Internet de las Cosas con Arduino y Raspberry Pi

Capítulo 15 · UART, I²C, SPI y tramas CRC-16

Conceptos clave

Elige el bus según distancia, velocidad y cantidad de dispositivos. Comprueba orden de bytes, variante de CRC y encuadre con vectores de prueba.



Ejemplo complementario

```
def crc16_modbus(datos):
    crc = 0xFFFF
    for byte in datos:
        crc ^= byte
        for _ in range(8):
            crc = (crc >> 1) ^ 0xA001 if crc & 1 else crc >> 1
    return crc

assert crc16_modbus(b"123456789") == 0x4B37
```

Guía Paso a Paso — Laboratorio 04: Comunicación Directa Arduino ↔ Raspberry Pi (UART, I²C, SPI y Bus Industrial RS-485 / Modbus RTU)

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 1, 3, 5, 9, 10, 12, 13 y 14

---

1. Objetivo de Ingeniería del Laboratorio

Integrar el nodo controlador de tiempo real basado en **Arduino UNO (`5.0 V`)** con la computadora de borde **Raspberry Pi (`3.3 V`)** analizando cuatro tecnologías de enlace físico y lógico:

- **Enlace Serie Asíncrono (`UART`)**: Comprensión del fenómeno de reinicio automático por línea `DTR` a través del condensador de acoplamiento de $100\ \text{nF}$ y aseguramiento de integridad de trama mediante **verificación de redundancia cíclica (`CRC-16/MODBUS`)**.
- **Buses Síncronos de Corto Alcance (`I²C` y `SPI`)**: Traslación segura de nivel lógico bidireccional con transistores MOSFET de canal N `BSS138` y desactivación por software de las resistencias internas de elevación a `5.0 V` del microcontrolador AVR.
- **Capa Física Diferencial Industrial (`RS-485`)**: Cableado en cadena (*Daisy-Chain*) con par trenzado blindado, cálculo e instalación de resistencias terminadoras de $120\ \Omega$ únicamente en los dos extremos físicos de la línea troncal ($R_{eq} = 60\ \Omega$) y control de dirección semidúplex (`DE` / `/RE`) en el transceptor `MAX485`.
- **Protocolo Industrial `Modbus RTU`**: Exposición del nodo de invernadero como Esclavo `ID = 1` a `19200 bps` en formato `8E1` (8 bits de datos, paridad Par, 1 bit de parada), implementando las funciones `0x03` (Lectura de Registros de Retención) y `0x06` (Escritura de Registro Individual) sobre 6 registros de $16\ \text{bits}$ con escalado en punto fijo.

---

2. Etapa 1: Enlace UART y Prevención del Auto-Reset por Línea DTR

2.1 Diagnóstico del Fenómeno de Auto-Reset (`100 nF`)

Al conectar el Arduino UNO por cable USB a la Raspberry Pi (`/dev/ttyACM0`), el chip conversor USB-Serie controla la línea `DTR` (*Data Terminal Ready*), la cual está acoplada al pin `RESET` del ATmega328P mediante un condensador cerámico en serie de $C_{DTR} = 100\ \text{nF}$:

- Cada vez que un proceso en Linux abre el descriptor del puerto serie (`serial.Serial()`), la señal `DTR` cae de $5.0\ \text{V}$ a $0\ \text{V}$.
- El condensador de $100\ \text{nF}$ transmite ese escalón negativo al pin `RESET`, reiniciando el microcontrolador y activando el cargador de arranque (*Bootloader*) durante aproximadamente $1.5\ \text{s}$ a $1.8\ \text{s}$.
- **Regla de Oro de Software**: Mantener el puerto serie abierto de forma persistente durante toda la vida del servicio, aguardar `time.sleep(2.0)` tras su apertura inicial y vaciar las colas de recepción (`reset_input_buffer()`) antes de emitir la primera consulta.

2.2 Alternativa de Enlace UART Directo en Cabezal GPIO (`/dev/serial0`)

Si se enlazan directamente los pines `TX` / `RX` sin pasar por USB:

- Conectar el pin `D1 (TX, 5.0 V)` del Arduino hacia una resistencia en serie $R_1 = 1.8\ \text{k}\Omega$, y derivar desde el nodo central una resistencia $R_2 = 3.3\ \text{k}\Omega$ hacia `GND`.
- La tensión resultante entregada al **Pin Físico 10 (`GPIO 15 / RXD`)** de la Raspberry Pi es:

> $V_{RX,RPi} = 5.0\ V \cdot (3.3\ k\Omega) / (1.8\ k\Omega + 3.3\ k\Omega) = 3.235\ V < 3.30\ V$

---

3. Etapa 2: Coprocesador en Bus I²C con Conversor Bidireccional BSS138

Cuando el Arduino UNO y la Raspberry Pi residen dentro del mismo gabinete eléctrico a menos de $20\ cm$:

