

Universidad Nacional de Colombia  
Sede Bogotá

Sistemas Embebidos

2025-2

## **Sistema de monitoreo y control de invernaderos con Milk Duo 256 y LoRa (RoseWatch)**

### **Integrantes:**

Carlos Felipe Betin Escobar  
Manuel Vasquez  
Sebastian Pantoja

**Profesor:** Carlos Camargo

Bogotá D.C.  
11 de diciembre de 2025

# Índice

<b>Resumen</b>	<b>3</b>
<b>1. Introducción</b>	<b>4</b>
<b>2. Planteamiento del problema y objetivos</b>	<b>4</b>
2.1. Planteamiento del problema . . . . .	4
2.2. Justificación . . . . .	5
2.3. Objetivo general . . . . .	5
2.4. Objetivos específicos . . . . .	6
<b>3. Marco teórico</b>	<b>6</b>
3.1. Control ambiental en invernaderos de rosas . . . . .	6
3.2. Sensores agroclimáticos utilizados . . . . .	7
3.2.1. DHT11: temperatura y humedad del aire . . . . .	7
3.2.2. BH1750: iluminancia . . . . .	7
3.2.3. HW-080: humedad del suelo . . . . .	7
3.3. Tecnología LoRa y comunicaciones de largo alcance . . . . .	8
3.4. Plataforma principal: Milk-V Duo 256M y sistemas embebidos . . . . .	9
<b>4. Metodología de diseño (alineada con la rúbrica)</b>	<b>10</b>
4.1. Especificación del sistema . . . . .	10
4.1.1. Restricciones y requisitos . . . . .	10
4.1.2. Funciones del sistema . . . . .	11
4.1.3. Públicos objetivo y criterios de aceptación . . . . .	11
4.1.4. Evaluación de alternativas . . . . .	11
4.2. Modelo del sistema . . . . .	12
4.2.1. Bloques constructores básicos . . . . .	12
4.2.2. Modelado funcional . . . . .	12
4.2.3. Validación y crítica del modelo . . . . .	13
4.3. Arquitectura del sistema . . . . .	13
4.3.1. División hardware/software y arquitectura global . . . . .	13
4.3.2. Arquitectura de comunicaciones . . . . .	13
4.3.3. Arquitectura de software . . . . .	14
4.3.4. Evaluación de arquitecturas alternativas . . . . .	14
4.4. Nodo de sensado y firmware de transmisión LoRa . . . . .	14
4.4.1. Montaje de hardware del nodo de sensado . . . . .	15
4.4.2. Firmware de transmisión en Arduino UNO . . . . .	15
4.4.3. Pruebas de transmisión con monitor serie . . . . .	16
4.5. Desarrollo de software del sistema embebido . . . . .	17
4.6. Prototipado y pruebas . . . . .	17
4.6.1. Implementación del hardware . . . . .	17
4.6.2. Implementación del firmware y software . . . . .	17
4.6.3. Plan de pruebas . . . . .	18
4.6.4. Resultados de las pruebas . . . . .	18
4.7. Desarrollo de software del sistema embebido . . . . .	18
4.8. Objetivos del desarrollo de software . . . . .	18
4.9. Selección del repositorio y estrategia de compilación . . . . .	19
4.10. Preparación del entorno de compilación . . . . .	19
4.10.1. Sistema anfitrión y contenedor . . . . .	19
4.10.2. Carga del entorno de compilación . . . . .	20

4.11. Configuración para la placa Milk-V Duo 256M . . . . .	20
4.11.1. Selección del <code>defconfig</code> adecuado . . . . .	20
4.12. Construcción de la imagen del sistema . . . . .	22
4.12.1. Ejecución del script de construcción . . . . .	22
4.12.2. Localización de la imagen generada . . . . .	25
4.12.3. Grabación en microSD y validación básica . . . . .	25
4.12.4. Personalización de la imagen . . . . .	25
4.13. Configuración y Compilación del SDK para Milk-V Duo 256M . . . . .	25
4.14. Toolchain y compilador . . . . .	31
4.14.1. Integración de librerías y resolución de dependencias . . . . .	32
4.14.2. Transferencia y despliegue de ejecutables en la Milk-V Duo . . . . .	33
4.15. Errores encontrados y lecciones aprendidas . . . . .	34
4.15.1. Relación con los objetivos del proyecto . . . . .	35
<b>5. Prototipado y pruebas</b>	<b>35</b>
5.1. Prototipado de hardware . . . . .	35
5.1.1. Nodo de sensado . . . . .	35
5.1.2. Pasarela con Milk-V Duo 256M . . . . .	36
5.2. Prototipado de software y pruebas del nodo de sensado . . . . .	38
5.3. Pruebas de la interfaz SPI y del módulo LoRa en la pasarela . . . . .	39
5.4. Síntesis de resultados de las pruebas . . . . .	39
5.4.1. Implementación del firmware y software . . . . .	40
5.4.2. Plan de pruebas . . . . .	43
5.4.3. Resultados de las pruebas . . . . .	44
<b>6. manejo de información</b>	<b>46</b>
<b>7. Resultados y análisis</b>	<b>46</b>
<b>8. Conclusiones y trabajo futuro</b>	<b>47</b>
8.1. Conclusiones . . . . .	47
8.2. Trabajo futuro . . . . .	48
<b>A. Código fuente Arduino (transmisor)</b>	<b>49</b>
<b>B. Código fuente receptor / interfaz</b>	<b>51</b>

## Resumen

**Palabras clave:** sistemas embebidos, LoRa, invernaderos, monitoreo remoto.

## 1. Introducción

El cultivo de rosas para exportación exige un control cuidadoso de las condiciones ambientales dentro del invernadero. Variables como la temperatura, la humedad relativa, la radiación y la humedad del suelo influyen directamente en el crecimiento, el tamaño y la calidad de la flor. En particular, las heladas que se presentan durante la madrugada representan un riesgo crítico. La caída brusca de temperatura puede afectar de forma irreversible el estado de las plantas y comprometer la producción de toda una temporada.

La idea de este proyecto surgió a partir de la experiencia cercana de uno de los integrantes del grupo, quien conoce a un productor de rosas para exportación que se enfrenta precisamente a este problema. Actualmente, el productor no cuenta con una forma confiable de monitorear la temperatura interna de los invernaderos ni de anticipar eventos de helada, especialmente en horarios en los que no es posible estar físicamente presente. Esta situación evidencia la necesidad de una solución tecnológica que permita vigilar el microclima del invernadero y reaccionar a tiempo ante condiciones adversas.

A partir de esta problemática se diseñó RoseWatch, un sistema de monitoreo agroclimático para invernaderos de rosas que registra variables ambientales clave y las transmite de forma inalámbrica a largas distancias, sin depender de conexión a internet. El sistema se basa en una red de nodos de sensado que utilizan módulos LoRa para enviar la información hacia una unidad central de procesamiento, construida alrededor de la placa Milk-V Duo 256M, que actúa como pasarela (gateway), procesa los datos y los pone a disposición del usuario a través de una interfaz accesible de manera remota.

Con esta arquitectura, RoseWatch busca ofrecer al productor de rosas una herramienta práctica para visualizar el comportamiento térmico del invernadero, detectar con anticipación posibles heladas y tomar decisiones informadas sobre acciones de protección (por ejemplo, encendido de calefactores o cierre de ventilaciones). Además de atender una necesidad concreta en un contexto real, el proyecto permite aplicar de manera integrada conceptos de sistemas embebidos, comunicaciones inalámbricas de baja potencia y diseño de soluciones tecnológicas orientadas al sector agroindustrial.

Finalmente, este documento se organiza de la siguiente manera. En la sección de *Planteamiento del problema y objetivos* se describe con detalle la situación actual, la justificación del proyecto y las metas específicas que se persiguen. A continuación, en el *Marco teórico* se presentan los conceptos necesarios sobre control ambiental en invernaderos, sensores agroclimáticos, tecnología LoRa y la plataforma Milk-V Duo 256M. Luego, en la sección de *Metodología de diseño* se expone la especificación del sistema, el modelo, la arquitectura propuesta y el proceso de prototipado, incluyendo la creación de la imagen Linux para la placa principal. Posteriormente, en *Resultados y análisis* se discute el desempeño del sistema en las pruebas realizadas. Por último, en la sección de *Conclusiones y trabajo futuro* se resumen los principales hallazgos y se proponen posibles mejoras, mientras que en los anexos se recopilan códigos fuente, esquemáticos y datos experimentales.

## 2. Planteamiento del problema y objetivos

### 2.1. Planteamiento del problema

El cultivo de rosas para exportación exige mantener condiciones ambientales controladas dentro del invernadero, especialmente en lo relacionado con la temperatura, la humedad relativa, la radiación y la humedad del suelo. En la práctica, estas variables pueden variar de forma significativa a lo largo del día y de la noche, y en particular durante la madrugada, cuando se presentan descensos bruscos de temperatura asociados a heladas. Estos eventos pueden ocasionar daños severos en los tejidos de la planta, afectar el tamaño y la calidad de la flor y comprometer la producción de una o varias camas de cultivo.

En el caso específico que motiva este proyecto, un productor de rosas para exportación realiza el monitoreo de la temperatura de manera principalmente manual y puntual, basado en la experiencia y en mediciones ocasionales dentro del invernadero. Este enfoque presenta varias limitaciones: no permite tener un registro continuo de las condiciones agroclimáticas, dificulta anticipar eventos de helada que ocurren en la madrugada, y obliga al productor a estar físicamente presente o a desplazarse en horarios en los que no siempre es posible. Como consecuencia, las decisiones de protección (por ejemplo, encender calefactores, cerrar ventilaciones o en último caso aplicar agua dentro de la rosa mientras se nivela la temperatura) se toman de forma reactiva y no preventiva.

