CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA

Video 03 — Parte 2A: Microcontroladores y Microprocesadores — Arquitectura Interna y Fundamentos de Hardware — MPU frente a MCU, Von Neumann vs. Harvard, RISC vs. CISC, ALU/FPU/DSP, Memorias, Registros SFR, Reloj PLL y Timers

Parte del Curso: Parte 2A — Microcontroladores y Microprocesadores: Arquitectura Interna y Hardware
Arquitectura / Controlador Estudiado: Arquitectura Harvard RISC (ARM Cortex-M), FPU/DSP, Mapa de Memoria, Registros SFR, Reloj PLL y Timers PWM

```
f_PWM = f_TIM / ((PSC + 1)·(ARR + 1)) | Duty = CCRx / (ARR + 1) | Instrucción MAC: Acc <= Acc + (A × B)
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Diferencias entre Microprocesador (MPU multichip con MMU para OS) y Microcontrolador (MCU monolítico SoC determinista con Flash, SRAM y periféricos integrados).
- Arquitectura Harvard de buses separados (Fetch de instrucciones en Flash simultáneo con acceso a datos en SRAM), cauce RISC Load-Store, FPU IEEE 754 y extensión DSP MAC.
- Jerarquía de memorias (Flash, SRAM con Stack/Heap, EEPROM/FRAM y Core Coupled Memory CCM) y banco de registros del núcleo (R0-R12, SP, LR, PC, xPSR).
- Registros de Funciones Especiales (SFR) de puertos GPIO (MODER, ODR, IDR, BSRR), árbol de reloj con cristal de cuarzo y PLL, temporizadores con Input Capture, PWM complementario con Dead-Time y Watchdog (WDT).

2. Parámetros Nominales de Configuración del Controlador (Video 03)

Parámetro del Controlador / Interfase	Símbolo	Valor Nominal	Función en el Lazo de Control / Automatización
Ganancia Proporcional	`Kp (PB = 100/Kp)`	5	Esfuerzo inmediato proporcional al error actual e(t)
Tiempo Integral	`Ti [s]`	0.25 s	Elimina el error estacionario (Offset) acumulando la integral del error
Tiempo Derivativo y Filtro Divisor	`Td [s] / N`	Td = 0.04 s, N = 16	Anticipa la tendencia sobre -PV y atenúa el ruido de alta frecuencia
Periodo de Muestreo en Tiempo Real	`Ts [ms]`	0.5 ms	Intervalo determinista de conversión ADC, cálculo discreto y actualización PWM
Límites de Saturación Anti-Windup	`[u_min, u_max]`	[0, 1000]	Cota física del actuador para integración condicional (Clamping)
Rango Escalado de Ingeniería (SCALE_X)	`[PV_min, PV_max]`	[0, 3.3]	Conversión lineal desde cuentas crudas del ADC (0..1023 o 0..27648)

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Diferencias Estructurales entre Microprocesador (MPU) y Microcontrolador (MCU)

Sistema Multichip de Alto Cómputo frente a Integración Monolítica en un Chip (SoC) para Control Determinista

- **Turno 1 (Masculina):** Al iniciar la Parte dos del curso debemos responder una pregunta fundamental de arquitectura de hardware: ¿cuál es la diferencia estructural y funcional entre un microprocesador y un microcontrolador?
- **Turno 2 (Femenina):** Un microprocesador o M P U es una unidad central de procesamiento pura diseñada para maximizar la potencia de cálculo: no posee memoria de programa ni periféricos físicos dentro de su pastilla, por lo que requiere chips externos de memoria RAM dinámica y almacenamiento conectados mediante buses de alta velocidad.
- **Turno 3 (Masculina):** Por el contrario, un microcontrolador o M C U integra en un único chip monolítico de silicio el núcleo del procesador, la memoria Flash no volátil, la memoria SRAM estática, los temporizadores, los conversores analógico-digitales y los puertos de entrada y salida.
- **Turno 4 (Femenina):** Mientras el microprocesador ejecuta sistemas operativos complejos con alto consumo energético, el microcontrolador está optimizado para arrancar en microsegundos, consumir miliamperios y garantizar respuesta determinista en lazos de control en tiempo real.

Diapositiva 02: Arquitecturas de Computación: Von Neumann frente a Harvard

Cuello de Botella del Bus Único Compartido frente a Buses Separados de Instrucciones (Flash) y Datos (SRAM)

- **Turno 1 (Masculina):** La velocidad con la que un procesador ejecuta un algoritmo de control depende de cómo se conecta internamente su núcleo con las memorias de programa y de datos, existiendo dos arquitecturas clásicas: Von Neumann y Harvard.
- **Turno 2 (Femenina):** En la arquitectura Von Neumann existe un único espacio de memoria y un solo bus compartido para código y variables: si la CPU está leyendo una instrucción de programa, no puede al mismo tiempo leer ni escribir un dato en memoria, generando el famoso cuello de botella de Von Neumann.
- **Turno 3 (Masculina):** En cambio, la arquitectura Harvard —empleada en la mayoría de los microcontroladores como AVR, PIC y ARM Cortex-M— separa físicamente la memoria Flash de instrucciones y la memoria SRAM de datos, dotando a cada una de su propio bus independiente.
- **Turno 4 (Femenina):** Gracias a esta doble autopista paralela, el microcontrolador realiza accesos simultáneos en un mismo ciclo de reloj: mientras ejecuta la multiplicación del dato actual sobre el bus de SRAM, ya está trayendo la siguiente instrucción desde el bus de Flash.

Diapositiva 03: Repertorio de Instrucciones: RISC (ARM Cortex-M, RISC-V) frente a CISC

Filosofía Load-Store, Segmentación de Cauce (Pipeline) y Ejecución Determinista en un Ciclo de Reloj

- **Turno 1 (Masculina):** Desde el punto de vista del lenguaje máquina que entiende el silicio, los procesadores se dividen en arquitecturas de conjunto complejo de instrucciones, llamadas CISC, y arquitecturas de conjunto reducido de instrucciones, llamadas RISC.
- **Turno 2 (Femenina):** Un procesador CISC, como la familia equis ochenta y seis de computadoras personales, dispone de cientos de instrucciones complejas de longitud variable capaces de operar directamente sobre la memoria, pero que tardan múltiples ciclos de reloj impredecibles en completarse.
- **Turno 3 (Masculina):** Por el contrario, los núcleos de microcontroladores modernos como ARM Cortex-M y RISC-V siguen la filosofía RISC con arquitectura Load-Store: las instrucciones son simples, uniformes, y toda operación aritmética se ejecuta exclusivamente sobre registros internos en un solo ciclo.
- **Turno 4 (Femenina):** Esta regularidad permite implementar una línea de ensamblaje interna llamada segmentación de cauce o Pipeline de tres etapas: mientras una instrucción se ejecuta, la siguiente se decodifica y una tercera se busca en memoria, maximizando el rendimiento por megahercio.

Diapositiva 04: Unidad Aritmético-Lógica (ALU), Punto Flotante (FPU) y Extensiones DSP

Operaciones Enteras, Estándar IEEE 754 por Hardware e Instrucción MAC (Multiply-Accumulate) en un Ciclo

- **Turno 1 (Masculina):** En el corazón ejecutor del núcleo del microcontrolador se encuentra la Unidad Aritmético-Lógica o A L U, encargada de realizar sumas, restas, desplazamientos y operaciones lógicas bit a bit sobre palabras enteras de ocho, dieciséis o treinta y dos bits.
- **Turno 2 (Femenina):** En un microcontrolador económico que solo posee A L U entera, escribir una multiplicación de números con decimales tipo float obliga al compilador a insertar rutinas de emulación por software que consumen decenas o cientos de ciclos de reloj.
- **Turno 3 (Masculina):** Por ello, para control mecatrónico avanzado elegimos núcleos como ARM Cortex-M4, Cortex-M7 o ESP32-S3, que incorporan una Unidad de Punto Flotante o F P U por hardware compatible con el estándar IEEE setecientos cincuenta y cuatro, capaz de multiplicar y sumar números reales en un solo ciclo.
- **Turno 4 (Femenina):** Además, estos núcleos añaden extensiones de procesamiento digital de señales con la instrucción MAC de multiplicación y acumulación en un ciclo y aritmética saturada, permitiendo calcular filtros digitales y transformadas rápidas de Fourier a velocidades extraordinarias.

Diapositiva 05: Jerarquía de Memorias en un Microcontrolador

Memoria Flash No Volátil, SRAM (Datos Estáticos, Pila Stack y Heap), EEPROM/FRAM y Memoria CCM

