CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA

Video 05 — Parte 3A: Plataformas Electrónicas — Ecosistema Arduino y Alternativas de Tiempo Real (ESP32, STM32, RP2040, Teensy) — Abstracción HAL vs. Registros, Evolución Uno R3/R4/Portenta, ESP32 MCPWM, STM32 Cube, RP2040 PIO y Teensy 4.1

Parte del Curso: Parte 3A — Plataformas Electrónicas: Ecosistema Arduino y Alternativas de Tiempo Real
Arquitectura / Controlador Estudiado: Comparativa e Implementación en Arduino (Uno R3/R4/Portenta), ESP32-S3 (MCPWM/PCNT), STM32, RP2040 (PIO) y Teensy 4.1

```
Sincronismo Hardware: Center-Aligned PWM (TRGO) --> Conversión ADC --> Transferencia DMA --> ISR Lazo PID (Jitter < 50 ns)
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Sobrecarga de digitalWrite (~50 ciclos) frente a manipulación directa de registros de puerto (1-2 ciclos); evolución desde ATmega328P (8-bit 16 MHz) hasta Uno R4 (Cortex-M4 48 MHz) y Portenta H7 / Opta.
- ESP32 y ESP32-S3: doble núcleo Xtensa a 240 MHz con FreeRTOS (Núcleo 0 para Wi-Fi/MQTT y Núcleo 1 para control en tiempo real), periférico MCPWM para puentes H y contador de cuadratura PCNT por hardware.
- Familia STM32 (Cortex-M0+/M4/M7 con STM32CubeMX y librerías HAL/LL), Raspberry Pi Pico RP2040/RP2350 con máquinas de estado de E/S programables (PIO) deterministas al ciclo, y Teensy 4.1 (Cortex-M7 superscalar a 600 MHz con FPU de 64 bits y memoria TCM).
- Disparo de conversión ADC sincronizado con el centro del pulso PWM (Center-Aligned Trigger) para muestrear corriente libre de ruido de conmutación, y gestión reproducible con PlatformIO y CMake.

2. Parámetros Nominales de Configuración del Controlador (Video 05)

Parámetro del Controlador / Interfase	Símbolo	Valor Nominal	Función en el Lazo de Control / Automatización
Ganancia Proporcional	`Kp (PB = 100/Kp)`	6	Esfuerzo inmediato proporcional al error actual e(t)
Tiempo Integral	`Ti [s]`	0.2 s	Elimina el error estacionario (Offset) acumulando la integral del error
Tiempo Derivativo y Filtro Divisor	`Td [s] / N`	Td = 0.03 s, N = 20	Anticipa la tendencia sobre -PV y atenúa el ruido de alta frecuencia
Periodo de Muestreo en Tiempo Real	`Ts [ms]`	0.1 ms	Intervalo determinista de conversión ADC, cálculo discreto y actualización PWM
Límites de Saturación Anti-Windup	`[u_min, u_max]`	[0, 4095]	Cota física del actuador para integración condicional (Clamping)
Rango Escalado de Ingeniería (SCALE_X)	`[PV_min, PV_max]`	[0, 5000]	Conversión lineal desde cuentas crudas del ADC (0..1023 o 0..27648)

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: El Ecosistema Arduino en Ingeniería Mecatrónica: HAL frente a Bare-Metal

Estructura setup/loop, Sobrecarga de digitalWrite (~50 ciclos) frente a Acceso Directo a Puertos (1-2 ciclos)

- **Turno 1 (Masculina):** Iniciamos la Parte tres del curso analizando las plataformas electrónicas sobre las que se materializan los controladores mecatrónicos, comenzando por el ecosistema que democratizó el prototipado mundial: la plataforma Arduino.
- **Turno 2 (Femenina):** El éxito de Arduino radica en su capa de abstracción de hardware, que oculta la complejidad de los registros detrás de dos funciones universales: setup para inicializar periféricos una sola vez, y loop para el ciclo infinito de ejecución.
- **Turno 3 (Masculina):** Sin embargo, como ingenieros debemos medir el costo de esa comodidad: en un microcontrolador AVR a dieciséis megahercios, invocar la función digitalWrite tarda más de cincuenta ciclos de reloj, unos tres punto cuatro microsegundos, porque debe buscar en tablas de memoria Flash a qué puerto y bit corresponde el número de pin.