Adicionalmente, las soluciones tecnológicas disponibles comercialmente para monitoreo remoto suelen depender de conexión a internet, infraestructura de comunicaciones costosa o plataformas cerradas de difícil adaptación a las necesidades de pequeños y medianos productores. Esto deja un vacío importante: no existe en el contexto del productor considerado una herramienta accesible que permita monitorear de forma continua el microclima del invernadero, visualizar la información a distancia y recibir indicadores tempranos de riesgo de heladas sin depender de redes de datos convencionales.

En síntesis, el problema central que aborda este proyecto es la falta de un sistema embebido de monitoreo agroclimático, de bajo costo y con comunicaciones de largo alcance, que permita al productor de rosas anticiparse a eventos de helada y tomar decisiones oportunas de manejo, aun cuando se encuentre a grandes distancias del invernadero y sin disponibilidad de internet.

## 2.2. Justificación

La floricultura orientada a la exportación es una actividad de alta sensibilidad frente a las condiciones ambientales, pues la calidad del producto final depende de parámetros como el tamaño del botón, la longitud del tallo y la ausencia de daños por frío. Pequeñas variaciones de temperatura durante ventanas críticas del desarrollo de la planta pueden traducirse en pérdidas económicas significativas para el productor. Contar con un sistema que permita monitorear el entorno del cultivo de manera continua y a distancia se convierte, por tanto, en un factor clave para reducir riesgos y mejorar la toma de decisiones.

Desde la perspectiva tecnológica, el uso de comunicaciones de largo alcance y baja potencia como LoRa ofrece una alternativa atractiva frente a soluciones basadas exclusivamente en WiFi o datos móviles, especialmente en zonas rurales donde la cobertura de internet es limitada o costosa. Al combinar nodos de sensado instalados en el invernadero con una unidad central de procesamiento basada en la placa **Milk-V Duo 256M**, es posible construir una solución que no dependa de infraestructura de red compleja y que, al mismo tiempo, permita la visualización remota de la información y la generación de alertas.

Académicamente, el proyecto se justifica porque permite integrar de manera aplicada conceptos de sistemas embebidos, comunicaciones inalámbricas, adquisición de datos, diseño de software y desarrollo de interfaces de usuario. Además, favorece el trabajo con herramientas y plataformas contemporáneas (Linux embebido en la Milk-V Duo 256M, enlaces LoRa, servidores web ligeros), lo cual contribuye al desarrollo de competencias relevantes para la ingeniería orientada al sector agroindustrial.

## 2.3. Objetivo general

Diseñar e implementar un sistema embebido de monitoreo agroclimático denominado **RoseWatch** para invernaderos de rosas destinadas a la exportación, que permita medir variables ambientales clave y transmitirlas mediante comunicación LoRa hacia una unidad central basada en la placa **Milk-V Duo 256M**, con el fin de visualizar la información de forma remota y apoyar la toma de decisiones oportunas frente a eventos de helada.

## **2.4. Objetivos específicos**

- Caracterizar las condiciones agroclimáticas relevantes del invernadero de rosas (rangos de temperatura, humedad relativa, radiación y humedad del suelo) e identificar los umbrales críticos asociados a riesgo de heladas.
- Seleccionar y dimensionar los sensores adecuados para la medición de las variables ambientales de interés, considerando precisión, rango de operación y condiciones propias del invernadero.
- Diseñar la arquitectura hardware y software del sistema RoseWatch, definiendo la estructura de los nodos de sensado, el enlace LoRa y la unidad central basada en la placa Milk-V Duo 256M.
- Desarrollar y documentar la imagen Linux y el entorno de ejecución necesario en la Milk-V Duo 256M para soportar la recepción de datos, el procesamiento básico y la ejecución de un servidor web de visualización.
- Implementar el firmware de los nodos de sensado para la adquisición de datos y su transmisión mediante módulos LoRa hacia la unidad central.
- Diseñar e implementar una interfaz de usuario que permita visualizar de manera remota las variables agroclimáticas medidas, así como identificar tendencias y posibles eventos de helada.
- Realizar pruebas de funcionamiento del sistema en un escenario representativo, evaluando el desempeño del enlace LoRa, la continuidad del registro de datos y la capacidad del sistema para detectar oportunamente condiciones de riesgo.

## **3. Marco teórico**

### **3.1. Control ambiental en invernaderos de rosas**

El uso de invernaderos en la floricultura de rosas permite modificar y controlar el microclima donde se desarrollan las plantas, con el objetivo de garantizar una producción estable y una calidad adecuada para la exportación. Entre las variables ambientales más relevantes se encuentran la temperatura del aire, la humedad relativa, la radiación (o intensidad de luz) y la humedad del suelo. Estas variables influyen directamente en procesos fisiológicos como la fotosíntesis, la transpiración, el crecimiento vegetativo y la formación del botón floral.

En el caso de las rosas de corte, la literatura reporta que existen rangos de temperatura considerados óptimos para el crecimiento y la calidad del producto; desviaciones por debajo de dichos rangos, especialmente durante la noche o la madrugada, pueden generar estrés por frío y daños en tejidos, mientras que temperaturas excesivamente altas afectan la fotosíntesis y reducen la vida en florero. De forma similar, un control deficiente de la humedad relativa puede favorecer el desarrollo de enfermedades fúngicas o, en el extremo opuesto, causar estrés hídrico y pérdida de turgencia.

La radiación incidente determina, junto con la temperatura, la tasa fotosintética y la distribución de la biomasa en la planta. Un nivel de luz insuficiente puede prolongar demasiado el ciclo de producción o generar tallos débiles; por el contrario, una radiación excesiva puede causar sobrecalentamiento del invernadero. Finalmente, la humedad del suelo condiciona la disponibilidad de agua y nutrientes; su manejo inadecuado se traduce en problemas de asfixia radicular o estrés por sequía.

En este contexto, disponer de un sistema que mida de manera continua estas variables en puntos estratégicos del invernadero proporciona información clave para planear la ventilación, el



Figura 1: Invernadero de rosas en donde se vive la problemática.

sombreado, el riego y, en particular, las estrategias de protección contra heladas. La detección temprana de descensos de temperatura en la madrugada permite activar mecanismos de mitigación antes de que se produzcan daños irreversibles en el cultivo.

### 3.2. Sensores agroclimáticos utilizados

El nodo de sensado de RoseWatch se basa en un Arduino UNO que integra tres sensores principales para medir las variables agroclimáticas de interés: temperatura y humedad relativa del aire, iluminancia y humedad del suelo. Estos sensores entregan señales digitales o analógicas que el microcontrolador adquiere y empaqueta para su envío mediante LoRa hacia la unidad central.

#### 3.2.1. DHT11: temperatura y humedad del aire

El DHT11 es un sensor digital sencillo y económico que mide temperatura y humedad relativa del aire en un solo encapsulado. Se comunica con el Arduino a través de una línea de datos única y una librería específica se encarga de gestionar el protocolo. Aunque su precisión es moderada, resulta suficiente para el monitoreo general del microclima del invernadero en un prototipo de bajo costo.

#### 3.2.2. BH1750: iluminancia

El BH1750 es un sensor de luz que mide la iluminancia en lux y se comunica mediante el bus I<sup>2</sup>C. Entrega directamente valores numéricos de intensidad lumínica, lo que simplifica el procesamiento en el Arduino. En RoseWatch se utiliza para conocer la cantidad de luz disponible en el interior del invernadero y relacionarla con la temperatura y el desarrollo del cultivo.

#### 3.2.3. HW-080: humedad del suelo

El HW-080 es un sensor de humedad de suelo con salida analógica. Su señal se lee mediante el conversor analógico-digital del Arduino y se transforma, mediante una calibración sencilla, en un porcentaje relativo de humedad. Esta información sirve como referencia para evitar tanto el exceso de riego como el estrés por falta de agua en las raíces de las rosas.

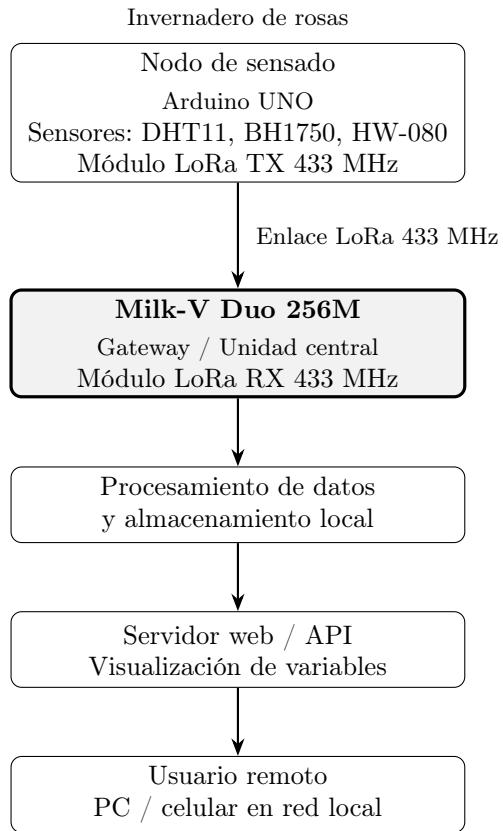


Figura 2: Diagrama de bloques vertical del sistema RoseWatch. La placa Milk-V Duo 256M actúa como unidad central, recibiendo datos del nodo de sensado vía LoRa y ofreciendo procesamiento, almacenamiento y visualización remota.