- **Cableado del Adaptador `BSS138`**:
- Lado de Baja Tensión (`LV`): Conectar `LV` al **Pin 01 (`+3.3V`)** de la Raspberry Pi, `A1` a **`GPIO 2 (SDA, Pin 03)`**, `A2` a **`GPIO 3 (SCL, Pin 05)`** y `GND` al **Pin 06 (`GND`)**.
- Lado de Alta Tensión (`HV`): Conectar `HV` al pin **`5V`** del Arduino UNO, `B1` al pin **`A4 (SDA)`**, `B2` al pin **`A5 (SCL)`** y `GND` a tierra común.
- **Protección del Dominio de `3.3 V` en el Firmware del Arduino (`coprocesador_i2c_esclavo.ino`)**:
- Inmediatamente después de invocar `Wire.begin(0x08)`, la biblioteca estándar de Arduino activa por defecto las resistencias internas de elevación a $+5.0\ V$ en los pines `A4` y `A5`.
- **Acción obligatoria**: Ejecutar `digitalWrite(SDA, LOW)` y `digitalWrite(SCL, LOW)` justo después de `Wire.begin(0x08)` para desactivar los *pull-ups* internos a $5\ V$.
- **Verificación en la Raspberry Pi**:
- Habilitar el bus `I²C` con `sudo raspi-config nonint do_i2c 0`.
- Ejecutar `i2cdetect -y 1` y comprobar que aparece la dirección hexadecimal **`08`** en la matriz de escrutinio.

---

4. Etapa 3: Montaje del Bus Diferencial Industrial RS-485 (`MAX485`)

Para distancias de $10\ m$ a $1200\ m$ en entornos con motores, variadores de frecuencia y contactores:

4.1 Conexión entre Arduino UNO y Transceptor `MAX485`

- **Alimentación**: Conectar `VCC` del módulo `MAX485` a `+5.0 V` del Arduino y `GND` a tierra común (con condensador de desacoplo de $100\ nF$ entre `VCC` y `GND`).
- **Señales UART**:
- Pin **`RO`** (*Receiver Output*) del `MAX485` → Pin **`D0 (RX)`** del Arduino UNO.
- Pin **`DI`** (*Driver Input*) del `MAX485` → Pin **`D1 (TX)`** del Arduino UNO.
- **Control de Dirección Semidúplex (`DE` + `/RE`)**:
- Unir físicamente los pines **`DE`** (*Driver Enable*, activo en alto) y **`/RE`** (*Receiver Enable*, activo en bajo) del módulo `MAX485`.
- Conectar ambos pines unidos al **Pin Digital `D4`** del Arduino UNO, añadiendo una resistencia *pull-down* de $10\ k\Omega$ hacia `GND` para garantizar que el nodo permanezca en modo escucha (`LOW`) mientras el microcontrolador arranca.

4.2 Cableado de la Línea Troncal Diferencial (`A+` , `B-` y `GND`)

- Utilizar **cable de par trenzado blindado** de impedancia característica $Z_0 = 120\ \Omega$.
- Unir el borne **`A (+)`** del adaptador USB-RS485 de la Raspberry Pi con el borne **`A (+)`** del módulo `MAX485` del Arduino.
- Unir el borne **`B (-)`** del adaptador USB-RS485 con el borne **`B (-)`** del módulo `MAX485`.
- Unir la referencia común **`GND`** entre todos los nodos para mantener la tensión de modo común dentro del margen operativo del `MAX485` ($-7\ V$ a $+12\ V$).

- **Instalación de Resistencias Terminadoras ($120\ \Omega$)**:
- Instalar una resistencia $R_{term,1} = 120\ \Omega\ (1/4\ W)$ entre `A(+)` y `B(-)` en el **Extremo Inicial** (adaptador de la Raspberry Pi).
- Instalar una segunda resistencia $R_{term,2} = 120\ \Omega\ (1/4\ W)$ entre `A(+)` y `B(-)` en el **Extremo Final** del cable troncal.
- **Verificación con óhmetro (con el sistema apagado)**: Medir la resistencia equivalente entre `A(+)` y `B(-)`; debe leerse $R_{eq} = 120\ \Omega\ ||\ 120\ \Omega = 60\ \Omega\ \pm 5\%$. **Jamás** colocar resistencias de $120\ \Omega$ en nodos intermedios.

---

5. Etapa 4: Puesta en Marcha del Enlace Maestro-Esclavo Modbus RTU

5.1 Carga del Firmware Esclavo en Arduino (`esclavo_modbus_rtu_arduino.ino`)

- Desconectar temporalmente el pin `D0 (RX)` del módulo `MAX485` mientras se sube el programa por USB al Arduino UNO.
- Compilar y cargar `esclavo_modbus_rtu_arduino.ino`. El firmware configura el puerto serie a **`19200 bps`** con formato **`SERIAL_8E1`** (8 bits de datos, paridad Par, 1 bit de parada) y asigna la dirección **`SLAVE_ID = 1`**.
- Reconectar el pin `D0 (RX)` al pin `RO` del módulo `MAX485`.