- **Turno 4 (Femenina):** Si en cambio escribimos directamente sobre el registro de salida del puerto B mediante una máscara de bits en C embebido, la conmutación del pin se ejecuta en un único ciclo de reloj de sesenta y dos punto cinco nanosegundos: más de cincuenta veces más rápido.

Diapositiva 02: Evolución del Hardware Arduino: De 8 Bits al Ámbito Industrial de 32 Bits

Arduino Uno R3 (ATmega328P), Mega 2560, Uno R4 (Renesas Cortex-M4) y Familia Industrial Portenta H7 y Opta

- **Turno 1 (Masculina):** Es un error común pensar que Arduino se reduce únicamente a la placa clásica de ocho bits; hoy el ecosistema abarca desde microcontroladores educativos básicos hasta controladores industriales de doble núcleo de treinta y dos bits.
- **Turno 2 (Femenina):** El clásico Arduino Uno erre tres utiliza el chip ATmega trescientos veintiocho pe de ocho bits a dieciséis megahercios con solo dos kilobytes de memoria SRAM y seis salidas P W M; mientras que el Arduino Mega veinticinco sesenta amplía los pines a cincuenta y cuatro entradas y salidas digitales y cuatro puertos serie por hardware.
- **Turno 3 (Masculina):** El salto generacional llega con el Arduino Uno erre cuatro, impulsado por un microcontrolador Renesas ARM Cortex-M4 de treinta y dos bits a cuarenta y ocho megahercios con unidad de punto flotante, treinta y dos kilobytes de SRAM, convertidor ADC de catorce bits, salida DAC analógica real de doce bits y controlador de bus CAN integrado.
- **Turno 4 (Femenina):** Y para el entorno fabril, la línea Arduino Pro ofrece la placa Portenta H siete con un microcontrolador STM32 de doble núcleo Cortex-M7 a cuatrocientos ochenta megahercios más Cortex-M4, y el micro-PLC industrial Arduino Opta para montaje en riel DIN con entradas de veinticuatro voltios.

Diapositiva 03: Alternativa 1 — Espressif ESP32 y ESP32-S3 en Control e IIoT

Doble Núcleo Xtensa LX6/LX7 a 240 MHz, FreeRTOS Nativo, Periférico MCPWM para Motores y Contador PCNT

- **Turno 1 (Masculina):** Cuando un proyecto mecatrónico exige alto poder de cómputo en treinta y dos bits junto con telemetría inalámbrica Wi-Fi y Bluetooth de baja energía a un costo mínimo, la plataforma reina es la familia ESP32 y ESP32-S3 de Espressif.
- **Turno 2 (Femenina):** Estos sistemas en un chip integran dos núcleos de treinta y dos bits corriendo a doscientos cuarenta megahercios con unidad de punto flotante, quinientos doce kilobytes de SRAM y el sistema operativo de tiempo real FreeRTOS ejecutándose de forma nativa.
- **Turno 3 (Masculina):** La arquitectura de doble núcleo nos permite aplicar una regla de oro en mecatrónica conectada: anclamos toda la pila de red Wi-Fi, MQTT y servidor web al Núcleo cero, y reservamos en exclusiva el Núcleo uno para ejecutar nuestro lazo de control pe i de sin interrupciones de red.
- **Turno 4 (Femenina):** Además, el ESP32 incluye dos periféricos de hardware extraordinarios para robótica: el bloque M C P W M dedicado al control de motores con inserción de tiempo muerto y frenado por falla en hardware, y el módulo contador de pulsos P C N T que decodifica encoders en cuadratura de forma autónoma.

Diapositiva 04: Alternativa 2 — Familia STM32 (ARM Cortex-M0+, M4 y M7)

El Estándar Industrial en Control de Movimiento, STM32CubeMX, Aceleradores CORDIC/FMAC y Drivers HAL vs. LL

- **Turno 1 (Masculina):** Si abrimos el gabinete de un servodrive industrial moderno, un dron profesional, un brazo colaborativo o un instrumento médico, el microcontrolador que encontraremos en su interior es casi con seguridad un miembro de la familia STM32 de STMicroelectronics.