### 3.3. Tecnología LoRa y comunicaciones de largo alcance

LoRa (acrónimo de *Long Range*) es una tecnología de comunicación inalámbrica diseñada para enlaces de largo alcance y bajo consumo de energía, enmarcada dentro de las denominadas redes de área amplia de baja potencia (LPWAN). A nivel físico, LoRa emplea una modulación basada en *Chirp Spread Spectrum* (CSS), en la cual la información se codifica mediante señales tipo *chirp* cuya frecuencia varía de forma controlada en el tiempo. Este esquema confiere a LoRa una alta inmunidad al ruido y la posibilidad de alcanzar distancias de varios kilómetros en entornos rurales, utilizando bandas de frecuencia libres de licencia en el rango sub-GHz.

Los dispositivos LoRa operan típicamente en bandas ISM como 433 MHz, 868 MHz o 915 MHz, dependiendo de la región, y permiten ajustar parámetros como el *spreading factor*, el ancho de banda y la tasa de codificación. El *spreading factor* controla el compromiso entre alcance y velocidad de transmisión: valores altos incrementan la sensibilidad del receptor y el alcance del enlace, a costa de reducir el caudal de datos, mientras que valores más bajos permiten mayores tasas de transmisión con menor alcance efectivo.

En el proyecto RoseWatch se emplea un módulo basado en el transceptor LoRa **LORA\_1278**, que opera en la banda de 433 MHz y se conecta al Arduino UNO mediante la interfaz SPI y un pin de interrupción digital para la señal de *IRQ*. El firmware del nodo de sensado configura el módulo con una potencia de transmisión de 14 dBm, un *spreading factor* alto (SF12), un ancho de banda de 62,5 kHz y una tasa de codificación 4/8, parámetros que privilegian el alcance y la robustez de la comunicación sobre la velocidad de transmisión de datos. :contentReference[oaicite:4]index=4

Sobre la capa física LoRa se puede construir una red bajo la especificación LoRaWAN, que define una arquitectura de tipo estrella con múltiples nodos finales y uno o varios *gateways*. Sin embargo, en RoseWatch se utiliza un enlace punto a punto (LoRa P2P) entre el nodo de sensado

y la unidad central basada en la Milk-V Duo 256M. Este enfoque simplifica el diseño, reduce la complejidad del protocolo y es suficiente para las necesidades de transmisión de datos de un único invernadero o de un conjunto reducido de nodos.

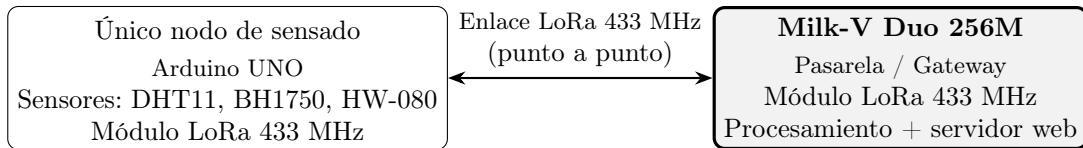


Figura 3: Esquema conceptual del enlace LoRa punto a punto entre el único nodo de sensado del prototipo (Arduino + sensores) y la pasarela basada en la Milk-V Duo 256M.

### 3.4. Plataforma principal: Milk-V Duo 256M y sistemas embebidos

La plataforma Milk-V Duo 256M es una tarjeta de desarrollo embebida de tamaño reducido, basada en el sistema en chip (SoC) SG2002. Esta placa integra un conjunto de núcleos de procesamiento que combinan arquitecturas RISC-V y ARM, junto con una unidad de procesamiento neuronal (NPU) y memoria DDR de 256 MB, lo que le permite ejecutar sistemas operativos como Linux y, en paralelo, sistemas de tiempo real (RTOS) en subsistemas específicos. Además, incluye una ranura para tarjeta microSD, interfaces de comunicación (UART, SPI, I<sup>2</sup>C, Ethernet), pines de propósito general (GPIO) y un conector USB-C para alimentación y programación.

En términos de diseño de sistemas embebidos, la Milk-V Duo 256M se sitúa en un punto intermedio entre microcontroladores de propósito general (como los utilizados en placas tipo Arduino) y computadores de placa reducida más potentes (como los basados en procesadores de aplicaciones). Su capacidad para ejecutar Linux la convierte en una plataforma adecuada para tareas que requieren manejo de archivos, ejecución de servidores web ligeros, conectividad de red y ejecución de múltiples procesos, manteniendo al mismo tiempo una huella de hardware y un consumo relativamente moderados.

En el contexto de RoseWatch, la Milk-V Duo 256M cumple el rol de *unidad central de procesamiento* o pasarela (*gateway*). Sobre ella se despliega una imagen Linux personalizada que incluye:

- Los componentes de arranque necesarios (FSBL, OpenSBI y U-Boot) adaptados al SoC SG2002.
- Un kernel Linux configurado para habilitar las interfaces requeridas (SPI, UART, GPIO) que permiten la comunicación con el módulo LoRa receptor y otros periféricos.
- Un sistema de archivos raíz con las bibliotecas y herramientas necesarias para ejecutar scripts de recepción de datos LoRa y un servidor web de visualización.

La arquitectura escogida permite separar responsabilidades: los nodos de sensado, basados en microcontroladores como el Arduino UNO, se enfocan en la adquisición de datos de los sensores DHT11, BH1750 y HW-080 y en su transmisión mediante LoRa; la Milk-V Duo 256M se encarga de recibir los datos, almacenarlos, procesarlos y exponerlos al usuario mediante una interfaz accesible a través de una red local o, eventualmente, otros mecanismos de conectividad. Este enfoque es consistente con las tendencias actuales en sistemas embebidos para Internet de las Cosas (IoT), donde nodos de borde simples se combinan con *gateways* más potentes que ejecutan sistemas operativos completos.

En suma, el marco teórico descrito proporciona las bases conceptuales para el diseño de RoseWatch: la comprensión del microclima en invernaderos de rosas, la selección de sensores específicos (DHT11, BH1750 y HW-080) para medir las variables clave, el uso de comunicaciones de largo alcance de bajo consumo mediante LoRa a 433 MHz y la integración de una plataforma embebida capaz de ejecutar Linux y servir como núcleo del sistema de monitoreo.

## 4. Metodología de diseño (alineada con la rúbrica)

La metodología de diseño de RoseWatch se organizó siguiendo cinco ejes principales, coherentes con la rúbrica del curso: (i) especificación del sistema, (ii) modelo del sistema, (iii) arquitectura, (iv) prototipado y pruebas, y (v) documentación. En cada etapa se tomaron decisiones técnicas y se evaluaron alternativas, integrando tanto consideraciones de ingeniería como restricciones propias del contexto real del productor de rosas.

### 4.1. Especificación del sistema

#### 4.1.1. Restricciones y requisitos

El punto de partida fue la identificación de las restricciones físicas, de entorno y de uso asociadas al caso real:

- **Ubicación y conectividad:** el invernadero se encuentra en un contexto rural con conectividad a internet limitada o intermitente. Se asumió que el sistema debe funcionar sin depender de acceso permanente a la red.
- **Distancia al productor:** el productor no siempre puede estar físicamente en el invernadero, por lo que se requiere un enlace de comunicación de *largo alcance* entre el invernadero y la unidad de visualización.
- **Condiciones ambientales:** alta humedad, cambios de temperatura y presencia de polvo y agua, lo que restringe la selección de sensores y la forma de encapsularlos.
- **Presupuesto y complejidad:** el sistema debe ser de bajo costo relativo y con una complejidad acorde a la capacidad de mantenimiento del usuario final.

A partir de estas restricciones se definieron los siguientes **requisitos funcionales**:

- Medir temperatura del aire, humedad relativa, iluminancia y humedad del suelo en el interior del invernadero.
- Enviar periódicamente las lecturas desde el invernadero hacia una unidad central mediante un enlace inalámbrico de largo alcance (LoRa).
- Recibir, almacenar y procesar las mediciones en la unidad central, generando un histórico consultable.
- Visualizar las variables en una interfaz accesible desde un PC o dispositivo móvil dentro de la red local del usuario.
- Permitir la incorporación futura de más nodos de sensado sin rediseñar por completo el sistema.

Y los principales **requisitos no funcionales**:

- Funcionamiento continuo con un ciclo de muestreo del orden de decenas de segundos, sin intervención constante del usuario.
- Robustez frente a fallos de comunicación puntuales (no se debe bloquear el sistema si se pierde un paquete).
- Consumo energético moderado en el nodo de sensado, permitiendo su futura alimentación con batería o energía solar.
- Utilización de tecnologías abiertas o ampliamente documentadas (Arduino, Linux embebido, LoRa), que faciliten el mantenimiento y la replicabilidad académica.

#### 4.1.2. Funciones del sistema

Con base en los requisitos se definieron las funciones principales de RoseWatch:

1. **Adquisición:** leer periódicamente las variables agroclimáticas mediante los sensores DHT11, BH1750 y HW-080 conectados al Arduino.
2. **Empaquetamiento y transmisión:** construir una trama con las lecturas y transmitirla por LoRa a 433 MHz desde el nodo de sensado al gateway.
3. **Recepción y almacenamiento:** recibir la trama LoRa en la Milk-V Duo 256M, decodificarla y almacenarla localmente (archivo o base de datos ligera).
4. **Procesamiento:** calcular indicadores básicos (por ejemplo, máximos/mínimos diarios, tasas de variación de temperatura) y preparar los datos para la visualización.
5. **Visualización:** servir una página web donde el usuario pueda consultar los valores actuales y parte del histórico.
6. **Expansión futura:** permitir la incorporación de nuevos sensores o nodos mediante la extensión del formato de trama y de las tablas de datos.