5.2 Ejecución del Maestro en Raspberry Pi (`maestro_modbus_rtu_raspberry.py`)

- Activar el entorno virtual Python en la Raspberry Pi e instalar las bibliotecas requeridas:

  ```bash
  source ~/iot_edge_node/venv/bin/activate
  pip install minimalmodbus pyserial
  ```

- Ejecutar el programa maestro:

  ```bash
  python3 maestro_modbus_rtu_raspberry.py --port /dev/ttyUSB0 --slave-id 1
  ```

- Verificar en consola la lectura cíclica de los 6 registros de retención (`0x0000` a `0x0005`) y probar la escritura remota de una nueva consigna térmica con la función **`0x06`**:

  ```bash
  python3 maestro_modbus_rtu_raspberry.py --port /dev/ttyUSB0 --setpoint 26.50
  ```

---

6. Lista de Verificación de Diagnóstico de Fallos

Síntoma Observado	Causa Raíz Probable	Solución de Ingeniería
El último byte de respuesta del Arduino llega corrupto (error de `CRC-16`)	El pin `D4` (`DE` / `/RE`) pasa a `LOW` antes de que el registro de desplazamiento UART termine de emitir los bits físicos	Invocar obligatoriamente `Serial.flush()` más una guarda de `delayMicroseconds(1820)` antes de `digitalWrite(4, LOW)`
Error `No communication with the instrument` en `minimalmodbus`	Líneas diferenciales `A(+)` y `B(-)` invertidas o formato de paridad distinto entre Maestro y Esclavo	Verificar polaridad `A(+)` con `A(+)`, `B(-)` con `B(-)` y confirmar `19200 bps` en formato **`8E1`** en ambos extremos
Sobrecalentamiento de los transceptores `MAX485` y caída de tensión diferencial	Múltiples nodos intermedios tienen instalada su resistencia SMD de $120\ \Omega$ de fábrica ($R_{eq} \ll 60\ \Omega$)	Desoldar la resistencia `R7` ($120\ \Omega$) de todos los módulos `MAX485` ubicados en posiciones intermedias del bus
El Arduino se reinicia cada vez que se ejecuta el script en Python por USB	Pulso negativo en la línea `DTR` a través del condensador de $100\ nF$	Mantener la instancia de puerto abierta dentro de un bucle continuo y aguardar $2.0\ s$ tras abrir el descriptor

Listado de Materiales (BOM) — Módulo 04: Comunicación Directa UART, I²C, SPI, RS-485 y Modbus RTU

