CURSO COMPLETO: FUNDAMENTOS DE ELECTRÓNICA

Video 09 — Parte 5A: Representaciones Digitales, Lógica Combinacional, Diagramas de Tiempo, Álgebra de Boole y Diseño de Redes Lógicas — Representaciones Digitales, Clases de Lógica, Diagramas de Tiempo, Álgebra de Boole, Diseño Paso a Paso de Redes Lógicas y Tablas de Verdad

Parte del Curso: Parte 5A — Circuitos Digitales: Representaciones, Lógica Combinacional, Diagramas de Tiempo, Álgebra de Boole y Diseño de Redes
Arquitectura / Controlador Estudiado: Niveles Lógicos y Margen de Ruido, Binario/Hex/Gray, Compuertas, Timing Diagrams, Álgebra de Boole, Diseño en 5 Pasos y Tablas de Verdad

```
NM_H = V_OH(min) - V_IH(min) | De Morgan: ~(A·B) = ~A + ~B , ~(A+B) = ~A·~B | SOP = Σ m_k | POS = Π M_k
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Representaciones digitales: niveles de tensión lógica, margen de ruido, sistemas binario, hexadecimal, complemento a dos, BCD y código Gray.
- Lógica combinacional y clases de compuertas (NOT, AND, OR, universales NAND/NOR y XOR/XNOR), diagramas de tiempo (retardos de propagación t_pHL y t_pLH y glitches).
- Postulados y teoremas del Álgebra de Boole (absorción, consenso y leyes de De Morgan) y metodología estructurada de 5 pasos para el diseño de redes lógicas (definir el problema en palabras, escribir proposiciones cuasi-lógicas, escribir la expresión booleana, realización con compuertas AND/NAND y dibujar el esquema).
- Obtención de expresiones booleanas a partir de una tabla de verdad mediante mintérminos (Suma de Productos SOP) y maxtérminos (Producto de Sumas POS), y minimización con mapas de Karnaugh.

2. Parámetros Nominales de Configuración del Controlador (Video 09)

Parámetro del Controlador / Interfase	Símbolo	Valor Nominal	Función en el Lazo de Control / Automatización
Ganancia Proporcional	`Kp (PB = 100/Kp)`	3	Esfuerzo inmediato proporcional al error actual e(t)
Tiempo Integral	`Ti [s]`	0.4 s	Elimina el error estacionario (Offset) acumulando la integral del error
Tiempo Derivativo y Filtro Divisor	`Td [s] / N`	Td = 0.05 s, N = 10	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]`	[0, 5]	Cota física del actuador para integración condicional (Clamping)
Rango Escalado de Ingeniería (SCALE_X)	`[PV_min, PV_max]`	[0, 5]	Conversión lineal desde cuentas crudas del ADC (0..1023 o 0..27648)

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Introducción a los Circuitos Digitales y Niveles Lógicos

Discretización en Dos Estados (0 y 1), Bandas de Tensión V_IL, V_IH, V_OL, V_OH, Margen de Ruido y Regeneración de Señal

- **Turno 1 (Masculina):** A diferencia de los circuitos analógicos donde cualquier pequeña interferencia se suma directamente a la señal continua, los circuitos digitales codifican la información utilizando únicamente dos estados discretos: el cero lógico o nivel bajo y el uno lógico o nivel alto.
- **Turno 2 (Femenina):** Físicamente, un estado lógico no es un voltaje puntual sino una banda permitida de tensiones separada de la banda opuesta por una región prohibida de transición.
- **Turno 3 (Masculina):** Una salida digital garantiza entregar un nivel alto por encima de ve o hache mínimo, mientras que la entrada siguiente reconoce como nivel alto válido cualquier tensión superior a ve i hache mínimo.
- **Turno 4 (Femenina):** La diferencia entre ve o hache y ve i hache en nivel alto, así como entre ve i ele y ve o ele en nivel bajo, define el margen de ruido estático, permitiendo que cada compuerta regenere una señal limpia sin acumular degradación.

Diapositiva 02: Representaciones Digitales: Binario, Hexadecimal, Complemento a 2 y Códigos

Ponderación Posicional en Base 2, Notación Hexadecimal Base 16, Enteros con Signo en Complemento a Dos, BCD y Código Gray

- **Turno 1 (Masculina):** En los sistemas digitales toda magnitud numérica, carácter o instrucción se representa mediante palabras de ene bits, donde un bit es un dígito binario con peso posicional en potencias de dos.