#### 4.1.3. Públicos objetivo y criterios de aceptación

Se identificaron tres grupos de interés:

- **Productor de rosas** (usuario principal): requiere una visualización clara de las variables y que el sistema funcione con mínima interacción. *Criterio de aceptación:* puede comprobar desde su PC/celular si la temperatura se aproxima a valores de riesgo sin desplazarse al invernadero.
- **Equipo de desarrollo:** necesita una plataforma flexible para experimentar con sistemas embebidos, Linux y LoRa. *Criterio de aceptación:* el sistema permite modificar el firmware y el software del gateway sin rehacer por completo el hardware.
- **Docente/evaluador:** busca evidencias de un diseño metodológico completo según la rúbrica. *Criterio de aceptación:* el informe documenta la especificación, el modelo, la arquitectura, el prototipado y la documentación de forma coherente.

#### 4.1.4. Evaluación de alternativas

Antes de seleccionar la arquitectura definitiva se evaluaron distintas opciones:

- **Sensado local sin comunicaciones:** uso de registradores de datos o termómetros simples. Se descartó porque no permite monitoreo a distancia ni anticipación a heladas.
- **Nodos WiFi/ESP32 con envío a la nube:** atractivo por la facilidad de uso, pero dependiente de conectividad a internet y de enrutadores WiFi en el invernadero, lo cual no está garantizado en el contexto real.
- **Sistema LoRa con gateway dedicado (opción elegida):** permite largo alcance, no depende de internet y es escalable a más nodos. Se eligió la Milk-V Duo 256M como gateway porque permite correr Linux, procesar datos y alojar un servidor web en la misma plataforma.

La alternativa LoRa + Milk-V Duo 256M resultó la más adecuada frente a los requisitos y restricciones identificados.

## 4.2. Modelo del sistema

### 4.2.1. Bloques constructores básicos

A partir de la especificación se definió un modelo funcional basado en bloques constructores básicos:

- **Nodo de sensado:** bloque que agrupa sensores (DHT11, BH1750, HW-080), microcontrolador Arduino UNO y módulo LoRa transmisor.
- **Enlace LoRa:** bloque de comunicación inalámbrica punto a punto a 433 MHz entre el nodo de sensado y el gateway.
- **Gateway:** bloque centrado en la Milk-V Duo 256M, con un módulo LoRa receptor, almacenamiento local y servidor web.
- **Usuario:** bloque que representa al productor consultando la información desde un PC o celular en la red local.

Este modelo se plasmó en el diagrama de bloques general (Fig. 2) y en el esquema conceptual del enlace LoRa punto a punto (Fig. 3).

### 4.2.2. Modelado funcional

A nivel funcional, el comportamiento del nodo de sensado se modeló como un ciclo periódico:

1. Inicialización de sensores (DHT11, BH1750, HW-080) y del módulo LoRa.
2. Espera de un intervalo de muestreo definido.
3. Lectura de las variables agroclimáticas y conversión a unidades físicas (por ejemplo, conversión de la lectura analógica del HW-080 a porcentaje de humedad de suelo mediante una función lineal calibrada).
4. Construcción de una trama de datos (cadena de texto con valores separados por delimitadores).
5. Transmisión de la trama mediante LoRa.

En el gateway, el modelo funcional incluye:

1. Inicialización del módulo LoRa receptor y del entorno Linux (puertos serie, rutas de archivos).
2. Bucle de escucha: esperar una trama LoRa, recibirla y validar su integridad básica.
3. Parsear la trama para extraer cada variable.
4. Registrar las mediciones en un archivo o base de datos, con marca de tiempo.
5. Actualizar las estructuras de datos utilizadas por el servidor web para mostrar la información al usuario.

#### 4.2.3. Validación y crítica del modelo

El modelo se validó de manera incremental:

- Primero se probaron los sensores conectados al Arduino, observando las lecturas en el monitor serie y comparándolas con mediciones de referencia (termómetro y observación cualitativa de luz y humedad de suelo).
- Luego se verificó el enlace LoRa enviando tramas con datos de prueba (por ejemplo, valores constantes conocidos) para confirmar la correcta recepción en la Milk-V Duo 256M.
- Finalmente se integró la lectura real de sensores con la transmisión LoRa y el registro en el gateway.

Durante este proceso se identificaron limitaciones del modelo, como la aproximación lineal en la conversión de la lectura del HW-080 a porcentaje de humedad de suelo, la falta de mecanismos avanzados de detección de paquetes perdidos y la ausencia de un identificador de nodo en la trama (por ahora se asume un único nodo). Estas observaciones se consignan como oportunidades de mejora para versiones posteriores.

### 4.3. Arquitectura del sistema

#### 4.3.1. División hardware/software y arquitectura global

Con el modelo funcional establecido, se definió la arquitectura global separando claramente hardware y software:

- **Hardware del nodo:** Arduino UNO, sensores DHT11/BH1750/HW-080, módulo LoRa a 433 MHz, fuente de alimentación.
- **Hardware del gateway:** Milk-V Duo 256M, módulo LoRa receptor, microSD con imagen Linux personalizada, conexión a red local (Ethernet o WiFi).
- **Software del nodo:** firmware en Arduino (C/C++) que realiza lectura de sensores, formateo de datos y transmisión LoRa.
- **Software del gateway:** imagen Linux construida específicamente para el SoC SG2002, script de recepción y almacenamiento de datos, y servidor web de visualización.

Esta división permite reutilizar hardware y software de manera modular: el firmware del nodo puede evolucionar independientemente de la lógica de visualización en el gateway.

#### 4.3.2. Arquitectura de comunicaciones

La comunicación entre nodo y gateway se organiza como un enlace **LoRa punto a punto** a 433 MHz:

- El nodo actúa como emisor periódico, utilizando un módulo basado en el transceptor LO-RA\_1278.
- El gateway integra un módulo LoRa configurado con los mismos parámetros físicos (frecuencia, *spreading factor*, ancho de banda y tasa de codificación).
- La trama de datos consiste en una cadena de texto con las cuatro variables medidas y, potencialmente, una marca de tiempo. El diseño contempla la posibilidad de añadir un identificador de nodo en versiones futuras.

Este esquema se refleja en el diagrama conceptual de comunicaciones (Fig. 3). La ausencia de LoRaWAN simplifica la implementación, al costo de renunciar a características como *roaming* o gestión centralizada de múltiples gateways.

#### 4.3.3. Arquitectura de software

En el nodo Arduino, la arquitectura de software se organiza en capas:

- **Capa de hardware:** inicialización de pines, buses (I<sup>2</sup>C, ADC) y módulo LoRa.
- **Capa de sensores:** uso de librerías para el DHT11 y el BH1750, y funciones específicas para la lectura y normalización del HW-080.
- **Capa de aplicación:** ciclo principal que coordina la lectura de sensores, el formateo de la trama y la transmisión LoRa.

En la Milk-V Duo 256M, la arquitectura de software incluye:

- **Capa de sistema:** imagen Linux personalizada para el SoC SG2002, con drivers y servicios mínimos necesarios.
- **Capa de comunicaciones:** script que abre el puerto asociado al módulo LoRa, configura los parámetros y recibe las tramas.
- **Capa de almacenamiento:** módulo que guarda las mediciones en archivos o en una base de datos ligera.
- **Capa de presentación:** servidor web sencillo que expone los datos al usuario mediante una página HTML, pudiendo ampliarse hacia gráficos o paneles más complejos.

#### 4.3.4. Evaluación de arquitecturas alternativas

Se consideraron arquitecturas alternativas, como:

- Conectar directamente los sensores a la Milk-V Duo 256M, eliminando el Arduino. Esta opción se descartó para mantener el nodo de sensado físicamente separado y aprovechar la sencillez de programación de Arduino para pruebas rápidas.
- Implementar una red LoRaWAN completa con un servidor de red y plataformas en la nube. Se descartó para este prototipo por su mayor complejidad y dependencia de infraestructura adicional.

La arquitectura elegida logra un equilibrio entre complejidad, escalabilidad y alineación con los objetivos académicos del proyecto.

### 4.4. Nodo de sensado y firmware de transmisión LoRa

El nodo de sensado se implementó sobre una placa Arduino UNO, encargada de adquirir las variables agroclimáticas (temperatura y humedad ambiente, humedad del suelo e iluminancia) y de empaquetar las lecturas en una trama de texto que se transmite mediante un módulo LoRa SX1278 a 433 MHz.

Dado que el Arduino UNO opera a 5 V y el transceptor LoRa trabaja a 3,3 V, fue necesario incorporar un conversor de nivel lógico (*level shifter*) en todas las líneas de salida del Arduino hacia el LoRa (SCK, MOSI, CS y RESET), mientras que las salidas del LoRa hacia el Arduino (MISO y DIO0) se conectaron directamente, ya que 3,3 V son compatibles con las entradas digitales del UNO.

