ejecuta:
    1. Cálculo de Capacidad de Canal de Shannon-Hartley y Límite de Nyquist.
    2. Cálculo del Presupuesto de Enlace de Radiofrecuencia (Link Budget),
       Pérdida en el Espacio Libre (Ecuación de Friis / FSPL) y Margen de
       Desvanecimiento (Fade Margin) en bandas ISM (915 MHz).
    3. Cálculo de Tiempo en el Aire (Time-on-Air) y tasa de bits de la
       modulación Chirp Spread Spectrum (CSS) de LoRa (SF9).
    4. Empaquetado y verificación de tramas Modbus RTU con CRC-16 y payload
       JSON para telemetría MQTT / OPC UA.

Enfoque del Módulo:
  Arquitectura Híbrida Cableada + Inalámbrica: Bus de Campo RS-485/CAN/EtherCAT + Pasarela IIoT Wi-Fi 6 / BLE /
LoRaWAN / 5G URLLC
================================================================================
"""

import math
import struct
import json
from typing import Dict, Any

# --- Parámetros Nominales del Enlace Mecatrónico (Video 05) ---
BW_CANAL_KHZ      = 125.0
SNR_DB_NOMINAL    = 15.0
FREQ_ISM_MHZ      = 915.0
TX_POWER_DBM      = 16.0
DISTANCIA_METROS  = 1500.0
LORA_SF_NOMINAL   = 9


def calcular_capacidad_shannon_nyquist(bw_khz: float, snr_db: float, niveles_m: int = 4) -> Dict[str, float]:
    """Calcula la capacidad máxima de Shannon-Hartley y la tasa de Nyquist."""
    bw_hz = bw_khz * 1000.0
    snr_lineal = 10.0 ** (snr_db / 10.0)
    capacidad_shannon_bps = bw_hz * math.log2(1.0 + snr_lineal)
    capacidad_nyquist_bps = 2.0 * bw_hz * math.log2(niveles_m)
    return {
        "Ancho_Banda_kHz": round(bw_khz, 2),
        "SNR_dB": round(snr_db, 2),
        "SNR_Lineal": round(snr_lineal, 4),
        "Capacidad_Nyquist_kbps": round(capacidad_nyquist_bps / 1000.0, 3),
        "Capacidad_Shannon_kbps": round(capacidad_shannon_bps / 1000.0, 3),
    }


def calcular_presupuesto_enlace_friis(
    pt_dbm: float,
    freq_mhz: float,
    distancia_m: float,
    gt_dbi: float = 2.15,
    gr_dbi: float = 2.15,
    perdidas_obstaculos_db: float = 12.0,
    sensibilidad_rx_dbm: float = -118.0,
) -> Dict[str, Any]:
    """Evalúa la ecuación de Friis, FSPL y el Margen de Desvanecimiento (Fade Margin)."""
    distancia_km = max(distancia_m / 1000.0, 0.001)
    longitud_onda_m = 299.792458 / freq_mhz
```

```
    antena_cuarto_onda_cm = (longitud_onda_m / 4.0) * 100.0

    # FSPL(dB) = 20*log10(d_km) + 20*log10(f_MHz) + 32.44
    fspl_db = 20.0 * math.log10(distancia_km) + 20.0 * math.log10(freq_mhz) + 32.44
    pr_dbm = pt_dbm + gt_dbi + gr_dbi - fspl_db - perdidas_obstaculos_db
    fade_margin_db = pr_dbm - sensibilidad_rx_dbm

    return {
        "Frecuencia_MHz": freq_mhz,
        "Longitud_Antena_Cuarto_Onda_cm": round(antena_cuarto_onda_cm, 2),
        "FSPL_Espacio_Libre_dB": round(fspl_db, 2),
        "Potencia_Recibida_RSSI_dBm": round(pr_dbm, 2),
        "Margen_Desvanecimiento_dB": round(fade_margin_db, 2),
        "Enlace_Industrial_Viable": fade_margin_db >= 15.0,
    }


def calcular_parametros_fisicos_lora(sf: int, bw_khz: float = 125.0, cr_denom: int = 5, payload_bytes: int = 16) -
> Dict[str, float]:
    """Calcula velocidad de símbolo, velocidad de bits y Time-on-Air para LoRa CSS."""
    bw_hz = bw_khz * 1000.0
    chips_por_simbolo = 2 ** sf
    tasa_simbolos_rs = bw_hz / chips_por_simbolo
    t_simbolo_ms = (1.0 / tasa_simbolos_rs) * 1000.0
    tasa_bits_bps = sf * tasa_simbolos_rs * (4.0 / cr_denom)

