

NOVA RP2350 — Guía Técnica para Ingenieros y Técnicos

Tu Plataforma Profesional de Desarrollo IoT

NOVA es una placa de desarrollo basada en el **Raspberry Pi RP2350A**, el microcontrolador de doble núcleo más reciente de la familia Pico. Diseñada para profesionales técnicos e ingenieros que necesitan una solución robusta para proyectos industriales, IoT y automatización.

¿Por qué NOVA RP2350?

Especificaciones del RP2350A

- **Procesador:** Dual-core ARM Cortex-M33 @ 150MHz (o RISC-V Hazard3)
- **Memoria RAM:** 520KB SRAM
- **Memoria Flash:** 16MB (externo)
- **GPIO:** 30 pines programables con soporte PWM
- **Interfaces:**
 - 2× UART (comunicación serial)
 - 2× I2C (hasta 400kHz Fast Mode)
 - 2× SPI (hasta 62.5MHz)
 - 12× ADC de 12 bits (500 ksps)
 - 24× canales PWM
 - USB 1.1 Device/Host

NOVA

- **DMA**: 12 canales independientes
- **PIO**: 2 bloques de E/S programable (8 máquinas de estado)
- **Seguridad**: Secure Boot, encriptación de memoria
- **Alimentación**: 1.8V-5.5V (regulador 3.3V integrado en NOVA)

Proyectos Industriales y Profesionales

1. Control de Motor Trifásico con FOC

El RP2350 es ideal para control de motores trifásicos usando **Field-Oriented Control (FOC)**.

Configuración Hardware

NOVA RP2350 → Driver de Puente H (DRV8323/DRV8305) → Motor Trifásico BLDC
↓
Sensor Hall/Encoder para realimentación

Pines Necesarios:

- 6 salidas PWM para las 3 fases (alta/baja por fase)
- 3 entradas para sensores Hall o encoder incremental
- 2 ADC para medición de corriente (shunt resistors)

Código Base MicroPython

```

from machine import Pin, PWM
import array

# Configurar 6 canales PWM para motor trifásico
pwm_pins = [0, 1, 2, 3, 4, 5] # UH, UL, VH, VL, WH, WL
pwm_freq = 20000 # 20kHz para operación silenciosa

pwm_channels = []
for pin_num in pwm_pins:
    pwm = PWM(Pin(pin_num))
    pwm.freq(pwm_freq)
    pwm_channels.append(pwm)

# Tabla de conmutación para 6 pasos (BLDC básico)
# Formato: [UH, UL, VH, VL, WH, WL]
commutation_table = [
    [65535, 0, 0, 0, 0, 65535], # Paso 1
    [65535, 0, 0, 65535, 0, 0], # Paso 2
    [0, 0, 65535, 0, 0, 65535], # Paso 3
    [0, 65535, 65535, 0, 0, 0], # Paso 4
    [0, 65535, 0, 0, 65535, 0], # Paso 5
    [0, 0, 0, 65535, 65535, 0], # Paso 6
]

def set_motor_step(step):
    """Aplica un paso de conmutación al motor."""
    for i, duty in enumerate(commutation_table[step]):
        pwm_channels[i].duty_u16(duty)

# Sensores Hall (pines 10, 11, 12)
hall_sensors = [Pin(10, Pin.IN), Pin(11, Pin.IN), Pin(12, Pin.IN)]

def read_hall_state():
    """Lee el estado de los sensores Hall (0-7)."""
    return (hall_sensors[0].value() << 2 |
            hall_sensors[1].value() << 1 |
            hall_sensors[2].value())

# Loop de control básico
import time
step = 0
while True:
    set_motor_step(step)
    step = (step + 1) % 6
    time.sleep_ms(10) # Velocidad controlada

```

NOVA

Control FOC Avanzado

Para implementar FOC verdadero, necesitas:

1. **Transformaciones Park-Clarke**: Convertir corrientes trifásicas a coordenadas dq
2. **Controladores PI**: Para corriente y velocidad
3. **PWM de espacio vectorial (SVPWM)**: Para modulación eficiente
4. **Estimación de posición**: BEMF o encoder absoluto