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 3, 5, 6, 8, 9, 10 y 14

---

1. Tabla Principal de Componentes de Hardware y Comunicaciones

Ítem	Cant.	Designador	Componente / Descripción Técnica	Especificación Eléctrica / Mecánica	Función en el Módulo 04
01	1	`SBC1`	**Raspberry Pi 4 Model B** (o RPi 5)	SoC ARM64 (`3.3 V` CMOS), puertos USB y cabezal GPIO de 40 pines	Nodo Maestro `Modbus RTU` / Maestro `I²C` y `SPI`
02	1	`U1`	**Arduino UNO R3** (`ATmega328P`)	Lógica TTL de `5.0 V`, reloj de $16\ \text{MHz}$, `UART` hardware (`D0` / `D1`), `I²C` (`A4` / `A5`)	Nodo Esclavo `Modbus RTU` (`ID = 1`) / Coprocesador `I²C` (`0x08`)
03	1	`U2`	**Adaptador Industrial USB a RS-485**	Chip `FT232RL` + `MAX485` (o `CH340E`), protección `TVS` contra transitorios y bornes `A(+)` , `B(-)` , `GND`	Interfaz física `RS-485` en la Raspberry Pi (`/dev/ttyUSB0`)
04	2	`U3, U4`	**Módulo Transceptor RS-485 (`MAX485CSA`)**	Alimentación `5.0 V` , semidúplex hasta $2.5\ \text{Mbps}$, hasta 32 cargas unitarias (`1 UL = 12 kΩ`)	Conversión entre `UART` TTL (`RO` , `DI` , `DE` / `/RE`) y par diferencial `RS-485`
05	1	`U5`	**Conversor Bidireccional de Nivel Lógico `BSS138` (4 Canales)**	4 transistores MOSFET Canal N `BSS138` con resistencias *pull-up* de $10\ \text{k}\Omega$ en `LV` (`3.3 V`) y `HV` (`5.0 V`)	Adaptación bidireccional de colector abierto para buses `I²C` (`SDA` / `SCL`) y `SPI`
06	2	`R_TERM1, R_TERM2`	**Resistencias Terminadoras de Línea RS-485**	$120\ \Omega\ \pm\ 1\%$, película metálica, potencia de $1/4\ \text{W}$ ($0.25\ \text{W}$)	Adaptación de impedancia característica ($Z_0 = 120\ \Omega$) en los 2 extremos físicos del bus
07	1	`R_PD`	**Resistencia Pull-Down de Habilitación TX**	$10\ \text{k}\Omega\ \pm\ 1\%$, $1/4\ \text{W}$	Mantiene los pines `DE` / `/RE` (`Pin D4`) en modo escucha (`0 V`) durante el arranque del AVR
08	2	`R_PU1, R_PU2`	**Resistencias Pull-Up Externas para Bus I²C**	$4.7\ \text{k}\Omega\ \pm\ 1\%$, $1/4\ \text{W}$	Ajuste del tiempo de subida ($\tau = R_p \cdot C_{bus}$) hacia el riel de `3.3 V` si hay múltiples esclavos
09	1	`R1, R2`	**Par de Resistencias para Divisor UART Directo**	$R_1 = 1.8\ \text{k}\Omega$ y $R_2 = 3.3\ \text{k}\Omega$ ($\pm\ 1\%$, $1/4\ \text{W}$)	Atenuación de la línea `TX` de `5.0 V` del Arduino a $3.24\ \text{V}$ hacia `GPIO 15 (RXD)`
10	2	`C_DEC1, C_DEC2`	**Condensadores de Desacoplo Cerámicos**	$100\ \text{nF}$ ($0.1\ \mu\text{F}$), dieléctrico `X7R` , $50\ \text{V}$	Filtrado de picos de corriente en el pin `VCC` de cada transceptor `MAX485`
11	5 m	`CAB1`	**Cable de Par Trenzado Blindado (`STP` / `RS-485`)**	Impedancia característica $Z_0 = 120\ \Omega$, calibre `24 AWG` con malla de drenaje	Línea troncal diferencial para interconexión en cadena (*Daisy-Chain*)

2. Cálculos de Verificación Eléctrica

2.1 Impedancia Equivalente y Corriente en el Bus RS-485

Con únicamente las dos resistencias terminadoras de $120\ \Omega$ instaladas en los extremos inicial y final de la línea troncal:

$$R_{eq} = \frac{R_{term,1} \cdot R_{term,2}}{R_{term,1} + R_{term,2}} = (120\ \Omega \cdot 120\ \Omega) / (120\ \Omega + 120\ \Omega) = 60\ \Omega$$

Para una tensión diferencial típica de salida $V_{OD} = 2.0\ V$ entregada por el transmisor `MAX485`:

- **Corriente en el bucle diferencial**: $I_{bus} = (2.0\ V) / (60\ \Omega) = 33.3\ mA$ (dentro del límite nominal de $60\ mA$ del `MAX485`).
- **Potencia disipada en cada terminador**: $P_{term} = ((2.0\ V)^2) / (120\ \Omega) = 33.3\ mW \ll 250\ mW$ ($1/4\ W$).

2.2 Tiempo de Carácter y Silencio de Trama Modbus RTU a 19200 bps (`8E1`)

Cada carácter en formato `8E1` consta de $11\ bits$ ($1\ Start + 8\ Datos + 1\ Paridad\ Par + 1\ Stop$):

$$T_{char} = (11\ bits) / (19200\ bits/s) = 572.92\ \mu s \quad\Longrightarrow\quad T_{3.5\ char} = 3.5 \times 572.92\ \mu s \approx 2005\ \mu s \approx 2.0\ ms$$

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

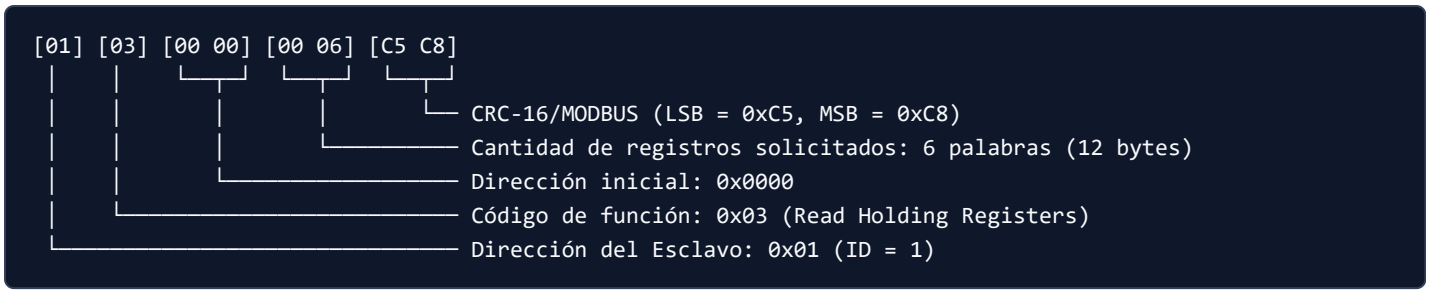
3. Mapa Oficial de Registros de Retención del Nodo de Invernadero (`ID = 1`)

