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