CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA

Video 04 — Parte 2B: Programación de Microcontroladores, Interrupciones, DMA, Máquinas de Estados y RTOS — Toolchain y Linker Script, Máscaras de Bits, Punteros Volatile, NVIC e ISR, DMA Ping-Pong, FSM y FreeRTOS

Parte del Curso: Parte 2B — Programación de Microcontroladores, Interrupciones, DMA, FSM y RTOS
Arquitectura / Controlador Estudiado: Firmware Bare-Metal con Máscaras de Bits, Punteros Volatile, NVIC, DMA Ping-Pong, FSM No Bloqueante y FreeRTOS

```
REG |= (1U << n) [Set] | REG &= ~(1U << n) [Clear] | #define PERIPH_REG (*(volatile uint32_t *)(ADDR))
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Toolchain de compilación cruzada y Linker Script (.ld) que posiciona .isr_vector y .text en Flash y .data, .bss y .stack en SRAM.
- Manipulación de registros con máscaras bit a bit (|=, &= ~, ^=), tipos enteros exactos de <stdint.h> y uso obligatorio del calificador volatile en registros de hardware y variables compartidas con ISR.
- Controlador de interrupciones anidadas NVIC (Preemption Priority vs. Subpriority), brevedad estricta de las ISR, secciones críticas atómicas y transferencias autónomas por DMA con doble buffer Ping-Pong.
- Diseño reactivo mediante Máquinas de Estados Finitos (FSM de Moore/Mealy) sin bloqueos delay(), multitarea determinista con FreeRTOS (vTaskDelayUntil, Queues, Semáforos, Mutex con herencia de prioridad) y punto fijo Q15/Q31.

2. Parámetros Nominales de Configuración del Controlador (Video 04)

Parámetro del Controlador / Interfase	Símbolo	Valor Nominal	Función en el Lazo de Control / Automatización
Ganancia Proporcional	`Kp (PB = 100/Kp)`	3.8	Esfuerzo inmediato proporcional al error actual e(t)
Tiempo Integral	`Ti [s]`	0.3 s	Elimina el error estacionario (Offset) acumulando la integral del error
Tiempo Derivativo y Filtro Divisor	`Td [s] / N`	Td = 0.05 s, N = 12	Anticipa la tendencia sobre -PV y atenúa el ruido de alta frecuencia
Periodo de Muestreo en Tiempo Real	`Ts [ms]`	1 ms	Intervalo determinista de conversión ADC, cálculo discreto y actualización PWM
Límites de Saturación Anti-Windup	`[u_min, u_max]`	[-100, 100]	Cota física del actuador para integración condicional (Clamping)
Rango Escalado de Ingeniería (SCALE_X)	`[PV_min, PV_max]`	[-50, 50]	Conversión lineal desde cuentas crudas del ADC (0..1023 o 0..27648)

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Cadena de Compilación Cruzada (Cross-Compilation Toolchain)

Preprocesador, Compilador, Ensamblador, Script de Enlazado (Linker Script .ld) y Secciones de Memoria

- **Turno 1 (Masculina):** Cuando programamos un microcontrolador, escribimos el código fuente en una computadora personal pero el archivo binario resultante debe ejecutarse en una arquitectura de silicio totalmente distinta: este proceso se denomina compilación cruzada o Cross-Compilation.
- **Turno 2 (Femenina):** La cadena de herramientas o Toolchain trabaja en cuatro etapas: primero el preprocesador expande las directivas include y define; segundo, el compilador traduce el código C o C más más a lenguaje ensamblador optimizado; y tercero, el ensamblador genera los archivos objeto en código máquina.
- **Turno 3 (Masculina):** La cuarta etapa es crucial en sistemas embebidos: el Enlazador o Linker toma todos los archivos objeto y, guiado por el archivo Linker Script, asigna las direcciones físicas exactas de cada función y variable dentro del mapa de memoria del chip.