#### 4.4.1. Montaje de hardware del nodo de sensado

En la Fig. ?? se muestra el montaje completo del nodo de sensado. El Arduino UNO se conecta a los tres sensores y al módulo LoRa a través de un *level shifter* de cuatro canales. Las conexiones principales son:

- **DHT11** (temperatura y humedad del aire): pin de datos al pin digital 7 del Arduino.
- **Sensor de humedad de suelo HW-080**: salida analógica al pin A2 del Arduino.
- **BH1750** (iluminancia): interfaz I<sup>2</sup>C sobre los pines A4 (SDA) y A5 (SCL) del Arduino.
- **Módulo LoRa SX1278**: alimentado a 3,3 V, con las líneas SCK, MOSI, CS y RESET atravesando el *level shifter* (5 V→3,3 V) y las líneas MISO y DIO0 conectadas directamente al Arduino.

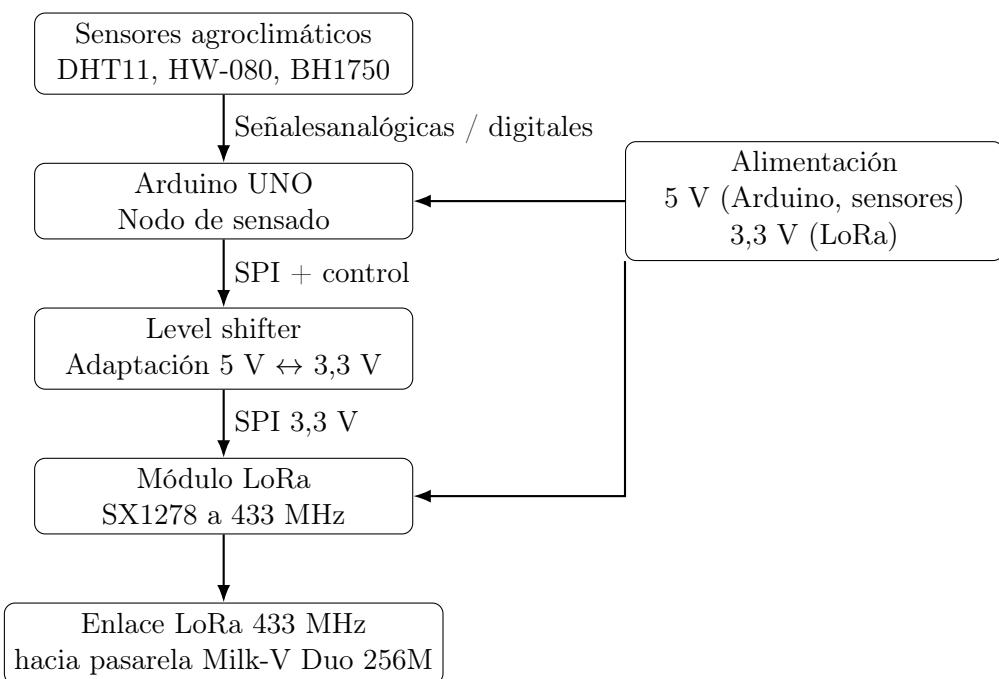


Figura 4: Diagrama de bloques vertical del nodo de sensado y transmisión LoRa. Los sensores agroclimáticos entregan sus lecturas al Arduino UNO, que las procesa y las envía al módulo LoRa a través de un *level shifter*. El módulo LoRa transmite los datos por el enlace inalámbrico a la pasarela Milk-V Duo 256M.

#### 4.4.2. Firmware de transmisión en Arduino UNO

El firmware del nodo de sensado se desarrolló en el entorno Arduino IDE, usando las librerías DHT, BH1750 y LoRa. El código realiza, en cada ciclo de la función `loop()`, las siguientes tareas:

1. Lectura de temperatura y humedad relativa del aire mediante el sensor DHT11 conectado al pin digital 7.
2. Lectura de la humedad de suelo en el pin analógico A2 y conversión del valor crudo a un porcentaje, utilizando una calibración lineal entre dos puntos de referencia (suelo seco y suelo húmedo).
3. Lectura de la iluminancia en lux mediante el sensor BH1750 conectado por I<sup>2</sup>C.

4. Construcción de una trama de texto con el formato:

```
temperatura,humedad_aire,humedad_suelo,iluminancia
```

donde la temperatura y la humedad del aire se envían con una cifra decimal, la humedad del suelo en porcentaje entero y la iluminancia como valor entero en lux.

5. Envío de la trama por LoRa usando las funciones `LoRa.beginPacket()`, `LoRa.print()` y `LoRa.endPacket()`, con una frecuencia portadora de 433 MHz y los parámetros físicos:

- Potencia de transmisión: 14 dBm.
- *Spreading factor*: 12.
- Ancho de banda: 62,5 kHz.
- Tasa de codificación: 4/8.

En el Listado 1 se presenta un fragmento del código correspondiente a la lectura de sensores y a la construcción del mensaje:

Listing 1: Fragmento del firmware de transmisión en Arduino UNO para el nodo de sensado.

```
float temperatura = dht.readTemperature();
float humedadAmb = dht.readHumidity();
int rawSoil      = analogRead(sensorSueloPin);
rawSoil = constrain(rawSoil, rawHumedo, rawSeco);
int humedadSoilPct = map(rawSoil, rawHumedo, rawSeco, 100, 0);
float iluminanciaLux = lightMeter.readLightLevel();

String mensaje = String(temperatura,1) + "," +
                String(humedadAmb,1) + "," +
                String(humedadSoilPct) + "," +
                String(iluminanciaLux,0);

LoRa.beginPacket();
LoRa.print(mensaje);
LoRa.endPacket();
```

El código completo del firmware se incluye en el Apéndice 1 para referencia.

#### 4.4.3. Pruebas de transmisión con monitor serie

Antes de integrar el nodo de sensado con la pasarela basada en la Milk-V Duo 256M, se verificó el funcionamiento del firmware utilizando únicamente el monitor serie del Arduino IDE. En la Fig. 5 se observa un ejemplo de salida, donde se muestran:

- Las lecturas de temperatura y humedad ambiente entregadas por el DHT11.
- El valor crudo del sensor de humedad de suelo y el porcentaje resultante tras la calibración.
- La iluminancia en lux reportada por el BH1750.
- La cadena completa que se envía por LoRa en cada ciclo de medición.

```

20:02:08.695 -> Temperatura: 23.2 °C
20:02:08.695 -> Humedad ambiente: 78.4 %
20:02:08.727 -> Humedad suelo: 900 raw → 0 %
20:02:08.760 -> Iluminancia: 3910 lx
20:02:08.793 -> Enviando por LoRa: 23.2,78.4,0,3910
20:02:18.710 -> -----
20:02:18.747 -> Temperatura: 23.1 °C
20:02:18.747 -> Humedad ambiente: 75.3 %
20:02:18.776 -> Humedad suelo: 900 raw → 0 %
20:02:18.809 -> Iluminancia: 52 lx
20:02:18.841 -> Enviando por LoRa: 23.1,75.3,0,52

```

Figura 5: Salida del monitor serie del Arduino UNO. Se muestran las lecturas de los sensores y la trama de texto enviada a través del módulo LoRa SX1278, lo que permite validar el firmware del nodo de sensado de manera independiente de la pasarela.

#### 4.5. Desarrollo de software del sistema embebido

En esta subsección se describe el proceso seguido para obtener la imagen Linux que corre en la Milk-V Duo 256M, a partir del repositorio [LicheeRV-Nano-Build](#). Este proceso incluye la preparación de un entorno de compilación en contenedor, la selección del `defconfig sg2002_duo_sd`, la ejecución de los scripts de construcción (`build_all`) y la grabación de la imagen resultante en una tarjeta microSD, así como los errores encontrados y las soluciones adoptadas.

#### 4.6. Prototipado y pruebas

##### 4.6.1. Implementación del hardware

El prototipo físico se montó en dos partes:

- **Nodo de sensado:** Arduino UNO montado en protoboard o PCB, con el DHT11 en una posición representativa del aire del invernadero, el BH1750 orientado hacia la zona de cultivo y la sonda HW-080 enterrada en el sustrato. El módulo LoRa se conectó al bus SPI del Arduino y se añadió una antena adecuada para 433 MHz.
- **Gateway:** Milk-V Duo 256M con su módulo LoRa receptor, tarjeta microSD con la imagen Linux generada y conexión a la red local mediante cable o WiFi. El prototipo se alimentó mediante una fuente estable de 5 V.

##### 4.6.2. Implementación del firmware y software

En el nodo de sensado se cargó el firmware Arduino que realiza la lectura de los sensores, el escalado básico de las señales (en especial la conversión de la lectura del HW-080 a porcentaje) y la transmisión periódica de la trama por LoRa.

En la Milk-V Duo 256M se desplegó el software de usuario que:

1. Inicializa el módulo LoRa receptor y abre el puerto correspondiente.
2. Recibe las tramas, las separa en sus componentes y las guarda en un archivo de registro.
3. Actualiza las estructuras de datos utilizadas por la página web de visualización.

#### 4.6.3. Plan de pruebas

Se diseñó un plan de pruebas sencillo pero alineado con los objetivos del prototipo:

- **Pruebas unitarias de sensores:** verificación de que cada sensor entrega valores razonables al modificar intencionalmente las condiciones (calentar el DHT11, iluminar el BH1750, humedecer/secar la zona de la sonda HW-080).
- **Pruebas del enlace LoRa:** envío de tramas de prueba a diferentes distancias dentro y fuera de recintos, verificando la recepción sin errores aparentes.
- **Pruebas de integración nodo–gateway:** comprobación de que las lecturas reales llegan correctamente al script de recepción y se almacenan en el formato esperado.
- **Pruebas de visualización:** revisión de que la página web muestra valores consistentes con las mediciones observadas localmente.