- **Turno 1 (Masculina):** Un ingeniero de firmware debe conocer con exactitud dónde se aloja cada byte dentro de las distintas tecnologías de memoria que conviven en el interior de un microcontrolador.
- **Turno 2 (Femenina):** La primera es la memoria Flash no volátil: retiene su contenido sin energía eléctrica, tiene alta densidad y aloja el código binario del programa junto con las tablas de constantes declaradas con el calificador const, aunque su escritura es lenta por bloques y tiene ciclos de borrado limitados.
- **Turno 3 (Masculina):** La segunda es la memoria estática de acceso aleatorio o SRAM: es volátil pero ultrarrápida a nivel de byte, y en ella residen las variables globales, el montículo dinámico o Heap y la Pila o Stack, donde se guardan las variables locales y el contexto de las interrupciones.
- **Turno 4 (Femenina):** Completan la jerarquía la memoria EEPROM o FRAM para guardar parámetros de calibración y ganancias pe i de que deben sobrevivir a un apagado, y en microcontroladores de alto desempeño la memoria acoplada al núcleo o C C M, que ejecuta rutinas críticas sin ningún estado de espera.

Diapositiva 06: Banco de Registros Internos del Núcleo CPU

Registros de Propósito General (R0-R12), Puntero de Pila (SP), Registro de Enlace (LR), Contador de Programa (PC) y PSR

- **Turno 1 (Masculina):** La memoria más rápida de todo el microcontrolador no es siquiera la SRAM, sino el banco de registros internos construidos con flip-flops directamente dentro del núcleo de la CPU.
- **Turno 2 (Femenina):** Tomando como referencia la arquitectura industrial ARM Cortex-M de treinta y dos bits, disponemos de trece registros de propósito general numerados de erre cero a erre doce, donde el compilador mantiene los operandos de las fórmulas y pasa los parámetros entre funciones.
- **Turno 3 (Masculina):** Los tres registros siguientes tienen misiones estructurales: erre trece es el Puntero de Pila o Stack Pointer, que apunta siempre al tope actual de la pila en SRAM; y erre catorce es el Registro de Enlace o Link Register, que almacena la dirección exacta a la que debe regresar el programa tras terminar una función.
- **Turno 4 (Femenina):** Finalmente, erre quince es el Contador de Programa o Program Counter, que contiene la dirección de la instrucción en ejecución; acompañado por el Registro de Estado del Programa o P S R, cuyos bits indican si la última operación dio resultado cero, negativo, acarreo o desbordamiento aritmético.

Diapositiva 07: Registros de Funciones Especiales (SFR) y Puertos GPIO

Periféricos Mapeados en Memoria, Registros MODER/ODR/IDR/BSRR, Pull-Up/Down y Drenador Abierto

- **Turno 1 (Masculina):** ¿Cómo logra una instrucción de software dentro del núcleo encender un transistor físico o leer un sensor externo? El puente entre el software y el silicio son los Registros de Funciones Especiales, abreviados S F R.
- **Turno 2 (Femenina):** Cada periférico físico del microcontrolador expone un conjunto de registros ubicados en direcciones fijas del mapa de memoria: escribir un uno o un cero en un bit específico de esa dirección acciona directamente compuertas electrónicas reales en el chip.
- **Turno 3 (Masculina):** En un puerto de Entrada y Salida de Propósito General o G P I O, el registro de modo configura si cada pin trabajará como entrada digital, salida digital, función alternativa de comunicación o entrada analógica desconectada del buffer digital.
- **Turno 4 (Femenina):** A su vez, los registros internos permiten activar resistencias de Pull-Up o Pull-Down para fijar un nivel lógico limpio cuando un interruptor está abierto, y elegir entre una etapa de salida Push-Pull de dos transistores o una etapa de Drenador Abierto para buses compartidos.

Diapositiva 08: Sistema de Reloj (Clock Tree): Osciladores, PLL y Buses AHB/APB

Oscilador Interno RC frente a Cristal de Cuarzo (ppm), Multiplicador de Fase (PLL) y Divisores Prescaler

- **Turno 1 (Masculina):** Todo lo que ocurre dentro de un microcontrolador está sincronizado por el árbol de distribución de reloj o Clock Tree, cuya exactitud determina la precisión de los lazos de control y de las comunicaciones en serie.
- **Turno 2 (Femenina):** Los microcontroladores incluyen un oscilador interno de resistencia-capacitor que permite arrancar sin componentes externos, pero cuya frecuencia deriva hasta un uno o dos por ciento con los cambios de temperatura; por ello en aplicaciones de precisión conectamos un cristal de cuarzo externo cuya estabilidad se mide en pocas partes por millón.
- **Turno 3 (Masculina):** ¿Cómo alcanza un microcontrolador frecuencias de ciento sesenta y ocho o cuatrocientos megahercios a partir de un cristal externo de solo ocho megahercios? Mediante un multiplicador electrónico interno llamado Lazo de Enganche de Fase o P L L.
- **Turno 4 (Femenina):** La frecuencia maestra generada por el P L L se reparte luego mediante divisores programables o Prescalers hacia el bus de alto rendimiento del sistema, llamado A H B, y hacia los buses de periféricos rápidos y lentos, llamados A P B, permitiendo apagar el reloj de los bloques no utilizados para ahorrar energía.

Diapositiva 09: Temporizadores por Hardware (Timers), PWM con Tiempo Muerto y Watchdog

Prescaler, Auto-Recarga (ARR), Input Capture, Output Compare, Dead-Time en Puentes H y Perro Guardián (WDT)

- **Turno 1 (Masculina):** Los periféricos más importantes para el ingeniero mecatrónico son los Temporizadores o Timers por hardware: contadores binarios autónomos que incrementan su cuenta con cada pulso de reloj sin consumir ni un solo ciclo de la CPU.
- **Turno 2 (Femenina):** Un temporizador consta de un divisor Prescaler que ajusta la resolución temporal y de un registro de Auto-Recarga que define el periodo exacto de conteo: en modo Captura de Entrada congela el valor del contador al llegar un flanco externo para medir anchos de pulso con precisión de nanosegundos.
- **Turno 3 (Masculina):** En modo Comparación y Modulación por Ancho de Pulso o P W M, el temporizador compara su cuenta con el registro de ciclo de trabajo para conmutar los pines de potencia, incorporando en los timers avanzados un generador de Tiempo Muerto o Dead-Time por hardware que impide cortocircuitos en las ramas de un puente H.
- **Turno 4 (Femenina):** Por último, todo sistema mecatrónico crítico debe activar el Perro Guardián o Watchdog Timer: un contador independiente con su propio oscilador que reinicia automáticamente al microcontrolador si el programa principal sufre un bloqueo por ruido electromagnético y deja de refrescarlo a tiempo.

Diapositiva 10: Síntesis de Arquitectura de Microcontroladores y Microprocesadores

Integración de CPU, Memorias, Reloj y Periféricos de Tiempo Real rumbo a la Programación Embebida

- **Turno 1 (Masculina):** En esta Parte dos A hemos abierto la cápsula de silicio para comprender cómo interactúan la arquitectura Harvard, el cauce RISC, la unidad de punto flotante, la jerarquía de memorias Flash y SRAM, los registros periféricos y los temporizadores.
- **Turno 2 (Femenina):** Comprender esta arquitectura física es lo que distingue a un programador casual de un verdadero ingeniero en sistemas embebidos: sabemos por qué una operación en punto flotante requiere F P U y cómo el temporizador genera señales P W M autónomas.
- **Turno 3 (Masculina):** Con estos fundamentos de hardware firmemente establecidos, estamos listos para escribir firmware determinista de nivel industrial que aproveche al máximo cada periférico.
- **Turno 4 (Femenina):** En el siguiente módulo, correspondiente a la Parte dos B, dominaremos la programación en C embebido a nivel de registros, el calificador volatile, las interrupciones anidadas, el acceso directo a memoria y los sistemas operativos de tiempo real.

4. Firmware C++/MCU: PID Discreto Tustin, FSM y CRC-16 (controlador_tiempo_real_video_03.ino)