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**:

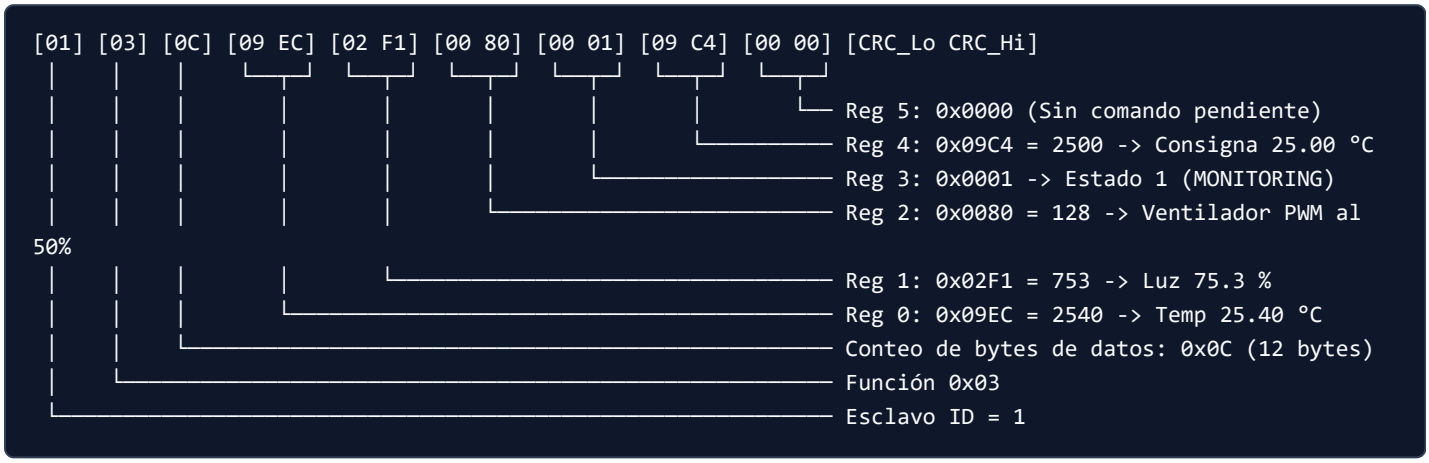
Dirección Hex (Dec)	Nombre Simbólico	Tipo de Dato	Factor de Escala	Acceso	Descripción y Conversión en la Raspberry Pi
`0x0000` (`0`)	`REG_TEMP_C_X100`	`int16_t`	$\times 100$	Solo Lectura (`0x03`)	Temperatura filtrada EMA en centésimas de °C ($T = Valor / 100.0$, ej. `2540` $\rightarrow 25.40$ ^\circC)
`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)



Planos y Esquemáticos de Circuito — Módulo 04: Enlaces UART, I²C (`BSS138`), RS-485 (`MAX485`) y Trama Modbus RTU

