Manual Técnico del Protocolo Modbus RTU y Verificación CRC-16 (Módulo 04)

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 4, 11, 12, 13 y 14

---

1. Fundamentos Matemáticos del Código de Redundancia Cíclica `CRC-16/MODBUS`

En enlaces serie asíncronos (`UART` y `RS-485`) expuestos a ruido electromagnético industrial, un simple bit de paridad o una suma aritmética de bytes (*Checksum*) no detecta inversiones simétricas de bits ni transposiciones de octetos. El estándar `Modbus RTU` emplea el polinomio generador algebraico de grado 16:

$$G(x) = x^{16} + x^{15} + x^2 + 1$$

1.1 Parámetros Canónicos del Algoritmo `CRC-16/MODBUS`

- **Representación normal del polinomio**: `0x8005` ($1000\,0000\,0000\,0101_2$).
- **Representación reflejada (procesamiento LSB primero)**: `0xA001` ($1010\,0000\,0000\,0001_2$).
- **Valor semilla inicial (`Init`)**: `0xFFFF` .
- **Orden de transmisión en la línea**: Primero se envía el byte menos significativo (`CRC_Lo`) y en último lugar el byte más significativo (`CRC_Hi`).
- **Capacidad de detección**: Detecta el 100% de los errores de 1, 2 y número impar de bits, todas las ráfagas de error de hasta 16 bits de longitud, y el 99.9985% ($1 - 2^{-16}$) de ráfagas mayores.

1.2 Algoritmo Iterativo Paso a Paso (C++ / Python)

Para una trama de N bytes de carga útil $B_0, B_1, \dots, B_{N-1}$:

- Inicializar el acumulador de 16 bits: $CRC \leftarrow 0xFFFF$.
- Para cada byte B_k ($k = 0 \dots N-1$):
- Operar $CRC \leftarrow CRC \oplus B_k$.
- Repetir 8 veces (para cada bit $i = 0 \dots 7$):
- Si el bit menos significativo $(CRC \,\&\, 0x0001) = 1$, entonces: $CRC \leftarrow (CRC >> 1) \oplus 0xA001$.
- En caso contrario: $CRC \leftarrow (CRC >> 1)$.
- Al finalizar, adjuntar $CRC_{Lo} = CRC \,\&\, 0xFF$ seguido de $CRC_{Hi} = (CRC >> 8) \,\&\, 0xFF$.

---

2. Estructura de Trama Binaria y Temporización `Modbus RTU`

A diferencia de formatos textuales como JSON o CSV, `Modbus RTU` transmite octetos binarios puros delimitados exclusivamente por **intervalos de silencio en la línea**:

Campo de la Trama	Tamaño	Codificación	Descripción Técnica
Silencio Inicial	≥ 3.5 char	Línea en reposo (`Idle`)	A `19200 bps` (`8E1`), equivale a ≥ 2.0 ms sin actividad
Dirección de Esclavo (`ID`)	1 Byte	`0x01` a `0xF7` (1\dots 247)	Identifica al nodo destinatario (`0x00` reservado para difusión *Broadcast*)
Código de Función (`FC`)	1 Byte	`0x03` o `0x06`	`0x03` : Leer Registros de Retención · `0x06` : Escribir Registro Individual
Campo de Datos (`Payload`)	N Bytes	**Big-Endian** (`MSB` primero)	Direcciones de registro de 16 bits, cantidad de palabras y valores escalados
Verificación `CRC-16`	2 Bytes	**Little-Endian** (`LSB` primero)	Primero se transmite `CRC_Lo` y finalmente `CRC_Hi`
Silencio Final	≥ 3.5 char	Línea en reposo (`Idle`)	Cierra la trama actual y habilita el bus para la siguiente transacción

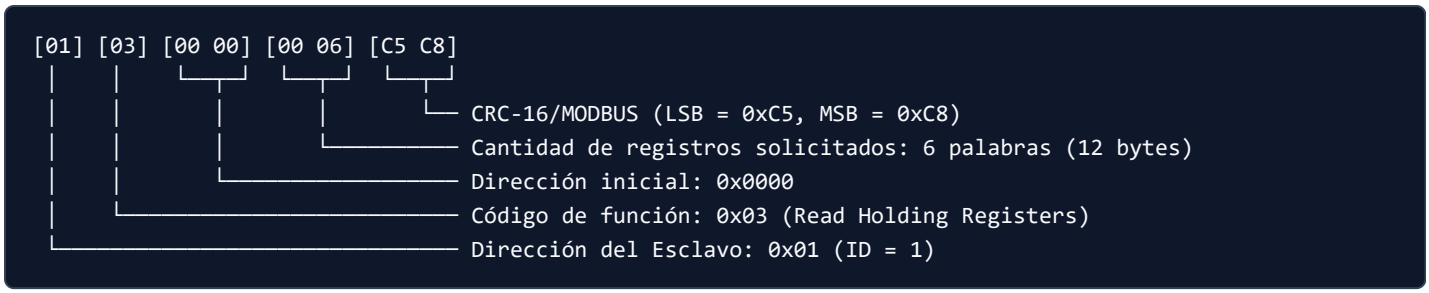
3. Mapa Oficial de Registros de Retención del Nodo de Invernadero (`ID = 1`)

Para evitar el envío de números en coma flotante (`float` IEEE-754 de 32 bits) y problemas de orden de palabras, todas las magnitudes continuas se transmiten como **enteros de 16 bits con escalado en punto fijo**:

Dirección Hex (Dec)	Nombre Simbólico	Tipo de Dato	Factor de Escala	Acceso	Descripción y Conversión en la Raspberry Pi
`0x0000` (`0`)	`REG_TEMP_C_X100`	`int16_t`	$\times 100$	Solo Lectura (`0x03`)	Temperatura filtrada EMA en centésimas de °C ($T = Valor / 100.0$, ej. `2540` $\rightarrow 25.40$ ^\circC)
`0x0001` (`1`)	`REG_LIGHT_PCT_X10`	`uint16_t`	$\times 10$	Solo Lectura (`0x03`)	Luminosidad relativa en décimas de porcentaje ($L = Valor / 10.0$, ej. `753` $\rightarrow 75.3\%$)
`0x0002` (`2`)	`REG_FAN_PWM`	`uint16_t`	$1:1$ (0\dots 255)	Solo Lectura (`0x03`)	Ciclo de trabajo PWM actual aplicado al ventilador en el pin `D9`
`0x0003` (`3`)	`REG_STATE_FLAGS`	`uint16_t`	Mapa de Bits	Solo Lectura (`0x03`)	Bits `0..3` : Estado FSM (`0..4`) · Bit `4` : Calefactor activo · Bits `8..15` : Código de fallo
`0x0004` (`4`)	`REG_SETPOINT_X100`	`int16_t`	$\times 100$	Lectura / Escritura (`0x03` , `0x06`)	Consigna térmica de control en centésimas de °C (rango válido: `1500` a `4000` , es decir, 15.00 ^\circC a 40.00 ^\circC)
`0x0005` (`5`)	`REG_CMD_CONTROL`	`uint16_t`	Comando	Lectura / Escritura (`0x03` , `0x06`)	`1` : Rearmar alarma de fallo (`FAULT_SENSOR` $\rightarrow$ `IDLE`) · `2` : Forzar parada segura

4. Ejemplo de Trama Hexadecimal Real en el Bus RS-485

4.1 Petición del Maestro (Lectura de los 6 Registros desde `0x0000` , Función `0x03`)



4.2 Respuesta del Esclavo Arduino (`17 Bytes` en total)