```
/*
 * ============================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 03: Video 03 — Parte 2A: Microcontroladores y Microprocesadores — Arquitectura Interna y Fundamentos de Hardware
 * Archivo: controlador_tiempo_real_video_03.ino
 * ----------------------------------------------------------------------------
 * Descripción Técnica:
 *   Firmware determinista en C/C++ Embebido para Microcontrolador (compatible
 *   con arquitectura AVR / ARM Cortex-M STM32 / ESP32 / RP2040) que implementa:
 *     1. Acceso a nivel de registros con punteros 'volatile' y máscaras de bits.
 *     2. Controlador PID Discreto por Transformación Bilineal de Tustin con
 *        derivada sobre la variable medida (-PV), filtro derivativo N y
 *        esquema Anti-Windup por retro-cálculo y Clamping.
 *     3. Máquina de Estados Finitos (FSM de Moore) no bloqueante para
 *        secuenciación segura e interbloqueo de emergencia.
 *     4. Trama de telemetría con verificación CRC-16-CCITT hacia SBC/PLC.
 *
 * Enfoque del Módulo:
 *   Arquitectura Harvard RISC (ARM Cortex-M), FPU/DSP, Mapa de Memoria, Registros SFR, Reloj PLL y Timers PWM
 * ============================================================================
 */

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

// --- Definición de Pines de Interfase (Entradas Optoaisladas y Salidas PWM/MOSFET) ---
const uint8_t PIN_ADC_PV_SENSOR   = A0; // Entrada analógica PV (ej. Shunt 165 ohm de lazo 4-20 mA)
const uint8_t PIN_ADC_SP_CONSIGNA = A1; // Consigna local o referencia analógica (0..5V / 0..3.3V)
const uint8_t PIN_DI_MARCHA_NO    = 2;  // Entrada optoacoplada PC817: Pulsador de Marcha (NO)
const uint8_t PIN_DI_PARO_NC      = 3;  // Entrada optoacoplada PC817: Paro / Emergencia (Cableado NC)
const uint8_t PIN_PWM_ACTUADOR    = 9;  // Salida PWM hacia Driver MOSFET / Puente H
const uint8_t PIN_DO_SSR_ALARMA   = 8;  // Salida hacia Relé de Estado Sólido (SSR) / Piloto