    # Estimación estándar de símbolos de trama y Time-on-Air
    de_low_rate = 1 if (sf >= 11 and bw_khz <= 125.0) else 0
    num = 8 * payload_bytes - 4 * sf + 28 + 16
    den = 4 * (sf - 2 * de_low_rate)
    n_payload = 8 + max(math.ceil(num / den) * cr_denom, 0)
    tiempo_aire_ms = (12.25 + n_payload) * t_simbolo_ms

    return {
        "Spreading_Factor": sf,
        "Chips_por_Simbolo": chips_por_simbolo,
        "Tasa_Bits_Efectiva_bps": round(tasa_bits_bps, 2),
        "Tiempo_en_el_Aire_ms": round(tiempo_aire_ms, 2),
    }


def calcular_crc16_modbus(data: bytes) -> int:
    """Calcula el CRC-16-Modbus (polinomio reflectado 0xA001)."""
    crc = 0xFFFF
    for b in data:
        crc ^= b
        for _ in range(8):
            if crc & 0x0001:
                crc = (crc >> 1) ^ 0xA001
            else:
                crc >>= 1
    return crc & 0xFFFF


def construir_trama_modbus_rtu(slave_id: int, funcion: int, reg_inicio: int, valor_reg: int) -> bytes:
    """Construye una trama Modbus RTU completa con verificación CRC-16."""
    cabecera = struct.pack(">BBHH", slave_id, funcion, reg_inicio, valor_reg)
    crc = calcular_crc16_modbus(cabecera)
    return cabecera + struct.pack("<H", crc)


if __name__ == "__main__":
    print("=" * 78)
    print(" CURSO: SISTEMAS DE COMUNICACIÓN Y REDES INALÁMBRICAS — VIDEO 05")
    print(" Enfoque:", "Arquitectura Híbrida Cableada + Inalámbrica: Bus de Campo RS-485/CAN/EtherCAT + Pasarela
IIoT Wi-Fi 6 / BLE / LoRaWAN / 5G URLLC")
    print("=" * 78)
```

```
    cap = calcular_capacidad_shannon_nyquist(BW_CANAL_KHZ, SNR_DB_NOMINAL)
    print("\n[1] Capacidad de Canal (Nyquist y Shannon-Hartley):", json.dumps(cap, ensure_ascii=False))

    rf = calcular_presupuesto_enlace_friis(TX_POWER_DBM, FREQ_ISM_MHZ, DISTANCIA_METROS)
    print("\n[2] Presupuesto de Enlace RF (Ecuación de Friis):", json.dumps(rf, ensure_ascii=False))

    lora_cfg = calcular_parametros_fisicos_lora(LORA_SF_NOMINAL, BW_CANAL_KHZ)
    print("\n[3] Parámetros Físicos LoRa CSS:", json.dumps(lora_cfg, ensure_ascii=False))

    trama_rtu = construir_trama_modbus_rtu(slave_id=1, funcion=0x06, reg_inicio=0x0100, valor_reg=1500)
    print("\n[4] Trama Modbus RTU con CRC-16:", trama_rtu.hex(" ").upper())
```

6. Programa PLC IEC 61131-3 (ST + IL): Pasarela de Red, Heartbeat y Watchdog (pasarela_industrial_plc_video_05.st)

```
(* ============================================================================
 * CURSO COMPLETO: SISTEMAS DE COMUNICACIÓN Y REDES INALÁMBRICAS EN MECATRÓNICA
 * VIDEO 05: Video 05 — Resumen Ejecutivo y Cierre del Curso de Sistemas de Comunicación y Redes Inalámbricas
 * Archivo: pasarela_industrial_plc_video_05.st
 * ----------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) + Equivalencia en Lista de
 *           Instrucciones (IL) para Pasarela de Comunicaciones Industriales
 * Descripción Técnica:
 *   Bloque de Función Industrial (FB_GestorRedMecatronica_V05) que
 *   supervisa el intercambio cíclico de datos entre el PLC maestro, buses de
 *   campo serie/Ethernet (Modbus RTU/TCP, CANopen, PROFINET) y nodos sensores
 *   inalámbricos (Wi-Fi MQTT, BLE, ZigBee, LoRaWAN, WirelessHART), validando:
 *     1. Contador de latido (Heartbeat) y secuencia de trama para detectar
 *        congelamiento o pérdida del enlace.
 *     2. Umbral de potencia recibida RSSI y relación señal a ruido SNR.
 *     3. Temporizador Watchdog de red (TON) de 30 ms con activación
 *        automática de estado seguro (Fail-Safe) en los actuadores de planta.
 * ============================================================================ *)