- **Turno 2 (Femenina):** Con una palabra de ene bits podemos representar dos elevado a la ene combinaciones distintas: por ejemplo, ocho bits o un byte abarcan doscientos cincuenta y seis estados, desde cero hasta doscientos cincuenta y cinco en formato sin signo.
- **Turno 3 (Masculina):** Para compactar cadenas largas de bits agrupamos bloques de cuatro bits en dígitos hexadecimales del cero al nueve y de la letra A a la F, mientras que los números negativos se codifican en complemento a dos invirtiendo los bits y sumando uno.
- **Turno 4 (Femenina):** Además, existen códigos especializados como el BCD para visualizadores decimales y el código Gray para encoders ópticos, donde entre dos posiciones consecutivas cambia únicamente un solo bit para evitar lecturas falsas en las fronteras.

Diapositiva 03: Lógica Combinacional y Clases de Lógica

Definición de Lógica Combinacional vs. Secuencial, Compuertas Básicas (NOT, AND, OR), Compuertas Universales (NAND, NOR) y XOR/XNOR

- **Turno 1 (Masculina):** Clasificamos los circuitos digitales en dos grandes clases: en la lógica combinacional las salidas en cualquier instante dependen única y exclusivamente de la combinación presente en las entradas actuales, sin poseer memoria de estados pasados.
- **Turno 2 (Femenina):** Por el contrario, en la lógica secuencial el circuito incluye elementos de almacenamiento y realimentación, de modo que su salida depende tanto de las entradas actuales como de la historia previa del sistema.
- **Turno 3 (Masculina):** Los bloques elementales de la lógica combinacional son las compuertas lógicas: el inversor NOT que complementa la variable, la compuerta AND cuya salida es uno solo si todas sus entradas son uno, y la compuerta OR cuya salida es uno si al menos una entrada es uno.
- **Turno 4 (Femenina):** Agregando inversión en la salida obtenemos las compuertas NAND y NOR, denominadas compuertas universales porque cualquier función booleana puede construirse usando exclusivamente compuertas NAND o exclusivamente compuertas NOR, además de la compuerta OR exclusiva XOR detectora de desigualdad.

Diapositiva 04: Diagramas de Tiempo (Timing Diagrams) y Retardos de Propagación

Evolución Temporal de Señales Lógicas, Retardos t_pHL y t_pLH, Camino Crítico y Riesgos Estáticos (Glitches / Hazards)

- **Turno 1 (Masculina):** Mientras la tabla de verdad describe el comportamiento estático de una red lógica, el diagrama de tiempo o timing diagram grafica las formas de onda de todas las entradas y salidas frente a un eje horizontal de tiempo común.
- **Turno 2 (Femenina):** Los diagramas de tiempo son indispensables para analizar circuitos reales porque ninguna compuerta física responde en tiempo cero: toda transición atraviesa capacitancias internas que introducen un retardo de propagación medido en nanosegundos.
- **Turno 3 (Masculina):** Especificamos dos tiempos de retardo al cincuenta por ciento de la excursión: te pe hache ele cuando la salida conmuta de nivel alto a nivel bajo, y te pe ele hache cuando conmuta de nivel bajo a nivel alto.
- **Turno 4 (Femenina):** Cuando una misma variable recorre dos caminos con distinto número de compuertas, la diferencia de retardos de propagación puede generar un pulso falso momentáneo en la salida conocido como glitch o riesgo estático.

Diapositiva 05: Álgebra de Boole y Teoremas de De Morgan

Postulados de Identidad, Dominación, Idempotencia, Complementación, Absorción y Dualidad de De Morgan ~(A·B) = ~A + ~B

- **Turno 1 (Masculina):** La herramienta matemática formal para especificar, transformar y minimizar circuitos lógicos es el álgebra de Boole, donde las variables adoptan solo los valores cero y uno bajo las operaciones de complemento, producto lógico AND y suma lógica OR.
- **Turno 2 (Femenina):** Las leyes de idempotencia establecen que A por A es igual a A y A más A es igual a A, mientras que las leyes de absorción permiten simplificar directamente A más A por B como simplemente A.
- **Turno 3 (Masculina):** Asimismo, el teorema de factorización útil demuestra que A más A negada por B se reduce de inmediato a A más B, eliminando un literal redundante y ahorrando una entrada de compuerta.
- **Turno 4 (Femenina):** Los teoremas más poderosos para transformar circuitos son las leyes de De Morgan: el complemento de un producto es igual a la suma de los complementos individuales, y el complemento de una suma es igual al producto de los complementos.