// --- Parámetros Nominales del Controlador PID Discreto (Video 03) ---
const float KP_GAIN      = 5.000f;
const float TI_SEC       = 0.250f;
const float TD_SEC       = 0.040f;
const float N_FILT       = 16.0f;
const float TS_SEC       = 0.0005f; // Periodo de muestreo Ts = 0.5 ms
const float U_MIN_LIM    = 0.0f;
const float U_MAX_LIM    = 1000.0f;
const float PV_MIN_ENG   = 0.0f;
const float PV_MAX_ENG   = 3.3f;

// --- Variables Volátiles Compartidas con Interrupción (ISR) ---
volatile bool     g_flagTickControl = false;
volatile uint32_t g_contadorPulsosEncoder = 0;

// --- Estructura de Estado del Controlador PID Discreto (Tustin + Anti-Windup) ---
struct ControladorPIDDiscreto {
  float kp;
  float ki;
  float kd;
  float n_divisor;
  float ts;
  float u_min;
  float u_max;
  float integral_acum;
  float termino_d_prev;
  float pv_prev;
  float error_prev;
  float u_salida;
};

ControladorPIDDiscreto g_pid;
```

```
// --- Máquina de Estados Finitos (FSM No Bloqueante de Moore) ---
enum EstadoControlador : uint8_t {
  ESTADO_REPOSO_SEGURO   = 0,
  ESTADO_PRECARGA_SUAVE  = 1,
  ESTADO_LAZO_PID_ACTIVO = 2,
  ESTADO_FALLA_PARO_NC   = 3
};

EstadoControlador g_estadoActual = ESTADO_REPOSO_SEGURO;
uint32_t g_prevMicrosLazo  = 0;
uint32_t g_timestampEstado = 0;
uint32_t g_pasoMuestreo    = 0;

// --- Rutina de Servicio de Interrupción Externa (Encoder / Sensor Rápido) ---
void isrPulsoEncoder() {
  g_contadorPulsosEncoder++;
}

// --- Inicialización de Coeficientes del PID Discreto ---
void inicializarPID(ControladorPIDDiscreto &c) {
  c.kp             = KP_GAIN;
  c.ki             = (TI_SEC > 1e-5f) ? (KP_GAIN / TI_SEC) : 0.0f;
  c.kd             = KP_GAIN * TD_SEC;
  c.n_divisor      = N_FILT;
  c.ts             = TS_SEC;
  c.u_min          = U_MIN_LIM;
  c.u_max          = U_MAX_LIM;
  c.integral_acum  = 0.0f;
  c.termino_d_prev = 0.0f;
  c.pv_prev        = 0.0f;
  c.error_prev     = 0.0f;
  c.u_salida       = 0.0f;
}

// --- Evaluación en Tiempo Real del PID por Regla Trapezoidal de Tustin ---
float evaluarPasoPID_Tustin(ControladorPIDDiscreto &c, float sp, float pv) {
  float error = sp - pv;

  // 1. Término Proporcional
  float p_term = c.kp * error;

  // 2. Término Derivativo sobre -PV filtrado por Tustin (Evita Derivative Kick)
  float tau_d  = (c.kp > 1e-6f) ? ((c.kd / c.kp) / c.n_divisor) : 0.0f;
  float denom  = 2.0f * tau_d + c.ts;
  float alpha1 = (2.0f * tau_d - c.ts) / denom;
  float beta0  = (2.0f * c.kd) / denom;
  float d_term = alpha1 * c.termino_d_prev - beta0 * (pv - c.pv_prev);

  // 3. Candidato de Integración Trapezoidal (Tustin)
  float delta_i = 0.5f * c.ki * c.ts * (error + c.error_prev);
  float i_cand  = c.integral_acum + delta_i;

  // 4. Suma Pre-Saturación y Esquema Anti-Windup Condicional (Clamping)
  float u_unsat = p_term + i_cand + d_term;
  float u_sat   = u_unsat;
  if (u_unsat > c.u_max) {
    u_sat = c.u_max;
    if (error < 0.0f) c.integral_acum = i_cand; // Solo integra si ayuda a desaturar
  } else if (u_unsat < c.u_min) {
    u_sat = c.u_min;
    if (error > 0.0f) c.integral_acum = i_cand;
  } else {
    c.integral_acum = i_cand;
  }

  // Actualizar memoria de estados discretos z^-1
  c.termino_d_prev = d_term;
  c.pv_prev        = pv;
```

```
  c.error_prev     = error;
  c.u_salida       = u_sat;
  return u_sat;
}

// --- Escalado Lineal Equivalente a NORM_X + SCALE_X ---
inline float escalarCuentasADC(int cuentasCrudas, int rawMin, int rawMax, float engMin, float engMax) {
  float u_norm = (float)(cuentasCrudas - rawMin) / (float)(rawMax - rawMin);
  if (u_norm < 0.0f) u_norm = 0.0f;
  if (u_norm > 1.0f) u_norm = 1.0f;
  return engMin + u_norm * (engMax - engMin);
}

// --- Cálculo de Verificación de Redundancia Cíclica CRC-16-CCITT (Polinomio 0x1021) ---
uint16_t calcularCRC16(const uint8_t *datos, uint8_t longitud) {
  uint16_t crc = 0xFFFFU;
  for (uint8_t i = 0; i < longitud; i++) {
    crc ^= ((uint16_t)datos[i] << 8);
    for (uint8_t b = 0; b < 8; b++) {
      if (crc & 0x8000U) {
        crc = (crc << 1) ^ 0x1021U;
      } else {
        crc <<= 1;
      }
    }
  }
  return crc;
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN_DI_MARCHA_NO, INPUT_PULLUP);
  pinMode(PIN_DI_PARO_NC, INPUT_PULLUP);
  pinMode(PIN_PWM_ACTUADOR, OUTPUT);
  pinMode(PIN_DO_SSR_ALARMA, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(PIN_DI_MARCHA_NO), isrPulsoEncoder, RISING);
  inicializarPID(g_pid);
  g_prevMicrosLazo  = micros();
  g_timestampEstado = millis();