FUNCTION_BLOCK FB_GestorRedMecatronica_V05
VAR_INPUT
    (* Señales de Diagnóstico de Capa de Enlace y Transporte *)
    xTramaValidaCrcOk     : BOOL; (* 1 = Verificación polinómica CRC-16 correcta *)
    xParoEmergenciaRed_NC : BOOL; (* 1 = Canal de seguridad PROFIsafe / E-Stop OK *)
    byContadorHeartbeatRx : BYTE; (* Contador incremental enviado por el nodo remoto *)
    iRssiMedidoDbm        : INT;  (* Nivel de potencia recibida en dBm, ej. -72 *)
    iSnrMedidoDb          : INT;  (* Relación Señal a Ruido en dB, ej. 15 *)

    (* Datos de Proceso Recibidos en el Payload de la Trama *)
    iVariableProcesoRemota : INT; (* Valor crudo 0..27648 desde nodo remoto *)
    rConsignaLocalMaestro  : REAL := 75.0; (* Consigna a transmitir hacia actuador remoto *)
END_VAR

VAR_OUTPUT
    (* Banderas de Salud del Enlace y Conmutación Fail-Safe *)
    xEnlaceOperativoOk    : BOOL; (* 1 = Comunicación determinista activa *)
    xAlertaCalidadRfBaja  : BOOL; (* 1 = Margen de desvanecimiento reducido *)
    xFalloTimeoutFailSafe : BOOL; (* 1 = Pérdida de comunicación -> Parada segura *)

    (* Variables de Proceso Validadas hacia el Lazo de Control / SCADA *)
    rVariableProcesoSegura : REAL; (* Valor escalado o valor seguro por defecto *)
    iPalabraMandoTx        : INT;  (* Registro empaquetado para PDO / Modbus 0x06 *)
    byHeartbeatTx          : BYTE; (* Latido transmitido por el PLC maestro *)
END_VAR

VAR
    tonWatchdogRed         : TON;
    byHeartbeatPrevio      : BYTE := 16#00;
    xLatidoActualizado     : BOOL := FALSE;
    iUmbralMinimoRssiDbm   : INT  := -98;
END_VAR

(* ============================================================================
 * 1. DETECCIÓN DE ACTUALIZACIÓN DE LATIDO (HEARTBEAT) Y INTEGRIDAD CRC-16
 *    Equivalencia en Lista de Instrucciones (IL):
 *      LD     byContadorHeartbeatRx
 *      NE     byHeartbeatPrevio
 *      AND    xTramaValidaCrcOk
 *      ST     xLatidoActualizado
 * ============================================================================ *)
xLatidoActualizado := (byContadorHeartbeatRx <> byHeartbeatPrevio) AND xTramaValidaCrcOk;

IF xLatidoActualizado THEN
    byHeartbeatPrevio := byContadorHeartbeatRx;
END_IF;

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

```
 * 2. TEMPORIZADOR WATCHDOG DE COMUNICACIÓN (TON DE 30 MS)
 *    Si no llegan tramas válidas dentro de la ventana determinista, activa Fail-Safe
 * ========================================================================== *)
tonWatchdogRed(
    IN := NOT xLatidoActualizado,
    PT := T#30MS
);

xFalloTimeoutFailSafe := tonWatchdogRed.Q OR (NOT xParoEmergenciaRed_NC);
xEnlaceOperativoOk    := (NOT xFalloTimeoutFailSafe) AND xTramaValidaCrcOk;

(* ============================================================================
 * 3. SUPERVISIÓN DEL PRESUPUESTO DE ENLACE INALÁMBRICO (RSSI Y SNR)
 * ========================================================================== *)
IF (iRssiMedidoDbm < iUmbralMinimoRssiDbm) OR (iSnrMedidoDb < -15) THEN
    xAlertaCalidadRfBaja := TRUE;
ELSE
    xAlertaCalidadRfBaja := FALSE;
END_IF;

(* ============================================================================
 * 4. GESTIÓN DETERMINISTA DE DATOS Y CONMUTACIÓN A ESTADO SEGURO (FAIL-SAFE)
 * ========================================================================== *)
IF xEnlaceOperativoOk THEN
    (* Escala la variable remota (0..27648 -> 0.0..100.0 %) y empaqueta consigna *)
    rVariableProcesoSegura := (INT_TO_REAL(iVariableProcesoRemota) / 27648.0) * 100.0;
    iPalabraMandoTx        := REAL_TO_INT((rConsignaLocalMaestro / 100.0) * 27648.0);
    byHeartbeatTx          := byHeartbeatTx + 16#01;
ELSE
    (* Estado Seguro (Fail-Safe): Fuerza consigna de potencia a cero *)
    rVariableProcesoSegura := 0.0;
    iPalabraMandoTx        := 0;
END_IF;

END_FUNCTION_BLOCK
```