Bibliotecas Recomendadas:

- `micropython-foc` (si existe port para RP2350)
- Implementación propia usando arrays y math

2. Sistema IoT Industrial con MQTT

Arquitectura del Sistema

Sensores → NOVA RP2350 → WiFi/Ethernet → Broker MQTT → Dashboard/Base de Datos
↓
Control Local (actuadores)

Implementación con ESP-01 (WiFi)

Conexión UART:

NOVA

```

from machine import UART, Pin
import json
import time

# Configurar UART para ESP-01 (AT commands)
uart = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1))

def send_at_command(cmd, timeout=1000):
    """Envía comando AT y espera respuesta."""
    uart.write(cmd + '\r\n')
    time.sleep_ms(timeout)
    response = uart.read()
    return response.decode() if response else ""

# Configurar WiFi
send_at_command('AT+CWMODE=1') # Modo cliente
send_at_command('AT+CWJAP="TU_SSID","TU_PASSWORD"')

# Conectar a broker MQTT (método simplificado)
# Para producción, usar biblioteca MQTT completa
def connect_mqtt(broker, port=1883):
    cmd = f'AT+CIPSTART="TCP","{broker}",{port}'
    return send_at_command(cmd, timeout=5000)

# Publicar datos
def publish_sensor_data(topic, value):
    payload = json.dumps({"sensor": topic, "value": value, "timestamp": time.time()})
    # Implementar protocolo MQTT aquí
    print(f"Publishing to {topic}: {payload}")

```

Sensores Industriales Comunes

Sensor de Temperatura (Termopar K con MAX31855):

NOVA

```

from machine import Pin, SPI

spi = SPI(0, baudrate=1000000, polarity=0, phase=0,
          sck=Pin(2), mosi=Pin(3), miso=Pin(4))
cs = Pin(5, Pin.OUT, value=1)

def read_thermocouple():
    """Lee temperatura de termopar tipo K."""
    cs.value(0)
    data = spi.read(4)
    cs.value(1)

    temp_raw = (data[0] << 24 | data[1] << 16 | data[2] << 8 | data[3]) >> 18
    if temp_raw & 0x2000: # Bit de signo
        temp_raw = temp_raw - 0x4000

    temp_celsius = temp_raw * 0.25
    return temp_celsius

# Lectura continua
import time
while True:
    temp = read_thermocouple()
    print(f"Temperatura: {temp:.2f}°C")
    time.sleep(1)

```

3. Adquisición de Datos de Alta Velocidad

El RP2350 puede muestrear hasta 500 ksps con sus ADC de 12 bits.

Ejemplo: Osciloscopio Digital Básico

```

from machine import ADC, Pin
import array

# Configurar ADC
adc = ADC(Pin(26)) # GPIO26 = ADC0
sample_rate = 100000 # 100 kHz

# Buffer para muestras
samples = array.array('H', [0] * 1000) # 1000 muestras de 16 bits

def capture_samples():
    """Captura muestras a alta velocidad."""
    for i in range(len(samples)):
        samples[i] = adc.read_u16()
        # Delay mínimo para rate control (usar timer para precisión)

    return samples

# Análisis básico
def analyze_signal(samples):
    """Calcula estadísticas básicas."""
    min_val = min(samples)
    max_val = max(samples)
    avg_val = sum(samples) / len(samples)

    # Convertir a voltaje (3.3V referencia)
    voltage_min = min_val * 3.3 / 65535
    voltage_max = max_val * 3.3 / 65535
    voltage_avg = avg_val * 3.3 / 65535

    return {
        "min_v": voltage_min,
        "max_v": voltage_max,
        "avg_v": voltage_avg,
        "pk_pk": voltage_max - voltage_min
    }

# Capturar y analizar
data = capture_samples()
stats = analyze_signal(data)
print(f"Pico-Pico: {stats['pk_pk']:.3f}V")

```

4. Comunicación Industrial RS485 (Modbus RTU)

Para entornos industriales, RS485 es estándar.