  Serial.println(F("# ============================================================="));
  Serial.println(F("# CURSO CONTROLADORES EN MECATRONICA - VIDEO 03"));
  Serial.println(F("# Columnas: t_s,estado_fsm,SP_eng,PV_eng,u_PID,pulsos_enc,crc16"));
  Serial.println(F("# ============================================================="));
}

void loop() {
  uint32_t ahoraMicros = micros();
  uint32_t periodoMicros = (uint32_t)(TS_SEC * 1000000.0f);

  if (ahoraMicros - g_prevMicrosLazo >= periodoMicros) {
    g_prevMicrosLazo += periodoMicros;
    float t_sec = g_pasoMuestreo * TS_SEC;

    // 1. Lectura de Entradas Digitales (Seguridad de Paro NC y Marcha NO)
    bool paroCableadoOK = (digitalRead(PIN_DI_PARO_NC) == HIGH);
    bool ordenMarcha    = (digitalRead(PIN_DI_MARCHA_NO) == LOW);

    // 2. Lectura y Escalado de Entradas Analógicas (NORM_X -> SCALE_X)
    int rawPV = analogRead(PIN_ADC_PV_SENSOR);
    int rawSP = analogRead(PIN_ADC_SP_CONSIGNA);
    float pv_eng = escalarCuentasADC(rawPV, 0, 1023, PV_MIN_ENG, PV_MAX_ENG);
    float sp_eng = escalarCuentasADC(rawSP, 0, 1023, PV_MIN_ENG, PV_MAX_ENG);

    // 3. Evaluación de la Máquina de Estados Finitos (FSM de Moore No Bloqueante)
    if (!paroCableadoOK) {
      g_estadoActual = ESTADO_FALLA_PARO_NC;
```

```
}

float mandoActuador = 0.0f;
switch (g_estadoActual) {
  case ESTADO_REPOSO_SEGURO:
    mandoActuador = 0.0f;
    digitalWrite(PIN_DO_SSR_ALARMA, LOW);
    if (ordenMarcha && paroCableadoOK) {
      g_timestampEstado = millis();
      g_estadoActual    = ESTADO_PRECARGA_SUAVE;
    }
    break;

  case ESTADO_PRECARGA_SUAVE:
    mandoActuador = 0.25f * U_MAX_LIM;
    if (millis() - g_timestampEstado >= 200UL) {
      // Transferencia sin saltos (Bumpless) al activar el lazo PID
      g_pid.integral_acum = mandoActuador;
      g_pid.pv_prev       = pv_eng;
      g_estadoActual      = ESTADO_LAZO_PID_ACTIVO;
    }
    break;

  case ESTADO_LAZO_PID_ACTIVO:
    mandoActuador = evaluarPasoPID_Tustin(g_pid, sp_eng, pv_eng);
    digitalWrite(PIN_DO_SSR_ALARMA, LOW);
    break;

  case ESTADO_FALLA_PARO_NC:
  default:
    mandoActuador = 0.0f;
    g_pid.integral_acum = 0.0f;
    digitalWrite(PIN_DO_SSR_ALARMA, HIGH);
    if (paroCableadoOK && ordenMarcha) {
      g_estadoActual = ESTADO_REPOSO_SEGURO;
    }
    break;
}

// 4. Escritura de Salida PWM hacia Etapa de Potencia (MOSFET / Puente H)
float u_norm_pwm = (mandoActuador - U_MIN_LIM) / (U_MAX_LIM - U_MIN_LIM);
int duty8bit     = constrain((int)(u_norm_pwm * 255.0f), 0, 255);
analogWrite(PIN_PWM_ACTUADOR, duty8bit);

// 5. Lectura Atómica de Contador de Interrupción y Telemetría con CRC-16
if (g_pasoMuestreo % 10 == 0) {
  noInterrupts();
  uint32_t pulsosCopia = g_contadorPulsosEncoder;
  interrupts();

  uint8_t tramaBytes[6];
  tramaBytes[0] = (uint8_t)g_estadoActual;
  tramaBytes[1] = (uint8_t)duty8bit;
  tramaBytes[2] = (uint8_t)((rawPV >> 8) & 0xFF);
  tramaBytes[3] = (uint8_t)(rawPV & 0xFF);
  tramaBytes[4] = (uint8_t)((pulsosCopia >> 8) & 0xFF);
  tramaBytes[5] = (uint8_t)(pulsosCopia & 0xFF);
  uint16_t crc  = calcularCRC16(tramaBytes, 6);

  Serial.print(t_sec, 3);              Serial.print(',');
  Serial.print((int)g_estadoActual);   Serial.print(',');
  Serial.print(sp_eng, 2);             Serial.print(',');
  Serial.print(pv_eng, 2);             Serial.print(',');
  Serial.print(mandoActuador, 2);      Serial.print(',');
  Serial.print(pulsosCopia);           Serial.print(",0x");
  Serial.println(crc, HEX);
}
g_pasoMuestreo++;
```