- **Turno 2 (Femenina):** El ecosistema STM32 cubre más de mil variantes basadas en núcleos ARM Cortex-M: desde la serie económica F uno y G cero, pasando por la serie F cuatro y G cuatro con punto flotante y aceleradores matemáticos CORDIC para control vectorial de motores trifásicos, hasta la serie H siete que supera los cuatrocientos ochenta megahercios.
- **Turno 3 (Masculina):** El desarrollo profesional sobre placas Nucleo y Discovery se apoya en la herramienta gráfica STM32CubeMX, donde configuramos visualmente el árbol de reloj P L L, la asignación de pines, los canales D M A y los temporizadores, generando automáticamente el código de inicialización en C.
- **Turno 4 (Femenina):** Al programar disponemos de dos niveles de librerías oficiales: las funciones H A L de alto nivel para la lógica general y las comunicaciones, y las macros de Capa Baja o Low-Layer que compilan a escrituras directas en registros para lograr latencia mínima dentro de las interrupciones de control.

Diapositiva 05: Alternativa 3 — Raspberry Pi Pico (RP2040 y RP2350) y Subsistema PIO

Doble Núcleo Simétrico y Máquinas de Estado de Entrada/Salida Programables (PIO) Deterministas al Ciclo

- **Turno 1 (Masculina):** La tercera gran alternativa moderna para controladores embebidos es la familia de microcontroladores diseñada por Raspberry Pi: el chip RP veinte cuarenta en la placa Pico original, y el RP veintitrés cincuenta con núcleo Cortex-M33 y punto flotante en la Pico dos.
- **Turno 2 (Femenina):** Además de ofrecer dos núcleos simétricos rápidos y doscientos sesenta y cuatro o quinientos veinte kilobytes de SRAM dividida en bancos independientes sin contención, estos chips introdujeron una innovación arquitectónica única en la industria: el subsistema de Entrada y Salida Programable o P I O.
- **Turno 3 (Masculina):** Los bloques P I O contienen ocho o doce máquinas de estado de hardware independientes de la CPU principal, cada una con sus propias colas FIFO y un micro-repertorio de nueve instrucciones ensamblador que se ejecutan en exactamente un ciclo de reloj con cero fluctuación temporal.
- **Turno 4 (Femenina):** Gracias a las máquinas de estado P I O podemos leer simultáneamente ocho encoders ópticos en cuadratura a decenas de megahercios, generar trenes de pulsos precisos al nanosegundo para motores paso a paso o implementar buses serie industriales a medida dejando ambos núcleos de la CPU cien por ciento libres.

Diapositiva 06: Alternativa 4 — Teensy 4.0 y 4.1 (NXP i.MX RT1062 a 600 MHz)

Procesador Crossover ARM Cortex-M7 Superscalar Dual-Issue, FPU de 64 Bits (double) y Memoria TCM

- **Turno 1 (Masculina):** Cuando diseñamos robots bípedos ágiles, plataformas Stewart de seis grados de libertad o sistemas de control acústico activo que necesitan resolver matrices cinemáticas y filtros de orden elevado a decenas de kilohercios, recurrimos a las placas Teensy cuatro punto cero y cuatro punto uno.
- **Turno 2 (Femenina):** Su corazón es el procesador Crossover NXP i punto M X R T diez sesenta y dos, basado en el núcleo ARM Cortex-M7 corriendo a una velocidad impresionante de seiscientos megahercios.
- **Turno 3 (Masculina):** A diferencia de los microcontroladores convencionales, este núcleo posee una arquitectura superscalar de doble emisión capaz de ejecutar dos instrucciones en paralelo en cada ciclo de reloj, e integra una unidad de punto flotante por hardware que acelera tanto precisión simple float de treinta y dos bits como doble precisión double de sesenta y cuatro bits.
- **Turno 4 (Femenina):** Su secreto para no sufrir cuellos de botella a seiscientos megahercios es su megabyte de memoria RAM conectado mediante buses de Memoria Estrechamente Acoplada o T C M de sesenta y cuatro bits de ancho, que alimentan instrucciones y datos al núcleo en un único ciclo sin estados de espera.