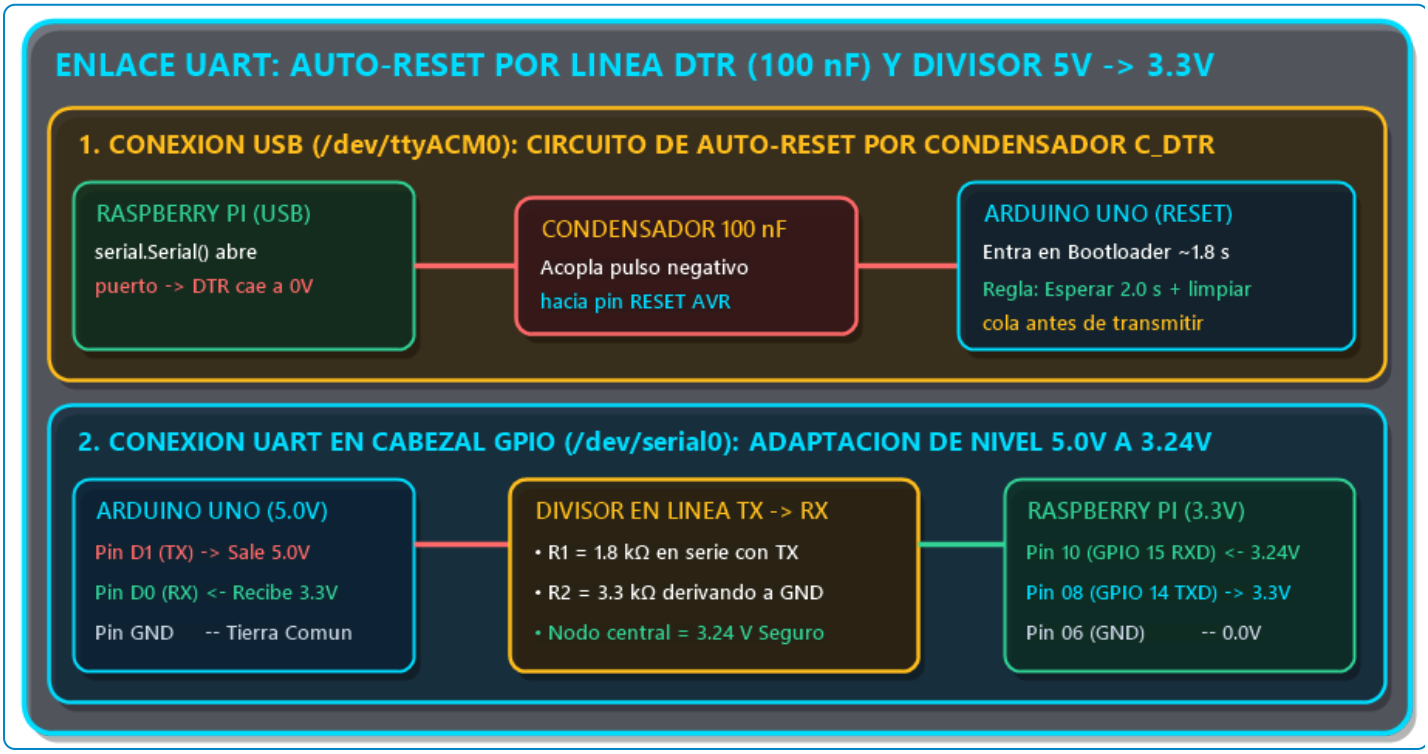
Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 3, 5, 8, 9, 10 y 12

---

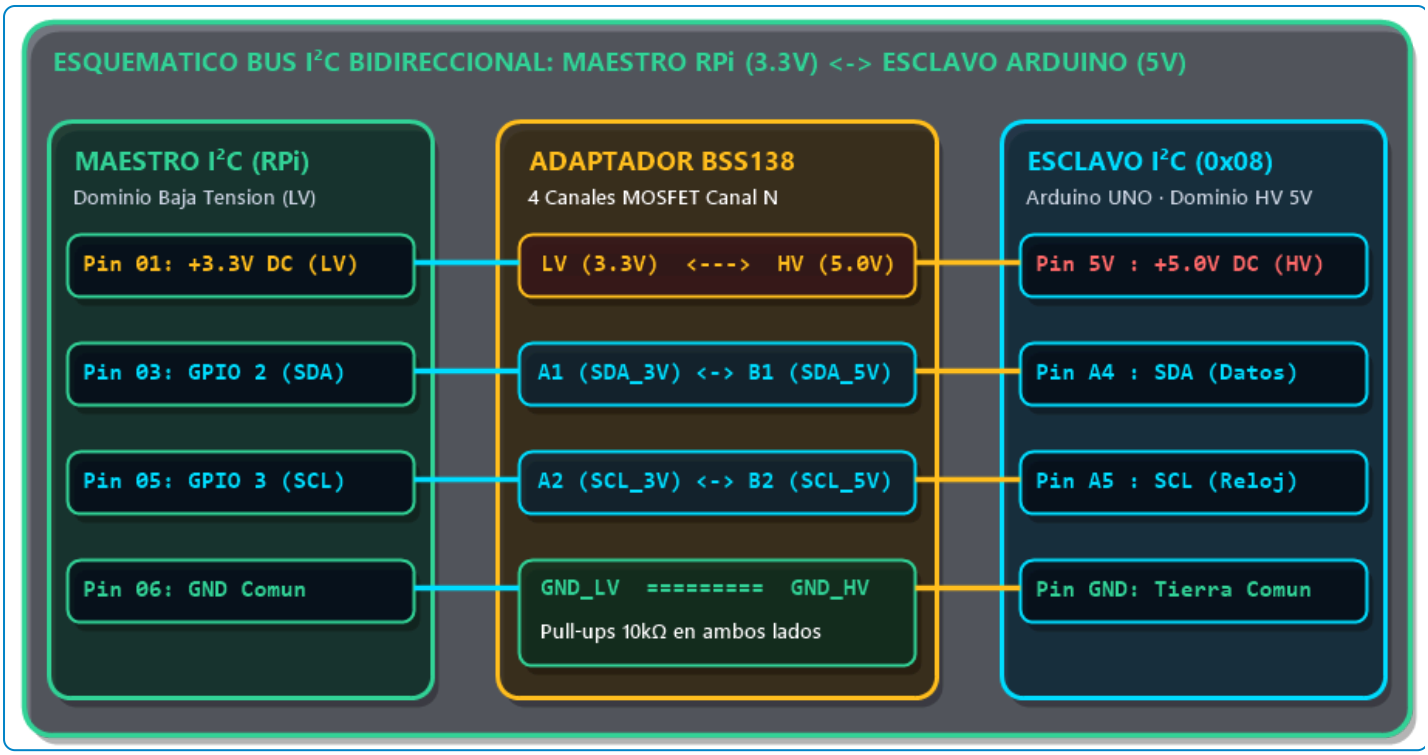
1. Plano 1: Enlace UART, Circuito de Auto-Reset por Línea DTR ($100\ \text{nF}$) y Divisor Resistivo $5\ \text{V} \rightarrow 3.3\ \text{V}$