Diapositiva 06: Diseño de Redes Lógicas — Pasos 1 y 2: Del Problema en Palabras a Proposiciones Cuasi-Lógicas

1. Definir el Problema en Palabras y Variables Binarias | 2. Escribir Enunciados Cuasi-Lógicos Estructurados (SI ... Y / O / NO)

- **Turno 1 (Masculina):** Para transformar un requerimiento práctico de ingeniería en un circuito digital funcional libre de errores seguimos una metodología sistemática de cinco pasos para el diseño de redes lógicas.
- **Turno 2 (Femenina):** El Paso uno consiste en definir el problema en palabras con total claridad, identificando cada condición de entrada y cada acción de salida, y asignando el valor lógico uno o cero al estado activo de cada sensor y actuador.
- **Turno 3 (Masculina):** Por ejemplo, en una prensa industrial podemos definir las variables de entrada: pieza presente pe igual a uno, guarda cerrada ge igual a uno, emergencia presionada e igual a uno, y la salida de activación del solenoide ye igual a uno.
- **Turno 4 (Femenina):** El Paso dos consiste en traducir esa especificación verbal en proposiciones cuasi-lógicas utilizando los conectores en mayúsculas SI, ENTONCES, Y, O y NO: el solenoide ye es verdadero SI hay pieza Y la guarda está cerrada Y NO está activa la emergencia.

Diapositiva 07: Diseño de Redes Lógicas — Pasos 3, 4 y 5: Expresión Booleana, Realización AND/NAND y Diagrama

3. Escribir la Expresión Booleana | 4. Simplificación y Realización con Compuertas AND/NAND | 5. Dibujar el Diagrama Esquemático

- **Turno 1 (Masculina):** Una vez redactadas las proposiciones cuasi-lógicas, en el Paso tres escribimos directamente la expresión booleana sustituyendo cada conector Y por un producto lógico, cada conector O por una suma lógica y cada NO por una barra de negación.
- **Turno 2 (Femenina):** En el Paso cuatro simplificamos algebraicamente la expresión booleana para minimizar el número de literales y aplicamos su realización con compuertas comerciales, con frecuencia convirtiendo la red a una realización exclusiva con compuertas NAND mediante doble negación y De Morgan.
- **Turno 3 (Masculina):** Realizar un circuito completo con un solo tipo de compuerta como la setenta y cuatro HC cero cero de cuatro compuertas NAND de dos entradas reduce el número de circuitos integrados necesarios en la placa.
- **Turno 4 (Femenina):** Finalmente, en el Paso cinco dibujamos el diagrama del circuito con sus símbolos normalizados, rieles de entradas ordenados a la izquierda, niveles de compuertas de izquierda a derecha y numeración de pines del encapsulado.

Diapositiva 08: Obtención de una Expresión Booleana Dada una Tabla de Verdad

Mintérminos (Filas donde Y = 1), Forma Canónica Suma de Productos (SOP), Maxtérminos (Y = 0) y Producto de Sumas (POS)

- **Turno 1 (Masculina):** En muchos diseños partimos directamente de una tabla de verdad que enumera las dos a la ene combinaciones posibles de las entradas y especifica el valor deseado cero o uno en la salida para cada fila.
- **Turno 2 (Femenina):** Para obtener de manera directa la expresión booleana a partir de la tabla de verdad mediante el método de mintérminos, nos enfocamos únicamente en aquellas filas donde la salida vale exactamente uno.
- **Turno 3 (Masculina):** Para cada fila con salida igual a uno escribimos un término producto que incluye todas las variables de entrada: si en esa fila una variable vale uno la escribimos en forma directa, y si vale cero la escribimos complementada.
- **Turno 4 (Femenina):** Al unir mediante operaciones OR todos esos mintérminos obtenemos la forma canónica Suma de Productos o SOP; de manera dual, tomando las filas donde la salida vale cero se construye el Producto de Sumas o POS con maxtérminos.

Diapositiva 09: Simplificación por Mapas de Karnaugh y Bloques Combinacionales

Adyacencia en Código Gray en Mapas de 3 y 4 Variables, Agrupación en Potencias de 2, Condiciones Don't Care (X) y Multiplexores

- **Turno 1 (Masculina):** La expresión canónica Suma de Productos obtenida de la tabla de verdad es funcionalmente exacta, pero rara vez es la más económica en número de compuertas.
- **Turno 2 (Femenina):** El mapa de Karnaugh organiza la tabla de verdad en una cuadrícula bidimensional cuyas filas y columnas siguen el orden del código Gray, de modo que dos celdas contiguas horizontales, verticales o en los bordes extremos difieren en una sola variable.