Diapositiva 07: Comparativa Cuantitativa de Plataformas MCU para Selección en Ingeniería

Frecuencia de Reloj, FPU, Resolución y Linealidad de ADC, Latencia de Interrupción, Consumo y Costo

- **Turno 1 (Masculina):** Con tantas opciones disponibles en el mercado, un ingeniero mecatrónico selecciona la plataforma de control evaluando criterios cuantitativos rigurosos frente a los requisitos del proyecto.
- **Turno 2 (Femenina):** El primer criterio crítico suele ser el convertidor analógico-digital: el Arduino Uno ofrece diez bits lentos a diez kilomuestras por segundo; el ADC interno del ESP32 es rápido pero presenta no linealidad en los extremos de tensión; mientras que un STM32F4 o G4 entrega doce a dieciséis bits de alta linealidad a varios millones de muestras por segundo.
- **Turno 3 (Masculina):** El segundo criterio es el determinismo y la capacidad matemática: para lazos pe i de simples de hasta quinientos hercios basta un microcontrolador de ocho bits o un Cortex-M0; para control de motores a veinte kilohercios elegimos STM32 o ESP32; y para cinemática densa en doble precisión elegimos Teensy cuatro punto uno o STM32H7.
- **Turno 4 (Femenina):** El tercer criterio es el nivel lógico eléctrico: solo el clásico Arduino Uno erre tres y el Uno erre cuatro operan a cinco voltios nativos; todas las plataformas modernas de treinta y dos bits —ESP32, STM32, RP2040 y Teensy cuatro— trabajan estrictamente a tres punto tres voltios y muchos de sus pines se destruyen si reciben cinco voltios directos.

Diapositiva 08: Configuración Avanzada de Periféricos en Plataformas MCU

Sincronización por Hardware entre el Centro del Pulso PWM, Conversión ADC, Transferencia DMA e ISR del PID

- **Turno 1 (Masculina):** Veamos cómo se configura profesionalmente una plataforma de treinta y dos bits como STM32 o ESP32 para ejecutar un lazo de control de corriente y velocidad de motor de precisión industrial.
- **Turno 2 (Femenina):** Cuando un puente H conmuta a veinte kilohercios, en los instantes en que los transistores MOSFET se encienden o se apagan se producen picos de ruido electromagnético en los sensores de corriente; si el ADC muestreara en ese flanco, mediría ruido puro.
- **Turno 3 (Masculina):** Para evitarlo, configuramos el temporizador P W M en modo de conteo simétrico alineado al centro y activamos su salida de disparo interno justo en el vértice de la onda portadora triangular, exactamente a mitad del pulso de encendido, cuando la corriente es estable y limpia.
- **Turno 4 (Femenina):** Ese pulso interno dispara al convertidor ADC por hardware sin intervención del software; al terminar la conversión en un microsegundo, el canal D M A deposita el valor en memoria y levanta la interrupción donde se calcula el algoritmo pe i de, logrando sincronismo perfecto ciclo a ciclo.

Diapositiva 09: Entornos Profesionales de Desarrollo y Configuración

PlatformIO sobre VS Code, CMake, STM32CubeIDE, ESP-IDF, Control de Versiones y Gestión Reproducible

- **Turno 1 (Masculina):** Aunque el entorno clásico Arduino IDE es excelente para los primeros pasos, en proyectos de ingeniería mecatrónica de mediana y gran escala necesitamos entornos de desarrollo integrados que garanticen reproducibilidad, control de versiones y depuración en hardware.
- **Turno 2 (Femenina):** El flujo multiplataforma más extendido es PlatformIO integrado sobre Visual Studio Code: mediante un archivo declarativo llamado platformio punto ini fijamos la versión exacta del compilador, la placa objetivo y las versiones exactas de cada librería para que el proyecto compile idéntico en cualquier computadora de la empresa.
- **Turno 3 (Masculina):** Cuando requerimos acceso total a las capacidades nativas de cada fabricante, empleamos los entornos oficiales basados en CMake: STM32CubeIDE para toda la familia de STMicroelectronics, y el marco oficial ESP-IDF para los procesadores de Espressif.