Hardware Necesario

- Módulo MAX485 o equivalente
- Resistencia terminadora 120Ω
- NOVA conectado vía UART

Implementación Modbus RTU

```

from machine import UART, Pin
import struct
import time

class ModbusRTU:
    def __init__(self, uart_id=1, baudrate=9600, de_pin=14):
        self.uart = UART(uart_id, baudrate=baudrate)
        self.de = Pin(de_pin, Pin.OUT, value=0) # Driver Enable

    def calculate_crc(self, data):
        """Calcula CRC16 Modbus."""
        crc = 0xFFFF
        for byte in data:
            crc ^= byte
            for _ in range(8):
                if crc & 0x0001:
                    crc = (crc >> 1) ^ 0xA001
                else:
                    crc >>= 1
        return crc

    def read_holding_registers(self, slave_addr, start_addr, count):
        """Lee registros holding (función 0x03)."""
        # Construir frame
        frame = bytearray([slave_addr, 0x03])
        frame += struct.pack('>HH', start_addr, count)

        # Agregar CRC
        crc = self.calculate_crc(frame)
        frame += struct.pack('= 5:
            # Validar CRC y parsear
            rx_crc = struct.unpack('{count}H', response[3:3+byte_count])
            return values

        return None

# Ejemplo de uso
modbus = ModbusRTU(uart_id=1, baudrate=9600, de_pin=14)

# Leer 2 registros del esclavo 1, dirección 0x1000
registers = modbus.read_holding_registers(slave_addr=1, start_addr=0x1000, count=2)
if registers:
    print(f"Registros: {registers}")

```

5. Control PID para Procesos Industriales

Control de temperatura, presión o nivel con PID.

```

import time

class PIDController:
    def __init__(self, kp, ki, kd, setpoint=0):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.setpoint = setpoint

        self.integral = 0
        self.prev_error = 0
        self.prev_time = time.time()

    def update(self, process_value):
        """Calcula salida PID."""
        current_time = time.time()
        dt = time.time_diff(current_time, self.prev_time) / 1000.0 # a segundos

        error = self.setpoint - process_value

        # Proporcional
        p_term = self.kp * error

        # Integral
        self.integral += error * dt
        i_term = self.ki * self.integral

        # Derivativo
        derivative = (error - self.prev_error) / dt if dt > 0 else 0
        d_term = self.kd * derivative

        # Salida
        output = p_term + i_term + d_term

        # Anti-windup (limitar integral)
        if output > 100:
            output = 100
            self.integral -= error * dt # Revertir integral
        elif output < 0:
            output = 0
            self.integral -= error * dt

        self.prev_error = error
        self.prev_time = current_time

        return output

# Ejemplo: Control de temperatura
from machine import Pin, PWM, ADC

```

NOVA

```

pid = PIDController(kp=2.0, ki=0.5, kd=1.0, setpoint=50.0) # 50°C

# Salida PWM para heater
heater = PWM(Pin(15))
heater.freq(1000)

# Sensor de temperatura (LM35)
temp_sensor = ADC(Pin(26))

while True:
    # Leer temperatura
    adc_val = temp_sensor.read_u16()
    temp_celsius = (adc_val * 3.3 / 65535) * 100 # LM35: 10mV/°C

    # Calcular control
    control = pid.update(temp_celsius)

    # Aplicar a heater
    duty = int(control * 655.35) # 0-100 → 0-65535
    heater.duty_u16(duty)

    print(f"Temp: {temp_celsius:.1f}°C, Control: {control:.1f}%")
    time.sleep(0.1)

```

Herramientas de Desarrollo Profesional

IDE y Entornos

1. **Thonny**: Para desarrollo rápido de MicroPython
2. **VS Code + PyMakr**: Desarrollo profesional con debugging
3. **Arduino IDE**: Para código C/C++ con core RP2040/RP2350
4. **CMake + GCC**: Desarrollo bare-metal con Pico SDK

Debugging con SWD

El RP2350 soporta debugging por SWD (Serial Wire Debug).