- **Turno 4 (Femenina):** Así, la sección punto text con el código y la tabla de vectores de interrupción se graban en la memoria Flash; mientras que las variables inicializadas punto data y las variables en cero punto bss se ubican en la memoria SRAM por medio de la rutina de arranque antes de llamar a la función main.

Diapositiva 02: Programación en C/C++ Embebido a Nivel de Registros (Bare-Metal)

Operadores Bit a Bit (OR, AND, XOR, Desplazamiento) y Máscaras para Modificar Bits sin Alterar Vecinos

- **Turno 1 (Masculina):** En un registro de control de treinta y dos bits, cada bit individual gobierna una función física distinta del periférico; por tanto, la primera habilidad en programación Bare-Metal es modificar un bit específico sin alterar accidentalmente los otros treinta y un bits del registro.
- **Turno 2 (Femenina):** Para poner en uno un bit en la posición ene construimos una máscara desplazando el número uno sin signo ene posiciones hacia la izquierda, y aplicamos el operador OR bit a bit de asignación sobre el registro: cualquier bit en cero de la máscara deja intacto su valor original.
- **Turno 3 (Masculina):** Para limpiar o poner en cero únicamente el bit ene sin tocar los demás, invertimos todos los bits de la máscara usando el operador de complemento a uno tilde, y aplicamos el operador AND bit a bit de asignación sobre el registro.
- **Turno 4 (Femenina):** De igual forma, utilizamos el operador OR exclusivo o XOR con la máscara para invertir el estado de un pin en cada ciclo de prueba, y empleamos la operación AND dentro de una condición if para consultar si una bandera de estado de hardware se ha activado.

Diapositiva 03: El Calificador volatile, Tipos de Ancho Fijo (<stdint.h>) y Punteros a Hardware

Prevención de Optimizaciones Destructivas del Compilador y Enteros Exactos (uint8_t, uint16_t, int32_t)

- **Turno 1 (Masculina):** Uno de los errores más sutiles y frecuentes en ingeniería de firmware ocurre cuando un programa funciona sin optimización pero falla misteriosamente al activar la optimización de velocidad del compilador: la causa casi siempre es omitir la palabra clave volatile.
- **Turno 2 (Femenina):** Un compilador estándar asume que una variable en memoria solo puede cambiar si el propio hilo de código la escribe; si hacemos un bucle esperando a que cambie un registro de estado de un sensor o una variable modificada en una interrupción, el compilador lee el valor una sola vez en un registro interno y crea un bucle infinito.
- **Turno 3 (Masculina):** Al declarar ese puntero a registro o variable compartida con el calificador volatile, le ordenamos al compilador que desactive toda caché en registros y realice obligatoriamente la lectura o escritura física en el bus de memoria en cada línea del programa.
- **Turno 4 (Femenina):** Junto con volatile, la norma de programación embebida segura prohíbe usar el tipo int genérico cuyo tamaño cambia entre arquitecturas de ocho y treinta y dos bits, exigiendo incluir la cabecera stdint punto hache y utilizar tipos de ancho fijo exacto como uint ocho te, int dieciséis te y uint treinta y dos te.

Diapositiva 04: Sistema de Interrupciones (ISR) y Controlador Anidado NVIC

Tabla de Vectores, Latencia Determinista de Entrada, Apilado de Contexto y Prioridades de Preempción

- **Turno 1 (Masculina):** En un sistema mecatrónico en tiempo real no podemos permitirnos revisar por encuesta si ha llegado un pulso de encoder o si se ha disparado una alarma de sobrecorriente; necesitamos que el hardware interrumpa de inmediato a la CPU mediante una Solicitud de Interrupción.
- **Turno 2 (Femenina):** Cuando ocurre el evento físico, el Controlador Vectorizado de Interrupciones Anidadas, abreviado N V I C, suspende la ejecución del programa principal al finalizar la instrucción actual, guarda automáticamente ocho registros de trabajo en la Pila y salta a la dirección guardada en la Tabla de Vectores.