#### 4.6.4. Resultados de las pruebas

Las pruebas realizadas permitieron confirmar que:

- El nodo de sensado responde a cambios en el entorno: el aumento de temperatura cerca del DHT11, la variación de luz sobre el BH1750 y cambios en la humedad del suelo se reflejan en las lecturas enviadas.
- El enlace LoRa funciona de manera estable en distancias compatibles con el escenario de un invernadero y la ubicación del gateway, sin pérdidas significativas de paquetes en las condiciones de prueba.
- El gateway registra correctamente las tramas recibidas y la información puede visualizarse desde un dispositivo conectado a la misma red local.

Estos resultados demuestran la viabilidad del enfoque y proporcionan una base para extender el sistema a escenarios con más nodos y funcionalidades adicionales.

En conjunto, la metodología aplicada permitió pasar de una necesidad detectada en un contexto real (productor de rosas expuesto a heladas) a un prototipo funcional de sistema embebido documentado, reproducible y abierto a mejoras.

### 4.7. Desarrollo de software del sistema embebido

En este apartado se documenta el proceso de desarrollo de software de bajo nivel necesario para que la placa principal del proyecto, la **Milk-V Duo 256M** (SoC SG2002), pueda ejecutar Linux y las aplicaciones de monitoreo y control. El foco está en la creación y personalización de la imagen del sistema operativo utilizando el repositorio **LicheeRV-Nano-Build**, así como en el registro de los errores encontrados y las soluciones aplicadas.

### 4.8. Objetivos del desarrollo de software

El desarrollo de software en este nivel tuvo los siguientes objetivos:

- Obtener una imagen Linux funcional y reproducible para la Milk-V Duo 256M, basada en el SoC SG2002.
- Integrar en una única imagen el *bootloader*, el kernel Linux y el sistema de archivos raíz.
- Dejar documentado un flujo de compilación que pueda repetirse en el futuro (por otros integrantes del equipo o por el docente).

- Sentar las bases para habilitar interfaces de hardware (SPI, UART, GPIO) necesarias para los módulos LoRa y el control del invernadero.

## 4.9. Selección del repositorio y estrategia de compilación

Para soportar Linux en el SG2002 se eligió el repositorio oficial **LicheeRV-Nano-Build**, proporcionado por Sipeed. Este proyecto integra:

- *Bootloaders*: FSBL y U-Boot adaptados al SoC SG2002.
- Kernel Linux 5.10 con los parches específicos del fabricante.
- Sistema de archivos raíz basado en Buildroot.
- Scripts de construcción (`build.sh`, `build_all.sh`) y configuraciones (`defconfig`) para varias placas, incluyendo el objetivo `sg2002_duo_sd`, compatible con la Milk-V Duo 256M.

El código se obtuvo mediante:

```
git clone https://github.com/sipeed/LicheeRV-Nano-Build --depth=1
cd LicheeRV-Nano-Build
git clone https://github.com/sophgo/host-tools --depth=1
```

La estrategia final fue usar el entorno de compilación ya preparado en contenedor (Docker/Singularity) que provee el propio repositorio, en lugar de instalar todas las dependencias manualmente en la máquina anfitriona.

## 4.10. Preparación del entorno de compilación

### 4.10.1. Sistema anfitrión y contenedor

La compilación se realizó sobre una distribución Linux tipo Ubuntu, instalando inicialmente las herramientas básicas (`git`, `build-essential`, `python3`, etc.). A partir de ahí se siguieron las instrucciones del directorio `host/ubuntu` del repositorio para construir una imagen de entorno:

```
git clone https://github.com/sipeed/LicheeRV-Nano-Build --depth=1
cd LicheeRV-Nano-Build
git clone https://github.com/sophgo/host-tools --depth=1
```

Durante este proceso apareció el error:

```
permission denied while trying to connect to the Docker daemon socket
```

Esto se debió a que el usuario no pertenecía al grupo `docker`. La solución fue:

```
sudo usermod -aG docker <usuario>
```

Tras esto fue necesario cerrar la sesión y abrir una nueva terminal para que el cambio surtiera efecto. Una vez corregidos los permisos, la imagen se construyó correctamente.

Adicionalmente se utilizó un sistema de contenedores tipo Singularity/Apptainer con un archivo `.sqfs`. Este contenedor se ejecutó montando el proyecto en `/work`:

```
sudo singularity shell -B /ruta/a/LicheeRV-Nano-Build:/work \
    licheervnano-build-ubuntu.sqfs

# Dentro del contenedor
cd /work
```

#### 4.10.2. Carga del entorno de compilación

El proyecto incluye un script `build/cvisetup.sh` que configura rutas y variables de entorno. Inicialmente se intentó ejecutarlo desde un directorio incorrecto, obteniendo:

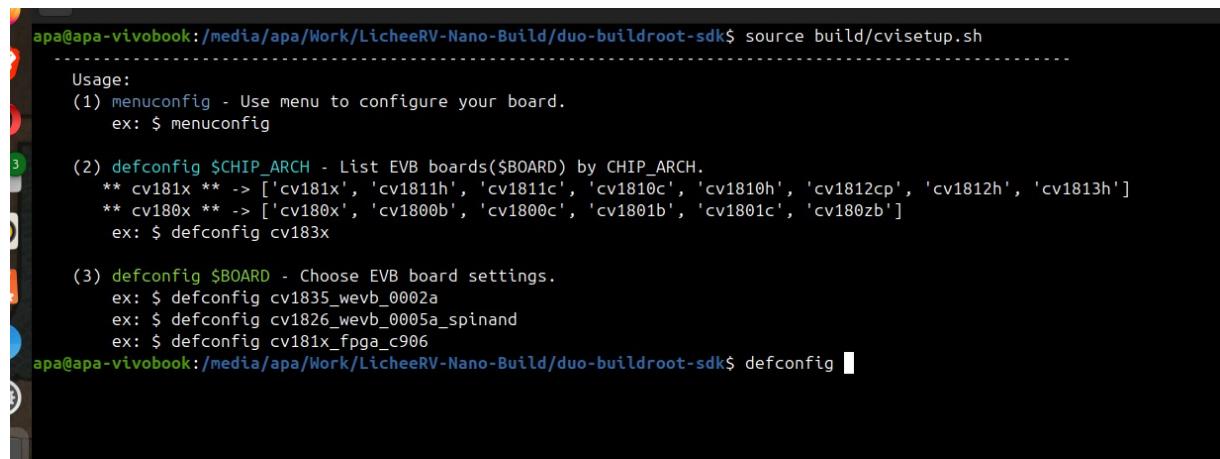
```
-bash: build/cvisetup.sh: No existe el archivo o el directorio
```

La solución fue ubicarse en la raíz del proyecto (`/work`) antes de ejecutar:

```
cd /work  
source build/cvisetup.sh
```

Solo después de este paso los comandos específicos del sistema de compilación (como `defconfig`) estuvieron disponibles.

La Fig. 6 muestra la salida del script `build/cvisetup.sh` una vez cargado el entorno, donde se resumen los comandos `menuconfig` y `defconfig` que se utilizaron en los pasos posteriores.



```
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ source build/cvisetup.sh  
-----  
Usage:  
(1) menuconfig - Use menu to configure your board.  
  ex: $ menuconfig  
  
(2) defconfig $CHIP_ARCH - List EVB boards($BOARD) by CHIP_ARCH.  
  ** cv181x ** -> ['cv181x', 'cv1811h', 'cv1811c', 'cv1810c', 'cv1810h', 'cv1812cp', 'cv1812h', 'cv1813h']  
  ** cv180x ** -> ['cv180x', 'cv1800b', 'cv1800c', 'cv1801b', 'cv1801c', 'cv1802b']  
  ex: $ defconfig cv183x  
  
(3) defconfig $BOARD - Choose EVB board settings.  
  ex: $ defconfig cv1835_wevb_0002a  
  ex: $ defconfig cv1826_wevb_0005a_spinand  
  ex: $ defconfig cv181x_fpga_c906  
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ defconfig
```

Figura 6: Ejecución de `source build/cvisetup.sh` en el SDK `duo-buildroot-sdk`, con el listado de comandos disponibles para configurar la placa (menú de configuración y `defconfig` por arquitectura y por tarjeta).