Origen (Arduino UNO `5.0 V`)	Etapa Intermedia de Acondicionamiento	Destino (Raspberry Pi `3.3 V`)	Nivel Eléctrico Resultante
Puerto USB (`ATmega16U2` / `CH340`, línea `DTR`)	Condensador cerámico en serie $C_{DTR} = 100\ \text{nF}$ hacia pin `RESET`	Puerto USB Host (`/dev/ttyACM0`)	Pulso bajo al abrir el puerto → espera obligatoria de $2.0\ \text{s}$
Pin `D1 (TX)` (`5.0 V` TTL)	Divisor resistivo: $R_1 = 1.8\ \text{k}\Omega$ en serie + $R_2 = 3.3\ \text{k}\Omega$ a `GND`	Pin Físico 10 (`GPIO 15 / RXD0`)	$V_{RX} = 5.0\ \text{V} \cdot (3.3) / (1.8 + 3.3) = 3.24\ \text{V}$ (Seguro)
Pin `D0 (RX)` ($V_{IH,mín} = 3.0\ \text{V}$)	Conexión directa (o vía `BSS138`)	Pin Físico 08 (`GPIO 14 / TXD0`)	$3.30\ \text{V} > 3.00\ \text{V}$ (Reconocido como `'1'` lógico)
Pin `GND`	Conductor directo de retorno	Pin Físico 06 (`GND`)	Referencia equipotencial de $0.0\ \text{V}$