- **Turno 3 (Masculina):** En los núcleos ARM Cortex-M este proceso de salvado de contexto se realiza íntegramente por hardware en tan solo doce ciclos de reloj deterministas, ejecutando de inmediato la Rutina de Servicio de Interrupción o I S R correspondiente.
- **Turno 4 (Femenina):** El controlador N V I C permite asignar a cada periférico una Prioridad de Preempción y una Subprioridad: si mientras se atiende una interrupción lenta de comunicación UART ocurre una interrupción de prioridad superior del lazo de corriente del motor, el hardware anida y ejecuta primero la urgente.

Diapositiva 05: Reglas de Diseño de Rutinas de Interrupción (ISR) y Secciones Críticas

Brevedad Estricta, Prohibición de Bloqueos, Banderas Volátiles, Ring Buffers y Atomicidad de Datos

- **Turno 1 (Masculina):** Dado que una rutina de servicio de interrupción congela el flujo normal del sistema y puede retrasar otras interrupciones, todo diseño profesional de firmware se rige por reglas estrictas de higiene dentro de las I S R.
- **Turno 2 (Femenina):** Primera regla: una interrupción debe ser ultracorta y ejecutarse en pocos microsegundos; está terminantemente prohibido invocar funciones de espera bloqueante como delay, realizar impresiones lentas por puerto serie o ejecutar cálculos pesados no deterministas dentro de ella.
- **Turno 3 (Masculina):** El patrón correcto consiste en que la interrupción solo lea el registro de datos del periférico, lo deposite en un buffer circular en memoria, levante una bandera booleana volatile para avisar al lazo principal y limpie la bandera de interrupción del hardware antes de salir.
- **Turno 4 (Femenina):** Segunda regla: cuidado con la atomicidad; si el programa principal lee una estructura o variable de sesenta y cuatro bits justo cuando una interrupción la modifica a mitad de camino, leerá un dato corrupto; para evitarlo envolvemos esa lectura en una Sección Crítica deshabilitando las interrupciones durante dos o tres instrucciones.

Diapositiva 06: Acceso Directo a Memoria (DMA) y Doble Buffer (Ping-Pong)

Transferencias Periférico-Memoria sin Intervención de la CPU para Adquisición y Telemetría Continua

- **Turno 1 (Masculina):** Imagina que necesitamos muestrear tres sensores de corriente y vibración a cien kilohercios: si generáramos cien mil interrupciones por segundo solo para copiar cada número del convertidor ADC a un arreglo en memoria, la CPU gastaría casi todo su tiempo entrando y saliendo de interrupciones.
- **Turno 2 (Femenina):** Para liberar por completo al procesador de esa tarea mecánica de copia de bytes existe el controlador de Acceso Directo a Memoria, abreviado D M A: un coprocesador de transferencia de datos conectado directamente a la matriz de buses del microcontrolador.
- **Turno 3 (Masculina):** Configuramos en el canal D M A la dirección del registro de datos del periférico origen, la dirección del arreglo en memoria SRAM destino y el número de muestras a transferir; a partir de ese instante, cada vez que el ADC termina una conversión, el D M A mueve el dato en silencio sin interrumpir a la CPU.
- **Turno 4 (Femenina):** En procesamiento continuo de señales activamos el modo circular con Doble Buffer o esquema Ping-Pong: el D M A avisa con una interrupción cuando llena la primera mitad del arreglo para que la CPU procese ese bloque con calma mientras el D M A sigue llenando la segunda mitad sin perder ni una muestra.