Hardware:

- Raspberry Pi Pico como debugger (Picoprobe)
- Conexión: SWCLK, SWDIO, GND

OpenOCD:

NOVA

```
# Instalar OpenOCD
sudo apt install openocd

# Conectar
openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg

# En otra terminal, GDB
arm-none-eabi-gdb firmware.elf
(gdb) target extended-remote localhost:3333
(gdb) load
(gdb) monitor reset init
(gdb) continue
```

Casos de Uso Profesionales

1. Datalogger Industrial

- Muestreo de sensores a intervalos regulares
- Almacenamiento en SD card (SPI)
- Timestamp con RTC (DS3231)
- Transmisión por WiFi/Ethernet

2. Controlador de Inversor Solar

- Medición de voltaje/corriente DC
- Control MPPT (Maximum Power Point Tracking)
- Monitoreo de baterías
- Interfaz Modbus para integración

3. Sistema de Monitoreo de Vibración

- Acelerómetros MEMS (I2C/SPI)
- Análisis FFT para detección de fallas
- Alarmas por umbral
- Conexión a SCADA

4. Gateway IoT Multi-protocolo

- Conversión Modbus → MQTT
- Agregación de datos de múltiples sensores

NOVA

- Buffer local ante pérdida de conectividad
- Encriptación TLS

Ventajas sobre Arduino y Otras Plataformas

Característica	NOVA RP2350	Arduino Uno	ESP32
CPU	Dual-core 150MHz	16MHz	Dual-core 240MHz
RAM	520KB	2KB	520KB
GPIO	30	14	34
ADC	12-bit, 500ksps	10-bit	12-bit
PIO	Sí (8 máquinas)	No	No
Precio	Competitivo	~\$25 USD	~\$5-10 USD
Consumo	Bajo	Medio	Medio-Alto
WiFi/BT	Externo	Externo	Integrado

PIO es el diferenciador: Permite protocolos personalizados sin ocupar la CPU.

Recursos y Documentación

Oficial Raspberry Pi

- **Datasheet RP2350:** <https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf>
- **Pico SDK:** <https://github.com/raspberrypi/pico-sdk>
- **Getting Started:** <https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf>

MicroPython

- **Documentación:** <https://docs.micropython.org/>
- **Bibliotecas:** <https://github.com/micropython/micropython-lib>

Comunidad LATAM

- Foros: Raspberry Pi Forums (sección RP2040/RP2350)
 - Discord: Canales en español de makers y electrónica
 - YouTube: Tutoriales en español de RP2350
-

Proveedores en México/LATAM

Componentes

- **Steren** (México): Componentes básicos
- **ElectronicaSV** (México): Módulos y sensores
- **Sigmaelectronics** (México): Industrial
- **TodoMicro** (Chile): RP2040/RP2350
- **330ohms** (Panamá): Distribución regional

Módulos Industriales

- **Driver de motor:** DRV8825, TB6600, DRV8323
 - **Sensores:** DHT22, BME280, INA219, MAX31855
 - **Comunicación:** MAX485, ESP-01, SIM800L
-

Soporte y Consultoría

Para proyectos industriales complejos que requieren:

- Diseño de PCB personalizado
- Certificaciones (CE, FCC)
- Firmware a medida
- Integración con sistemas SCADA/MES

Contacta a integradores locales o desarrolla internamente con esta guía.

Conclusión

NOVA RP2350 es una plataforma profesional para ingenieros y técnicos que necesitan:

- **Rendimiento:** Dual-core, alta velocidad, DMA
- **Flexibilidad:** PIO, múltiples protocolos
- **Costo-beneficio:** Precio accesible vs capacidades
- **Ecosistema:** Amplia documentación y comunidad

Ideal para IoT industrial, automatización y control en LATAM.

¿Listo para empezar? Instala MicroPython, carga tu primer programa y comienza a construir soluciones industriales profesionales.