- **Turno 3 (Masculina):** Agrupando celdas adyacentes que contienen unos en rectángulos lo más grandes posible de dos, cuatro u ocho casillas, eliminamos automáticamente las variables que cambian de estado dentro del grupo.
- **Turno 4 (Femenina):** Además, las combinaciones de entrada que físicamente nunca pueden ocurrir se marcan como condiciones no importa o don't care con una equis, pudiendo incluirse libremente para formar grupos aún mayores y simplificar al máximo el circuito.

Diapositiva 10: Síntesis de la Parte 5A y Transición hacia la Lógica Secuencial

De las Redes Combinacionales sin Memoria a los Latches, Flip-Flops, Contadores, Familias TTL/CMOS y Símbolos IEEE

- **Turno 1 (Masculina):** En esta Parte cinco A hemos dominado la representación de información digital, las compuertas lógicas, el análisis temporal con retardos de propagación, el álgebra de Boole y el método completo de síntesis desde un problema verbal o tabla de verdad.
- **Turno 2 (Femenina):** Toda red combinacional calcula su salida al instante a partir de sus entradas presentes, pero es incapaz de recordar un evento una vez que el pulso de entrada desaparece.
- **Turno 3 (Masculina):** Para construir registros de datos, contadores binarios, divisores de frecuencia y máquinas de estados necesitamos incorporar memoria al circuito digital.
- **Turno 4 (Femenina):** En la Parte cinco B estudiaremos la lógica secuencial, los flip-flops y sus aplicaciones, la interconexión real de familias TTL y CMOS, el temporizador quinientos cincuenta y cinco y los símbolos digitales estándar IEEE.

4. Firmware C++/MCU: PID Discreto Tustin, FSM y CRC-16 (controlador_tiempo_real_video_09.ino)

```
/*
 * ============================================================================
 * CURSO COMPLETO: FUNDAMENTOS DE ELECTRÓNICA
 * VIDEO 09: Video 09 — Parte 5A: Representaciones Digitales, Lógica Combinacional, Diagramas de Tiempo, Álgebra de Boole y Diseño de Redes Lógicas
 * Archivo: controlador_tiempo_real_video_09.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:
 *   Niveles Lógicos y Margen de Ruido, Binario/Hex/Gray, Compuertas, Timing Diagrams, Álgebra de Boole, Diseño en 5 Pasos y Tablas de Verdad
 * ============================================================================
 */

#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 09) ---
const float KP_GAIN      = 3.000f;
const float TI_SEC       = 0.400f;
const float TD_SEC       = 0.050f;
const float N_FILT       = 10.0f;
const float TS_SEC       = 0.0010f; // Periodo de muestreo Ts = 1 ms
const float U_MIN_LIM    = 0.0f;
const float U_MAX_LIM    = 5.0f;
const float PV_MIN_ENG   = 0.0f;
const float PV_MAX_ENG   = 5.0f;

// --- 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 09"));
  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_09.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: FUNDAMENTOS DE ELECTRÓNICA
VIDEO 09: Video 09 — Parte 5A: Representaciones Digitales, Lógica Combinacional, Diagramas de Tiempo, Álgebra de
Boole y Diseño de Redes Lógicas
Archivo: supervision_sbc_y_sintonizacion_video_09.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 09
KP_NOMINAL = 3
TI_NOMINAL = 0.4
TD_NOMINAL = 0.05
N_FILTRO   = 10
TS_SEG     = 0.0010
U_MIN      = 0
U_MAX      = 5


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: FUNDAMENTOS DE ELECTRÓNICA — VIDEO 09")
    print(" Enfoque:", "Niveles Lógicos y Margen de Ruido, Binario/Hex/Gray, Compuertas, Timing Diagrams, Álgebra
de Boole, Diseño en 5 Pasos y Tablas de Verdad")
    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=2.5)
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_09.st)

```
(* ==========================================================================
 * CURSO COMPLETO: FUNDAMENTOS DE ELECTRÓNICA
 * VIDEO 09: Video 09 — Parte 5A: Representaciones Digitales, Lógica Combinacional, Diagramas de Tiempo, Álgebra
de Boole y Diseño de Redes Lógicas
 * Archivo: automatizacion_plc_iec61131_video_09.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_V09) 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_V09
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 := 3.0;
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 := 5.0;

    (* 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 := 3.000;
    rTi_s                : REAL := 0.400;
    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
```