### 4.11. Configuración para la placa Milk-V Duo 256M

#### 4.11.1. Selección del `defconfig` adecuado

Una vez cargado el entorno con `cvisetup.sh`, el siguiente paso fue seleccionar la configuración específica de la placa Milk-V Duo 256M dentro del SDK `duo-buildroot-sdk`. Las configuraciones de tarjeta se encuentran en `build/boards/cv181x`, donde se incluye, entre otras, la opción:

- `cv1812cp_milkv_duo256m_sd_defconfig`

Tras revisar las opciones, se eligió esta configuración y se ejecutó:

```
defconfig cv1812cp_milkv_duo256m_sd
```

La salida del script (Fig. 7) muestra que se cargó correctamente el archivo de configuración, junto con las variables de entorno del proyecto (chip, versión del SDK, rutas de Linux 5.10 y U-Boot 2021.10, compilador cruzado, etc.).

```

CLEAN scripts/basic
CLEAN scripts/kconfig
CLEAN .config .config.old
spa@spa-vtobook:/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10$ cd ..
spa@spa-vtobook:/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ source build/cv1setup.sh
-----
Usage:
(1) menuconfig - Use menu to configure your board.
ex: $ menuconfig

(2) defconfig $BOARD - List EVB boards($BOARD) by CHIP_ARCH.
** cv181x ** -> ['cv181x', 'cv181h', 'cv181ic', 'cv1810h', 'cv1812cp', 'cv1812h', 'cv1813h']
** cv180x ** -> ['cv180x', 'cv1800b', 'cv1800c', 'cv1801b', 'cv1801c', 'cv1802b']
ex: $ defconfig cv183x

(3) defconfig $BOARD - Choose EVB board settings.
ex: $ defconfig cv1835_wevb_0002a
ex: $ defconfig cv1826_wevb_0005a_spinand
ex: $ defconfig cv181x_fpga_c906
-----
spa@spa-vtobook:/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ defconfig cv1812cp_milkv_duo256m_sd
Run defconfig function
Loaded configuration '/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/boards/cv181x/cv1812cp_milkv_duo256m_sd/cv1812cp_milkv_duo256m_sd_defconfig'
No change to configuration in '.config'
Loaded configuration '.config'
No change to minimal configuration in '/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/.defconfig'
/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build /media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk
/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk

===== Environment Variables =====

PROJECT: cv1812cp_milkv_duo256m_sd, DDR_CFG=ddr3_1866_x16
CHIP_ARCH: CV181X, DEBUG=0
SDK_VERSION: musl_riscv64, RPC=0
ATF_OPTIONS: ATF_KEY_SEL5=default, BL32=1
Linux source folder:linux_5.10, Uboot source folder: u-boot-2021.10
CROSS_COMPILE_PREFIX: riscv64-unknown-linux-musl-
ENABLE_BOOTLOGO: 0
Flash layout xml: /media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/boards/cv181x/cv1812cp_milkv_duo256m_sd/partition/partition_sd.xml
Sensor tuning bin: ov_os04a10
Output path: /media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/install/soc_cv1812cp_milkv_duo256m_sd

spa@spa-vtobook:/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ menuconfig_uboot
Run menuconfig_uboot() function
[TARGET] u-boot-menuconfig
make: *** No hay ninguna regla para construir el objetivo 'menuconfig'. Alto.

spa@spa-vtobook:/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10$ menuconfig
make[1]: se entra en el directorio '/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10'
make[2]: se entra en el directorio '/media/spa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/build/cv1812cp_milkv_duo256m_sd'
HOSTCC scripts/basic/fixedp
GEN Makefile
HOSTCC scripts/kconfig/conf.o

```

Figura 7: Ejecución de `defconfig cv1812cp_milkv_duo256m_sd` tras cargar el entorno. Se observa el mensaje *Run defconfig function* y el listado de variables de entorno específicas para la Milk-V Duo 256M.

En un intento inicial se trató de ejecutar `make menuconfig` desde la raíz del SDK, lo que produjo el error mostrado en la Fig. 8:

```
make: *** No hay ninguna regla para construir el objetivo 'menuconfig'. Alto.
```

Esto evidenció que la configuración del *bootloader* debía hacerse dentro del directorio de U-Boot (`u-boot-2021.10`) o mediante el objetivo `menuconfig_uboot` proporcionado por el propio SDK.

```

Flash layout xml: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/boards/cv181x/cv1812cp_milkv_duo256m_sd/partition/partition_sd.xml
Sensor tuning bin: ov_0s04a10
Output path: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/install/soc_cv1812cp_milkv_duo256m_sd

^[[ap@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ make menuconfig
make: *** No hay ninguna regla para construir el objetivo 'menuconfig'. Alto.
ap@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ cd u-boot-2021.10/
ap@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10$ make menuconfig
HOSTCC scripts/basic/fixedep
HOSTCC scripts/kconfig/conf.o
YACC scripts/kconfig/zconf.tab.c
LEX scripts/kconfig/zconf.lex.c
HOSTCC scripts/kconfig/zconf.tab.o
HOSTLD scripts/kconfig/conf
scripts/kconfig/Makefile:105: *** No configuration exists for this target on this architecture. Alto.
UPD scripts/kconfig/mconf-cfg
HOSTCC scripts/kconfig/mconfig.o
HOSTCC scripts/kconfig/lxdialog/checklist.o
HOSTCC scripts/kconfig/lxdialog/inputbox.o
HOSTCC scripts/kconfig/lxdialog/menubox.o
HOSTCC scripts/kconfig/lxdialog/textbox.o
HOSTCC scripts/kconfig/lxdialog/util.o
HOSTCC scripts/kconfig/lxdialog/yesno.o
HOSTLD scripts/kconfig/mconf
scripts/kconfig/mconf_Kconfig
arch/arm/mach-imx/nx7/Kconfig:22:warning: config symbol defined without type
arch/arm/mach-socfpga/Kconfig:24:warning: config symbol defined without type
arch/x86/Kconfig:949:warning: 'X86_OFFSET_SPL': number is invalid
common/spl/Kconfig:924:warning: config symbol defined without type

WARNING: unmet direct dependencies detected for OF_BOARD_SETUP
Depends on [n]: OF_LIBFDT [=n]
Selected by [y]:
- SANDBOX [=y] && <choice>

WARNING: unmet direct dependencies detected for CMD_FDT
Depends on [n]: OF_LIBFDT [=n]
Selected by [y]:
- CMD_EXTENSION [=y] && SUPPORT_EXTENSION_SCAN [=y]

WARNING: unmet direct dependencies detected for SYRESET_CMD_POWEROFF
Depends on [n]: SYRESET [=n] && CMD_POWEROFF [=y]
Selected by [y]:
- SANDBOX [=y] && <choice>
configuration written to .config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
ap@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10$
```

Figura 8: Error al intentar ejecutar `make menuconfig` desde la raíz del SDK y corrección al entrar en el directorio `u-boot-2021.10`. Este paso permitió editar la configuración de U-Boot sólo para la Milk-V Duo 256M.

## 4.12. Construcción de la imagen del sistema

### 4.12.1. Ejecución del script de construcción

Siguiendo las recomendaciones derivadas de la documentación y el análisis del script `build.sh`, se decidió lanzar la construcción completa desde el contenedor, montando el directorio del proyecto en `/work`. El comando final utilizado fue equivalente a:

```
docker run -it --rm -v $(pwd):/work licheeerv-nano-ubuntu ./build.sh
```

Dentro del entorno también se utilizó el script `build_all.sh` desde la raíz del proyecto:

```
cd /work
./build_all.sh
```

Este script se encarga de:

- Compilar el FSBL y OpenSBI para SG2002.
- Compilar U-Boot con la configuración `sg2002_duo_sd`.
- Compilar el kernel Linux 5.10 y sus módulos.
- Construir el sistema de archivos raíz (Buildroot).
- Empaquetar todo en una imagen `.img` arrancable desde microSD.

Durante los primeros intentos se presentaron errores de compilación, además de dudas sobre el uso de comandos como `make clean` en directorios donde no existía un `Makefile` válido, lo que generó mensajes del tipo:

```
make: *** No hay ninguna regla para construir el objetivo 'clean'. Alto.
```

Estos errores sirvieron para confirmar que el flujo correcto era el proporcionado por los scripts del propio repositorio (`build.sh`, `build_all.sh`) y no comandos genéricos de `make`.

Tras resolver los problemas de entorno y repetir la ejecución del script, la compilación terminó correctamente.

En la Fig. 9 se muestra la carpeta de compilación del FSBL, donde se genera el archivo `fip.bin`. Las Fig. 10 y 11 presentan, respectivamente, el binario del kernel `Image` y el *device tree* `cv1812cp_milkv_duo256m_sd.dtb` producidos durante la construcción.

```
4 de dic 09:44
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ source build/envsetup_soc.sh
bash: build/envsetup_soc.sh: No existe el archivo o el directorio
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ source build/cvlsetup.sh
-----
Usage:
  (1) menuconfig - Use menu to configure your board.
    ex: $ menuconfig

(2) defconfig $CHIP_ARCH - List EVB boards($BOARD) by CHIP_ARCH.
  ** cv181x ** -> ['cv181x', 'cv181h', 'cv181ic', 'cv1810h', 'cv1812cp', 'cv1812h', 'cv1813h']
  ** cv180x ** -> ['cv180x', 'cv1800h', 'cv1800c', 'cv1801b', 'cv1801c', 'cv1802b']
  ex: $ defconfig cv183x

(3) defconfig $BOARD - Choose EVB board settings.
  ex: $ defconfig cv1835_wevb_0002a
  ex: $ defconfig cv1826_wevb_0005a_spinand
  ex: $ defconfig cv181x_fpga_c986
-----
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ defconfig cv1812cp_milkv_duo256m_sd
Run defconfig function
Loaded configuration in '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/boards/cv181x/cv1812cp_milkv_duo256m_sd/cv1812cp_milkv_duo256m_sd_defconfig'
No change to configuration in '.config'
Loaded configuration '.config'
No change to minimal configuration in '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/.defconfig'
/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk
/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk

===== Environment Variables =====
PROJECT: cv1812cp_milkv_duo256m_sd, DDR_CFG=ddr3_1866_x16
CHIP_ARCH: CV181X, DEBUG=0
SDK VERSION: musl_riscv64, RPC=0
ATF options: ATF_KEY_SEL=defdefault, BL32=1
Linux source folder:linux 5.10, Uboot source folder: u-boot-2021.10
CROSS_COMPILE_PREFIX: riscv64-unknown-linux-musl-
ENABLE_BOOTLOGO: 0
Flash layout xml: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/boards/cv181x/cv1812cp_milkv_duo256m_sd/partition/partition_sd.xml
Sensor tuning bin: ov_0s04a10
Output path: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/install/soc_cv1812cp_milkv_duo256m_sd

apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ menuconfig_uboot
Run menuconfig_uboot() function
[TARGET] u-boot-menuconfig
make -j20 -C /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10 menuconfig
make[1]: se entra en el directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10'
make[2]: se entra en el directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/build/cv1812cp_milkv_duo256m_sd'
  GEN  Makefile
  /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/scripts/kconfig/Makefile:105: *** No configuration exists for this target on this architecture. Alto.
scripts/kconfig/conf Kconfig
  arch/arm/nach-inx/mx7/Kconfig:22:warning: config symbol defined without type
  arch/arm/nach-sorfnxa/Kconfig:24:warning: config symbol defined without type
```