Diapositiva 07: Diseño de Firmware Reactivo con Máquinas de Estados Finitos (FSM)

Erradicación de Retardos Bloqueantes (delay), Modelos de Moore y Mealy y Tablas de Transición

- **Turno 1 (Masculina):** En el nivel de aplicación del firmware mecatrónico, el peor enemigo de la seguridad y la capacidad de respuesta es programar secuencias usando retardos bloqueantes: mientras el procesador está atrapado en un delay de dos segundos, es completamente ciego a un botón de paro de emergencia o a un final de carrera.
- **Turno 2 (Femenina):** La arquitectura estándar para construir firmware cien por ciento reactivo y no bloqueante es la Máquina de Estados Finitos, abreviada F S M, combinada con marcas de tiempo leídas de un contador libre en milisegundos o microsegundos.
- **Turno 3 (Masculina):** Definimos un tipo enumerado con todos los estados operativos de la máquina —por ejemplo, Reposo, Referenciado Homing, Movimiento Rápido, Prensado Controlado y Falla de Seguridad— y en cada pasada del bucle principal ejecutamos en microsegundos únicamente la lógica del estado activo.
- **Turno 4 (Femenina):** En una máquina de Moore las salidas de los actuadores dependen exclusivamente del estado actual, garantizando un comportamiento predecible y seguro; mientras que las transiciones hacia el siguiente estado se disparan al cumplirse eventos de sensores o al superarse diferencias de tiempo sin bloquear jamás al procesador.

Diapositiva 08: Sistemas Operativos de Tiempo Real (RTOS / FreeRTOS)

Planificador Preemptivo por Prioridades, Tareas Periódicas, Colas (Queues), Semáforos y Mutex con Herencia

- **Turno 1 (Masculina):** Cuando un controlador mecatrónico debe ejecutar simultáneamente un lazo pe i de a un kilohercio, atender un bus CAN, actualizar una pantalla gráfica y gestionar telemetría por red, un único bucle principal se vuelve difícil de mantener y de garantizar en tiempos.
- **Turno 2 (Femenina):** En ese punto incorporamos un Sistema Operativo de Tiempo Real o R T O S, siendo FreeRTOS el estándar abierto de la industria: un kernel ligero que divide la aplicación en Tareas independientes, cada una con su propia pila de memoria y su nivel de prioridad.
- **Turno 3 (Masculina):** El Planificador Preemptivo del R T O S garantiza por diseño que siempre estará ejecutándose la tarea de mayor prioridad que esté lista; para el lazo de control usamos la función de retardo hasta instante exacto, que despierta a la tarea de control cada milisegundo sin deriva acumulativa.
- **Turno 4 (Femenina):** Para comunicar tareas sin corromper datos nunca usamos variables globales sueltas: sincronizamos interrupciones con tareas mediante Semáforos Binarios, enviamos mediciones de sensores mediante Colas seguras por copia, y protegemos buses compartidos como I dos C mediante objetos Mutex con herencia de prioridad.

Diapositiva 09: Aritmética de Punto Fijo (Q15 / Q31) y Depuración por Hardware (SWD / JTAG)

Optimización Determinista del Lazo de Control, Tiempo de Ejecución en el Peor Caso (WCET) y Trazado en Silicio

- **Turno 1 (Masculina):** ¿Cómo implementamos un controlador pe i de o un filtro digital de alta velocidad cuando diseñamos con un microcontrolador industrial de bajo costo que carece de unidad de punto flotante por hardware? Utilizando aritmética de Punto Fijo en formato cu quince o cu treinta y uno.
- **Turno 2 (Femenina):** En la aritmética de punto fijo escalamos las variables reales multiplicándolas por una potencia de dos, como dos elevado a quince, almacenándolas en enteros de dieciséis o treinta y dos bits: así la A L U realiza sumas y multiplicaciones fraccionarias a velocidad de entero usando simples desplazamientos de bits.
- **Turno 3 (Masculina):** Finalmente, para verificar y depurar nuestro firmware en la placa real no dependemos de imprimir mensajes lentos por puerto serie, sino de las interfaces de depuración en hardware JTAG y Serial Wire Debug, abreviada S W D.