```
  }
}
```

5. Script Python SBC: Sintonización, Índices IAE/ITAE y Tramas (supervision_sbc_y_sintonizacion_video_03.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
VIDEO 03: Video 03 — Parte 2A: Microcontroladores y Microprocesadores — Arquitectura Interna y Fundamentos de
Hardware
Archivo: supervision_sbc_y_sintonizacion_video_03.py
--------------------------------------------------------------------------------
Descripción Técnica:
  Script integral en Python para Computadora de Placa Única (Raspberry Pi /
  SBC) y estación de ingeniería que implementa:
    1. Simulación en lazo cerrado de un controlador PID discreto (Tustin +
       Filtro Derivativo N + Anti-Windup por Clamping y Back-Calculation).
    2. Cálculo cuantitativo de métricas transitorias (tr, tp, Mp, ts) y de
       los cuatro índices integrales de desempeño (IAE, ISE, ITAE, ITSE).
    3. Reglas automáticas de sintonización de Ziegler-Nichols (1.º y 2.º
       método), Cohen-Coon y sintonización robusta IMC / Lambda.
    4. Construcción y validación de tramas binarias Maestro-Coprocesador
       (SBC <-> MCU/PLC) protegidas con suma de verificación CRC-16-CCITT.
================================================================================
"""

import math
import struct
from typing import Dict, List, Tuple

# Parámetros nominales del Video 03
KP_NOMINAL = 5
TI_NOMINAL = 0.25
TD_NOMINAL = 0.04
N_FILTRO   = 16
TS_SEG     = 0.0005
U_MIN      = 0
U_MAX      = 1000


def sintonizar_pid_ziegler_nichols_y_lambda(
    K_planta: float, L_retardo: float, T_constante: float, lambda_deseado: float
) -> Dict[str, Dict[str, float]]:
    """
    Calcula las ganancias PID (Kp, Ti, Td) mediante:
      - Ziegler-Nichols 1.er Método (Curva de Reacción FOPDT)
      - Método de Cohen-Coon (para retardo significativo L/T)
      - Método IMC / Regla Lambda (respuesta sin sobreimpulso)
    """
    r = L_retardo / max(T_constante, 1e-6)

    # 1. Ziegler-Nichols Lazo Abierto (PID Estándar ISA)
    kp_zn = (1.2 * T_constante) / (K_planta * max(L_retardo, 1e-6))
    ti_zn = 2.0 * L_retardo
    td_zn = 0.5 * L_retardo

    # 2. Cohen-Coon (PID)
    kp_cc = (1.0 / (K_planta * r)) * (4.0 / 3.0 + r / 4.0)
    ti_cc = L_retardo * (32.0 + 6.0 * r) / (13.0 + 8.0 * r)
    td_cc = L_retardo * 4.0 / (11.0 + 2.0 * r)

    # 3. IMC / Lambda (PI robusto sin sobreimpulso + término D para retardo Padé)
    kp_imc = T_constante / (K_planta * (lambda_deseado + L_retardo))
    ti_imc = T_constante
    td_imc = (T_constante * L_retardo) / (2.0 * T_constante + L_retardo)

    return {
        "Ziegler_Nichols_FOPDT": {
            "Kp": round(kp_zn, 4), "Ti_s": round(ti_zn, 4), "Td_s": round(td_zn, 4)
        },
```

```
        "Cohen_Coon": {
            "Kp": round(kp_cc, 4), "Ti_s": round(ti_cc, 4), "Td_s": round(td_cc, 4)
        },
        "IMC_Lambda_Robusto": {
            "Kp": round(kp_imc, 4), "Ti_s": round(ti_imc, 4), "Td_s": round(td_imc, 4)
        },
    }


def simular_lazo_pid_discreto_tustin(
    kp: float, ti: float, td: float, n_filt: float, ts: float, pasos: int = 800
) -> Dict[str, float]:
    """
    Simula una planta mecatrónica de 2.º orden controlada por un PID discreto
    discretizado por Tustin con Anti-Windup y calcula tr, Mp, ts, IAE, ISE, ITAE.
    """
    ki = kp / ti if ti > 1e-6 else 0.0
    kd = kp * td
    tau_d = (td / n_filt) if n_filt > 0 else 0.0
    denom_d = 2.0 * tau_d + ts
    alpha_d = (2.0 * tau_d - ts) / denom_d
    beta_d  = (2.0 * kd) / denom_d

    # Estado de la planta (wn = 8 rad/s, zeta = 0.45, K = 1.0)
    wn = 8.0
    zeta = 0.45
    y_pos = 0.0
    y_vel = 0.0
    sp = 1.0

    integral = 0.0
    d_prev = 0.0
    y_prev = 0.0
    e_prev = 0.0

    iae = 0.0
    ise = 0.0
    itae = 0.0
    itse = 0.0
    y_max = 0.0
    t_pico = 0.0
    t_10 = None
    t_90 = None
    t_estable = 0.0

    for k in range(pasos):
        t = k * ts
        error = sp - y_pos

        # Acumulación de índices integrales de desempeño
        abs_e = abs(error)
        iae  += abs_e * ts
        ise  += (error ** 2) * ts
        itae += (t * abs_e) * ts
        itse += (t * (error ** 2)) * ts

        # Algoritmo PID Discreto con filtro derivativo sobre -PV
        p_term = kp * error
        d_term = alpha_d * d_prev - beta_d * (y_pos - y_prev)
        i_cand = integral + 0.5 * ki * ts * (error + e_prev)

        u_unsat = p_term + i_cand + d_term
        u_cmd = max(U_MIN, min(U_MAX, u_unsat))

        # Anti-Windup condicional (Clamping)
        if u_unsat == u_cmd or (u_unsat > U_MAX and error < 0) or (u_unsat < U_MIN and error > 0):
            integral = i_cand
```

```
        d_prev = d_term
        y_prev = y_pos
        e_prev = error

        # Integración de la planta física (Euler semi-implícito)
        acel = (wn ** 2) * u_cmd - 2.0 * zeta * wn * y_vel - (wn ** 2) * y_pos
        y_vel += acel * ts
        y_pos += y_vel * ts

        if y_pos >= 0.10 * sp and t_10 is None:
            t_10 = t
        if y_pos >= 0.90 * sp and t_90 is None:
            t_90 = t
        if y_pos > y_max:
            y_max = y_pos
            t_pico = t
        if abs(sp - y_pos) > 0.02 * sp:
            t_estable = t

    tr = (t_90 - t_10) if (t_10 is not None and t_90 is not None) else 0.0
    mp_pct = max(0.0, (y_max - sp) / sp * 100.0)

    return {
        "tiempo_subida_tr_s": round(tr, 4),
        "tiempo_pico_tp_s": round(t_pico, 4),
        "sobreimpulso_Mp_pct": round(mp_pct, 2),
        "tiempo_establecimiento_ts_2pct_s": round(t_estable, 4),
        "IAE": round(iae, 4),
        "ISE": round(ise, 4),
        "ITAE": round(itae, 4),
        "ITSE": round(itse, 4),
    }


def crc16_ccitt(data: bytes) -> int:
    """Calcula CRC-16-CCITT (0x1021, init 0xFFFF) para enlace serie SBC <-> MCU."""
    crc = 0xFFFF
    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
            if crc & 0x8000:
                crc = ((crc << 1) ^ 0x1021) & 0xFFFF
            else:
                crc = (crc << 1) & 0xFFFF
    return crc


def empaquetar_trama_consigna_sbc(comando: int, setpoint_float: float) -> bytes:
    """Construye trama binaria: [0xAA, 0x55, CMD, LEN=4, float32_LE, CRC16_BE]."""
    cabecera_y_payload = struct.pack("<BBBf", 0xAA, 0x55, comando, 4, setpoint_float)
    crc = crc16_ccitt(cabecera_y_payload)
    return cabecera_y_payload + struct.pack(">H", crc)


if __name__ == "__main__":
    print("=" * 78)
    print(" CURSO: CONTROLADORES EN MECATRÓNICA — VIDEO 03")
    print(" Enfoque:", "Arquitectura Harvard RISC (ARM Cortex-M), FPU/DSP, Mapa de Memoria, Registros SFR, Reloj
PLL y Timers PWM")
    print("=" * 78)

    tablas = sintonizar_pid_ziegler_nichols_y_lambda(
        K_planta=1.25, L_retardo=0.15, T_constante=0.85, lambda_deseado=0.35
    )
    print("\n[1] Sintonización Comparativa (Ziegler-Nichols, Cohen-Coon e IMC Lambda):")
    for metodo, gan in tablas.items():
        print(f"    {metodo:24s} -> Kp = {gan['Kp']:6.3f} | Ti = {gan['Ti_s']:6.3f} s | Td = {gan['Td_s']:6.3f}
s")
```

```
metricas = simular_lazo_pid_discreto_tustin(KP_NOMINAL, TI_NOMINAL, TD_NOMINAL, N_FILTRO, 0.005)
print("\n[2] Desempeño Temporal e Índices Integrales (IAE, ISE, ITAE, ITSE):")
for k_m, v_m in metricas.items():
    print(f"    {k_m:34s} : {v_m}")

trama = empaquetar_trama_consigna_sbc(comando=0x01, setpoint_float=1.65)
print("\n[3] Trama Binaria Híbrida SBC -> MCU con CRC-16-CCITT:", trama.hex(" ").upper())
```