- **Turno 4 (Femenina):** Estos entornos profesionales integran autocompletado inteligente, inspección de registros periféricos mediante archivos de descripción S V D, análisis estático del consumo de memoria Flash y SRAM, y depuración paso a paso con sondas en circuito.

Diapositiva 10: Síntesis de Plataformas de Microcontroladores en Tiempo Real

Criterios de Arquitectura Embebida y Transición hacia las Computadoras de Placa Única (SBC) en la Parte 3B

- **Turno 1 (Masculina):** En esta Parte tres A hemos recorrido y comparado las principales plataformas de microcontroladores de tiempo real disponibles para el ingeniero mecatrónico: desde la evolución de Arduino hasta la potencia de ESP32, STM32, RP2040 y Teensy.
- **Turno 2 (Femenina):** Comprendimos cómo superar los límites de la capa de abstracción básica, cómo aprovechar núcleos duales, unidades de punto flotante y máquinas de estado P I O, y cómo sincronizar por hardware los periféricos P W M y ADC.
- **Turno 3 (Masculina):** ¿Pero qué ocurre cuando nuestra máquina mecatrónica necesita además procesar video de cámaras de alta definición, ejecutar redes neuronales de visión artificial, alojar bases de datos o desplegar una interfaz gráfica táctil avanzada?
- **Turno 4 (Femenina):** En el siguiente módulo, correspondiente a la Parte tres B, estudiaremos las Computadoras de Placa Única encabezadas por Raspberry Pi, sus alternativas como NVIDIA Jetson y BeagleBone, y la arquitectura híbrida maestro-coprocesador.

4. Firmware C++/MCU: PID Discreto Tustin, FSM y CRC-16 (controlador_tiempo_real_video_05.ino)

```
/*
 * ============================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 05: Video 05 — Parte 3A: Plataformas Electrónicas — Ecosistema Arduino y Alternativas de Tiempo Real
(ESP32, STM32, RP2040, Teensy)
 * Archivo: controlador_tiempo_real_video_05.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:
 *   Comparativa e Implementación en Arduino (Uno R3/R4/Portenta), ESP32-S3 (MCPWM/PCNT), STM32, RP2040 (PIO) y
Teensy 4.1
 * ============================================================================
 */

#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 05) ---
const float KP_GAIN      = 6.000f;
const float TI_SEC       = 0.200f;
const float TD_SEC       = 0.030f;
const float N_FILT       = 20.0f;
const float TS_SEC       = 0.0001f; // Periodo de muestreo Ts = 0.1 ms
const float U_MIN_LIM    = 0.0f;
const float U_MAX_LIM    = 4095.0f;
const float PV_MIN_ENG   = 0.0f;
const float PV_MAX_ENG   = 5000.0f;

// --- 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 05"));
  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_05.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
VIDEO 05: Video 05 — Parte 3A: Plataformas Electrónicas — Ecosistema Arduino y Alternativas de Tiempo Real (ESP32,
STM32, RP2040, Teensy)
Archivo: supervision_sbc_y_sintonizacion_video_05.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 05
KP_NOMINAL = 6
TI_NOMINAL = 0.2
TD_NOMINAL = 0.03
N_FILTRO   = 20
TS_SEG     = 0.0001
U_MIN      = 0
U_MAX      = 4095


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 05")
    print(" Enfoque:", "Comparativa e Implementación en Arduino (Uno R3/R4/Portenta), ESP32-S3 (MCPWM/PCNT),
STM32, RP2040 (PIO) y Teensy 4.1")
    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=2500)
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_05.st)

```
(* ==========================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 05: Video 05 — Parte 3A: Plataformas Electrónicas — Ecosistema Arduino y Alternativas de Tiempo Real
(ESP32, STM32, RP2040, Teensy)
 * Archivo: automatizacion_plc_iec61131_video_05.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_V05) 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_V05
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 := 3000.0;
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 := 5000.0;

    (* 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 := 6.000;
    rTi_s                : REAL := 0.200;
    rTd_s                : REAL := 0.030;
```

```
    rTs_s                : REAL := 0.0001;
    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
```