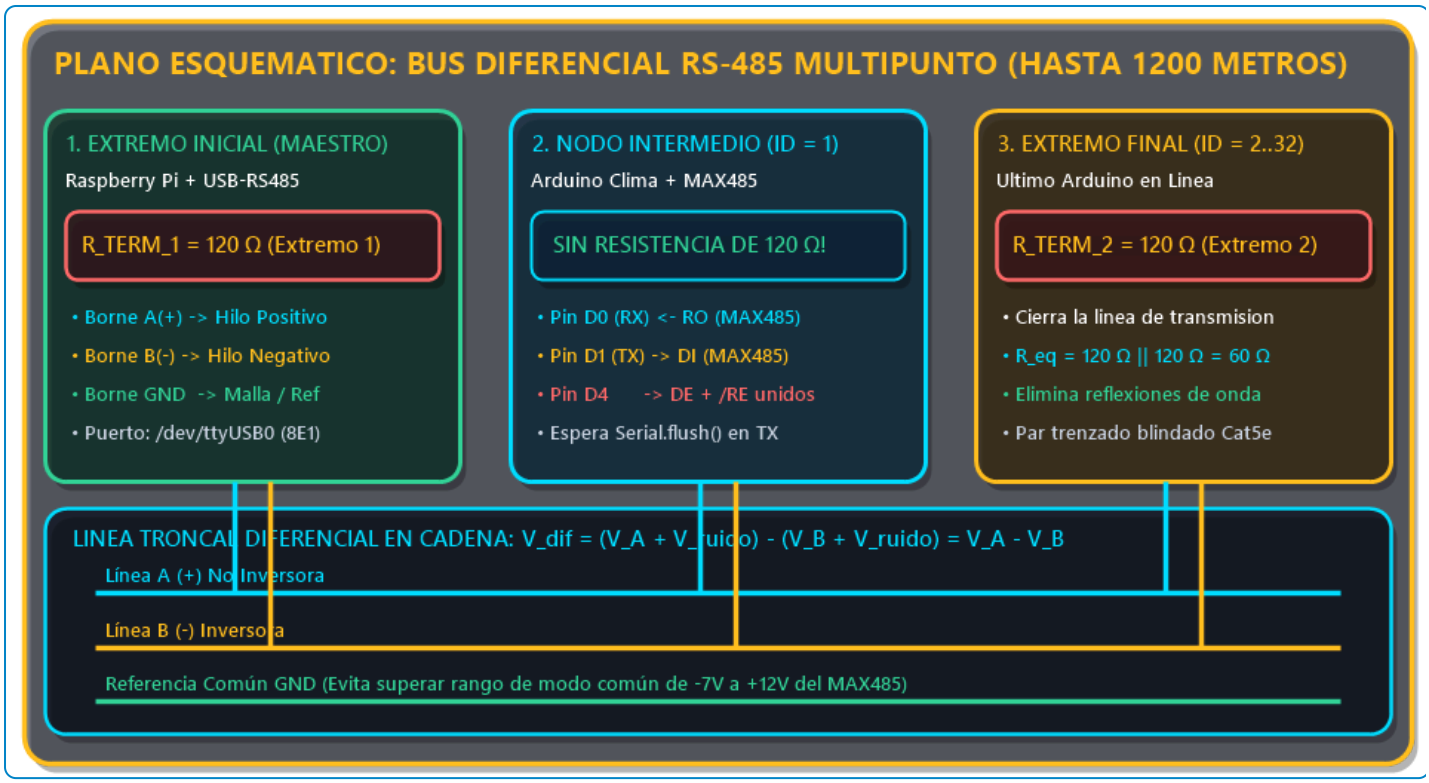
---

2. Plano 2: Esquemático del Bus I²C Bidireccional con Adaptador MOSFET `BSS138`



Señal del Bus	Maestro Raspberry Pi (`3.3 V`)	Conversor `BSS138` (4 Canales)	Esclavo Arduino UNO (`0x08`, `5.0 V`)	Nota de Configuración
Referencia Baja (`LV`)	Pin Físico 01 (`+3.3V DC`)	Pin `LV` (Surtidor + Compuerta MOSFET)	—	Polariza las resistencias *pull-up* de $10\ \text{k}\Omega$ del lado `3.3 V`
Referencia Alta (`HV`)	—	Pin `HV` (Drenador vía *pull-up* $10\ \text{k}\Omega$)	Pin `5V` (`+5.0V DC`)	Polariza el dominio de `5.0 V`
Datos Serie (`SDA`)	Pin Físico 03 (`GPIO 2 / SDA1`)	Canal 1: `A1` (`3.3 V`) $\leftrightarrow$ `B1` (`5.0 V`)	Pin `A4 (SDA)`	Desactivar *pull-up* interno en AVR con `digitalWrite(SDA, LOW)`
Reloj Serie (`SCL`)	Pin Físico 05 (`GPIO 3 / SCL1`)	Canal 2: `A2` (`3.3 V`) $\leftrightarrow$ `B2` (`5.0 V`)	Pin `A5 (SCL)`	Desactivar *pull-up* interno en AVR con `digitalWrite(SCL, LOW)`
Tierra Común (`GND`)	Pin Físico 06 (`GND`)	Pines `GND` (`LV` y `HV` unidos)	Pin `GND`	Indispensable para referencia de tensión compuerta-surtidor (V_{GS})

3. Plano 3: Bus Diferencial Industrial RS-485 Multipunto (`MAX485`) hasta $1200\ \text{m}$



Pin del Transceptor `MAX485`	Nombre Técnico	Conexión en el Nodo Esclavo (Arduino UNO)	Conexión en la Línea Troncal RS-485
Pin 1 (`RO` **)**	*Receiver Output* (Salida del Receptor)	Pin `D0 (RX)` del Arduino UNO	—
Pin 2 (`/RE` **)**	*Receiver Enable* (Activo en Bajo)	Unido a `DE` y conectado al **Pin** `D4` (con *pull-down* de $10\ \text{k}\Omega$ a `GND`)	—
Pin 3 (`DE` **)**	*Driver Enable* (Activo en Alto)	Unido a `/RE` y conectado al **Pin** `D4` (`HIGH` = Transmitir, `LOW` = Escuchar)	—
Pin 4 (`DI` **)**	*Driver Input* (Entrada del Transmisor)	Pin `D1 (TX)` del Arduino UNO	—
Pin 5 (`GND` **)**	Tierra de Referencia	Pin `GND` del Arduino UNO	Hilo de referencia común / malla del cable blindado
Pin 6 (`A` **)**	Línea Diferencial No Inversora `A (+)`	—	Hilo `A (+)` del par trenzado ($R_{term} = 120\ \Omega$ entre `A` y `B` solo en los 2 extremos)
Pin 7 (`B` **)**	Línea Diferencial Inversora `B (-)`	—	Hilo `B (-)` del par trenzado
Pin 8 (`VCC` **)**	Alimentación Positiva (`+5.0 V`)	Pin `5V` del Arduino UNO + condensador cerámico de $100\ \text{nF}$ a `GND`	—

4. Plano 4: Estructura de Trama Binaria y Mapa de Registros `Modbus RTU`



Consulte el documento `MANUAL_PROTOCOLO_MODBUS_RTU_Y_CRC16.pdf` para el detalle bit a bit de los 6 registros de retención (`0x0000` a `0x0005`) y las secuencias de bytes de las funciones `0x03` y `0x06` .