6. Programa PLC IEC 61131-3 (ST + IL): Enclavamiento, GRAFCET y Escalado 4-20 mA (automatizacion_plc_iec61131_video_03.st)

```
(* ================================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 03: Video 03 — Parte 2A: Microcontroladores y Microprocesadores — Arquitectura Interna y Fundamentos de
Hardware
 * Archivo: automatizacion_plc_iec61131_video_03.st
 * --------------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) + Equivalencia en Lista de
 *           Instrucciones (IL) y Peldaños de Escalera (Ladder Diagram)
 * Descripción Técnica:
 *   Bloque de Función Industrial (FB_ControladorCeldaMecatronica_V03) que
 *   integra el Ciclo de Barrido (Scan), circuito de enclavamiento prioritario
 *   al paro (cableado NC), secuenciador GRAFCET de 4 etapas, temporizador TON,
 *   contador de lotes CTU, registro de corrimiento de rechazo de calidad SHRB,
 *   escalado analógico 4-20 mA (NORM_X / SCALE_X con detección de rotura de
 *   hilo < 3.6 mA) y algoritmo PID discreto con Anti-Windup.
 * ================================================================================ *)

FUNCTION_BLOCK FB_ControladorCeldaMecatronica_V03
VAR_INPUT
    (* Entradas Digitales de la Tabla de Imagen de Proceso PII (%IX) *)
    xPulsadorMarcha_NO   : BOOL; (* %IX0.0 - Contacto Normalmente Abierto *)
    xPulsadorParo_NC     : BOOL; (* %IX0.1 - Cableado Físicamente NC en Campo *)
    xReleTermico_NC      : BOOL; (* %IX0.2 - Guardamotor NC (1 = Sin Falla) *)
    xSensorPiezaEntrada  : BOOL; (* %IX0.3 - Sensor Inductivo PNP 24V DC *)
    xCamaraPiezaDefecto  : BOOL; (* %IX0.4 - 1 = Envase Defectuoso Detectado *)
    xPulsoEncoderBanda   : BOOL; (* %IX0.5 - Reloj de Desplazamiento de Banda *)
    xResetContadores     : BOOL; (* %IX0.6 - Reinicio Manual de Lote *)

    (* Entradas Analógicas Crudas del Convertidor ADC (%IW) *)
    iEntradaAnalogicaRaw : INT;  (* %IW64  - Lazo 4-20 mA: 0 a 27648 cuentas *)
    rSetpointIngenieria  : REAL := 2.0;
END_VAR

VAR_OUTPUT
    (* Salidas Digitales hacia la Tabla de Imagen de Salidas PIQ (%QX) *)
    xMotorBandaContactor : BOOL; (* %QX0.0 - Contactor Principal Enclavado *)
    xValvulaDosificadora : BOOL; (* %QX0.1 - Solenoide 24V DC con Diodo Flyback *)
    xCilindroExpulsor    : BOOL; (* %QX0.2 - Rechazo en Posición Bit 5 del SHRB *)
    xAlarmaRoturaCable   : BOOL; (* %QX0.3 - Corriente < 3.6 mA o Paro Activo *)
    xLoteCompletado      : BOOL; (* %QX0.4 - Salida Q del Contador CTU *)

    (* Salida Analógica Escalada y Variables de Diagnóstico HMI *)
    iSalidaAnalogicaRaw  : INT;  (* %QW80  - Mando 0..27648 hacia Variador/SSR *)
    rVariableProcesoPV   : REAL; (* Variable física escalada en unidades de ing. *)
    rSalidaControlPID    : REAL; (* Esfuerzo de control 0.0 a 100.0 % *)
    iEstadoGrafcet       : INT;  (* Etapa activa del GRAFCET: 0, 10, 20, 30 *)
END_VAR

VAR
    (* Instancias de Temporizadores y Contadores Estándar IEC 61131-3 *)
    tonDosificado        : TON;
    ctuContadorLotes     : CTU;

    (* Variables Internas de Escalado NORM_X / SCALE_X *)
    rValorNormalizado    : REAL;
    rMinIngenieria       : REAL := 0.0;
    rMaxIngenieria       : REAL := 3.3;

    (* Registro de Corrimiento de 16 Bits (Shift Register para Rechazo) *)
    wRegistroCorrimiento : WORD := 16#0000;
    xPulsoEncoderPrev    : BOOL := FALSE;
    xFlancoEncoder       : BOOL := FALSE;

    (* Memoria Retentiva del Bloque PID Discreto *)
    rKp                  : REAL := 5.000;
    rTi_s                : REAL := 0.250;
    rTd_s                : REAL := 0.040;
```

```
    rTs_s                : REAL := 0.0005;
    rIntegralAcum        : REAL := 0.0;
    rErrorPrev           : REAL := 0.0;
    rPvPrev              : REAL := 0.0;
END_VAR

(* ================================================================================
 * 1. EQUIVALENCIA EN LISTA DE INSTRUCCIONES (IL) DEL PELDAÑO DE ENCLAVAMIENTO:
 *      LD     xPulsadorMarcha_NO
 *      OR     xMotorBandaContactor
 *      AND    xPulsadorParo_NC
 *      AND    xReleTermico_NC
 *      ANDN   xAlarmaRoturaCable
 *      ST     xMotorBandaContactor
 * ================================================================================ *)
xMotorBandaContactor := (xPulsadorMarcha_NO OR xMotorBandaContactor)
                        AND xPulsadorParo_NC
                        AND xReleTermico_NC