- **Turno 4 (Femenina):** Conectando una sonda de depuración por dos hilos dedicados podemos detener el procesador en puntos de ruptura en hardware, inspeccionar registros periféricos y memoria en vivo, y medir con el contador de ciclos del núcleo el Tiempo de Ejecución en el Peor Caso de nuestra rutina de control.

Diapositiva 10: Síntesis de Programación Embebida de Tiempo Real

Arquitectura de Firmware Industrial Confiable y Transición a las Plataformas Electrónicas de la Parte 3A

- **Turno 1 (Masculina):** Concluimos la Parte dos del curso habiendo integrado la arquitectura interna de los microcontroladores con las técnicas profesionales de programación en C embebido y sistemas operativos de tiempo real.
- **Turno 2 (Femenina):** Dominamos el uso de máscaras de bits a nivel de registros, el calificador obligatorio volatile, las interrupciones anidadas rápidas, las transferencias autónomas por D M A, las máquinas de estados no bloqueantes y la multitarea determinista con FreeRTOS.
- **Turno 3 (Masculina):** Con estas herramientas de software y hardware, el siguiente paso natural es evaluar sobre qué placas y familias comerciales de microcontroladores construiremos nuestros proyectos y productos industriales.
- **Turno 4 (Femenina):** En la siguiente parte del curso analizaremos y compararemos cuantitativamente el ecosistema Arduino frente a las plataformas de tiempo real más potentes del mercado: ESP32, STM32, RP2040 y Teensy.

4. Firmware C++/MCU: PID Discreto Tustin, FSM y CRC-16 (controlador_tiempo_real_video_04.ino)

```
/*
 * ============================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 04: Video 04 — Parte 2B: Programación de Microcontroladores, Interrupciones, DMA, Máquinas de Estados y RTOS
 * Archivo: controlador_tiempo_real_video_04.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:
 *   Firmware Bare-Metal con Máscaras de Bits, Punteros Volatile, NVIC, DMA Ping-Pong, FSM No Bloqueante y FreeRTOS
 * ============================================================================
 */

#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 04) ---
const float KP_GAIN      = 3.800f;
const float TI_SEC       = 0.300f;
const float TD_SEC       = 0.050f;
const float N_FILT       = 12.0f;
const float TS_SEC       = 0.0010f; // Periodo de muestreo Ts = 1 ms
const float U_MIN_LIM    = -100.0f;
const float U_MAX_LIM    = 100.0f;
const float PV_MIN_ENG   = -50.0f;
const float PV_MAX_ENG   = 50.0f;

// --- 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 04"));
  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_04.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
VIDEO 04: Video 04 — Parte 2B: Programación de Microcontroladores, Interrupciones, DMA, Máquinas de Estados y RTOS
Archivo: supervision_sbc_y_sintonizacion_video_04.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 04
KP_NOMINAL = 3.8
TI_NOMINAL = 0.3
TD_NOMINAL = 0.05
N_FILTRO   = 12
TS_SEG     = 0.0010
U_MIN      = -100
U_MAX      = 100


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 04")
    print(" Enfoque:", "Firmware Bare-Metal con Máscaras de Bits, Punteros Volatile, NVIC, DMA Ping-Pong, FSM No
Bloqueante y FreeRTOS")
    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=25)
    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_04.st)

```
(* ================================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 04: Video 04 — Parte 2B: Programación de Microcontroladores, Interrupciones, DMA, Máquinas de Estados y
RTOS
 * Archivo: automatizacion_plc_iec61131_video_04.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_V04) 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_V04
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 := 10.0;
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 := -50.0;
    rMaxIngenieria       : REAL := 50.0;

    (* 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 := 3.800;
    rTi_s                : REAL := 0.300;
    rTd_s                : REAL := 0.050;
```

```
    rTs_s                : REAL := 0.0010;
    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
```