                        AND NOT xAlarmaRoturaCable;

(* ================================================================================
 * 2. NORMALIZACIÓN (NORM_X) Y ESCALADO LINEAL (SCALE_X) DE LAZO 4-20 mA
 *    Detección de rotura de hilo si iEntradaAnalogicaRaw < -1728 (< 3.0 mA)
 * ================================================================================ *)
IF (iEntradaAnalogicaRaw < -1728) OR (NOT xPulsadorParo_NC) OR (NOT xReleTermico_NC) THEN
    xAlarmaRoturaCable := TRUE;
    rValorNormalizado  := 0.0;
ELSE
    xAlarmaRoturaCable := FALSE;
    (* NORM_X: Convierte 0..27648 cuentas a rango flotante 0.0..1.0 *)
    rValorNormalizado  := INT_TO_REAL(iEntradaAnalogicaRaw) / 27648.0;
    IF rValorNormalizado < 0.0 THEN rValorNormalizado := 0.0; END_IF;
    IF rValorNormalizado > 1.0 THEN rValorNormalizado := 1.0; END_IF;
END_IF;

(* SCALE_X: Convierte 0.0..1.0 a Unidades de Ingeniería [rMinIngenieria .. rMaxIngenieria] *)
rVariableProcesoPV := rMinIngenieria + rValorNormalizado * (rMaxIngenieria - rMinIngenieria);

(* ================================================================================
 * 3. REGISTRO DE CORRIMIENTO DE BITS (SHRB / BSL) SINCRONIZADO CON ENCODER
 * ================================================================================ *)
xFlancoEncoder    := xPulsoEncoderBanda AND NOT xPulsoEncoderPrev;
xPulsoEncoderPrev := xPulsoEncoderBanda;

IF xFlancoEncoder AND xMotorBandaContactor THEN
    wRegistroCorrimiento := SHL(IN := wRegistroCorrimiento, N := 1);
    IF xCamaraPiezaDefecto THEN
        wRegistroCorrimiento := wRegistroCorrimiento OR 16#0001;
    END_IF;
END_IF;

(* Si el bit desplazado llega a la estación 5 (bit 4, máscara 16#0010), activa expulsor *)
xCilindroExpulsor := (wRegistroCorrimiento AND 16#0010) <> 16#0000;

(* ================================================================================
 * 4. SECUENCIADOR GRAFCET / SFC DE 4 ETAPAS CON TEMPORIZADOR TON Y CONTADOR CTU
 * ================================================================================ *)
IF NOT xMotorBandaContactor THEN
    iEstadoGrafcet := 0;
END_IF;

CASE iEstadoGrafcet OF
    0: (* Etapa 0: Reposo esperando pieza en estación *)
        xValvulaDosificadora := FALSE;
        IF xMotorBandaContactor AND xSensorPiezaEntrada THEN
            iEstadoGrafcet := 10;
        END_IF;
```

```
    10: (* Etapa 10: Dosificación temporizada con bloque TON *)
        xValvulaDosificadora := TRUE;
        IF tonDosificado.Q THEN
            xValvulaDosificadora := FALSE;
            iEstadoGrafcet       := 20;
        END_IF;

    20: (* Etapa 20: Incremento de contador de lote y espera de liberación *)
        xValvulaDosificadora := FALSE;
        IF NOT xSensorPiezaEntrada THEN
            iEstadoGrafcet := 0;
        END_IF;
END_CASE;

tonDosificado(IN := (iEstadoGrafcet = 10), PT := T#1500MS);
ctuContadorLotes(CU := (iEstadoGrafcet = 20), R := xResetContadores, PV := 50);
xLoteCompletado := ctuContadorLotes.Q;

(* ================================================================================
 * 5. BLOQUE TECNOLÓGICO PID DISCRETO CON ANTI-WINDUP Y DES-ESCALADO ANALÓGICO
 * ================================================================================ *)
IF xMotorBandaContactor AND NOT xAlarmaRoturaCable THEN
    VAR
        rErrorActual : REAL;
        rTerminoP    : REAL;
        rTerminoD    : REAL;
        rCandI       : REAL;
        rSumaUnsat   : REAL;
    END_VAR
    rErrorActual := rSetpointIngenieria - rVariableProcesoPV;
    rTerminoP    := rKp * rErrorActual;
    rTerminoD    := -rKp * rTd_s * (rVariableProcesoPV - rPvPrev) / rTs_s;
    rCandI       := rIntegralAcum + (rKp / rTi_s) * rTs_s * 0.5 * (rErrorActual + rErrorPrev);

    rSumaUnsat   := rTerminoP + rCandI + rTerminoD;
    IF rSumaUnsat > 100.0 THEN
        rSalidaControlPID := 100.0;
    ELSIF rSumaUnsat < 0.0 THEN
        rSalidaControlPID := 0.0;
    ELSE
        rSalidaControlPID := rSumaUnsat;
        rIntegralAcum     := rCandI;
    END_IF;

    rErrorPrev := rErrorActual;
    rPvPrev    := rVariableProcesoPV;
ELSE
    rSalidaControlPID := 0.0;
    rIntegralAcum     := 0.0;
END_IF;

(* Conversión de 0.0..100.0 % a 0..27648 cuentas crudas de salida analógica %QW80 *)
iSalidaAnalogicaRaw := REAL_TO_INT((rSalidaControlPID / 100.0) * 27648.0);

END_FUNCTION_BLOCK
```