Figura 9: Directorio de compilación del FSBL para la configuración `cv1812cp_milkv_duo256m_sd`, donde se observa el archivo `fip.bin` que contiene las distintas etapas de arranque (BL2, BL31) empaquetadas para el SoC CV1812.

```

4 de dic 09:45
apa@apa-vivobook: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/.defconfig
No change to minimal configuration in '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/.defconfig'
/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk

===== Environment Variables =====
PROJECT: cv1812cp_milkv_duo256m_sd, DDR_CFG=ddr3_1866_x16
CHIP_ARCH: CV181X, DEBUG=0
SDK VERSION: musl_riscv64, RPC=#
ATF options: ATF_KEY_SEL=default, BL32=1
Linux source folder: linux_5.10, Uboot source folder: u-boot-2021.10
CROSS_COMPILE_PREFIX: riscv64-unknown-linux-musl
ENABLE_BOOTLOGO: 0
Flash layout xml: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/build/boards/cv181x/cv1812cp_milkv_duo256m_sd/partition/partition_sd.xml
Sensor tuning bin: ov_0s04a10
Output path: /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/install/soc_cv1812cp_milkv_duo256m_sd

apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sd$ menuconfig_uboot
Run menuconfig_uboot() function
[TARGET] u-boot-menuconfig
make -j20 -C /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10 menuconfig
make[1]: se entra en el directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10'
make[2]: se entra en el directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/build/cv1812cp_milkv_duo256m_sd'
  GEN  Makefile
/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/scripts/kconfig/Makefile:105: *** No configuration exists for this target on this architecture. Alto.
scripts/kconfig/mconf Kconfig
arch/arm/mach-lnx/mx7/Kconfig:22:warning: config symbol defined without type
arch/arm/mach-socfpga/Kconfig:24:warning: config symbol defined without type
arch/x86/Kconfig:949:warning: X86_OFFSET_SPL: number is invalid
common/spl/Kconfig:924:warning: config symbol defined without type

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

make[2]: se sale del directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/build/cv1812cp_milkv_duo256m_sd'
make[1]: se sale del directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10'
make -j20 -C /media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10 savedefconfig
make[1]: se entra en el directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10'
make[2]: se entra en el directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/build/cv1812cp_milkv_duo256m_sd'
  GEN  Makefile
/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/scripts/kconfig/Makefile:105: *** No configuration exists for this target on this architecture. Alto.
scripts/kconfig/conf --savedefconfig=deconfig Kconfig
arch/arm/mach-lnx/mx7/Kconfig:22:warning: config symbol defined without type
arch/arm/mach-socfpga/Kconfig:24:warning: config symbol defined without type
arch/x86/Kconfig:949:warning: X86_OFFSET_SPL: number is invalid
common/spl/Kconfig:924:warning: config symbol defined without type
make[2]: se sale del directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10/build/cv1812cp_milkv_duo256m_sd'
make[1]: se sale del directorio '/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/u-boot-2021.10'
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sd$
```

Figura 10: Ubicación del kernel Linux 5.10 compilado (Image) para la Milk-V Duo 256M en el árbol de `linux_5.10`.

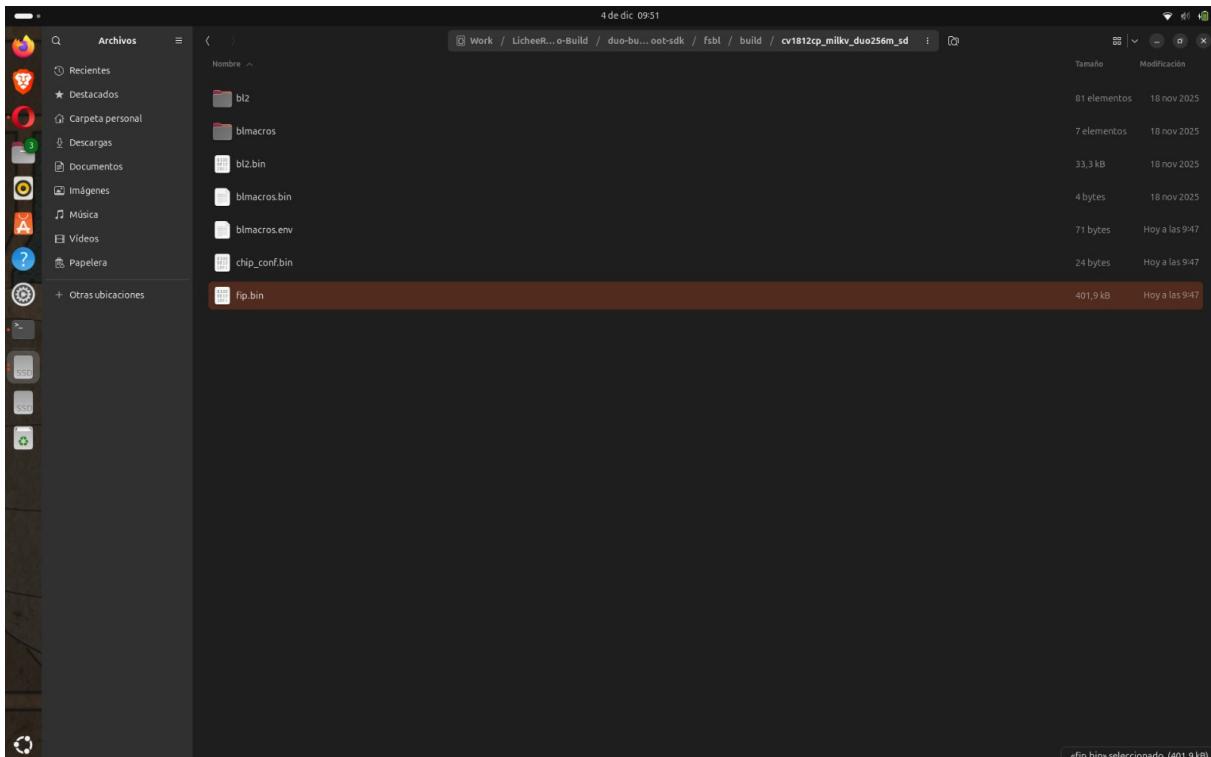


Figura 11: Listado de archivos `.dtb` generados para distintas tarjetas CVITEK. Se resalta el `cv1812cp_milkv_duo256m_sd.dtb` utilizado por RoseWatch.

#### 4.12.2. Localización de la imagen generada

Una vez completado el proceso, la imagen de salida quedó almacenada en:

```
/work/install/soc_sg2002_duo_sd/images/2025-09-25-09-41-505c0a.img
```

Este archivo corresponde a la imagen completa (bootloaders + kernel + rootfs) para la Milk-V Duo 256M arrancando desde microSD.

#### 4.12.3. Grabación en microSD y validación básica

La imagen generada se grabó en una tarjeta microSD utilizando herramientas estándar. En Linux se empleó, por ejemplo:

```
sudo dd if=install/soc_sg2002_duo_sd/images/2025-09-25-09-41-505c0a.img \
    of=/dev/sdX bs=4M status=progress conv=fsync
```

Tras insertar la tarjeta en la Milk-V Duo 256M y alimentar la placa, se validó:

- La aparición de mensajes de arranque por el puerto serie (U-Boot y kernel).
- El reconocimiento del SoC SG2002 y de la memoria principal.
- La creación de dispositivos en `/dev` necesarios para UART, SPI y otros periféricos usados en el proyecto.

Esta validación garantiza que la placa queda en un estado apto para instalar posteriormente las aplicaciones de usuario (recepción LoRa, procesamiento y servidor web).

#### 4.12.4. Personalización de la imagen

Una vez obtenida la imagen base, se exploraron mecanismos de personalización sin recompilar todo el sistema:

- **Modificación de la primera partición (FAT)** para añadir archivos de configuración (por ejemplo, `wifi.sta`) o cambiar elementos de arranque (logo).
- **Montaje de la partición rootfs (ext4)** para insertar scripts de usuario, utilidades de diagnóstico y herramientas para pruebas de SPI, LoRa y control de actuadores.

Estas modificaciones se realizaron montando las particiones de la imagen en un directorio temporal, copiando los archivos necesarios y desmontando de nuevo antes de grabarla en la microSD.

### 4.13. Configuración y Compilación del SDK para Milk-V Duo 256M

Para configurar y compilar correctamente el SDK para la Milk-V Duo 256M, se siguen varios pasos esenciales, desde la clonación del repositorio hasta la configuración del *kernel* y *uboot*. A continuación se detallan los pasos clave que aseguran que todo funcione correctamente, incluyendo la corrección de errores durante el proceso de compilación.

#### 1. Clonación del SDK

El primer paso es clonar el SDK de Buildroot para la plataforma Milk-V Duo 256M desde el repositorio oficial:

```
git clone https://github.com/milkv-duo/duo-buildroot-sdk.git
```

Este SDK contiene todos los archivos y configuraciones necesarias para compilar el sistema y las herramientas necesarias para la Milk-V Duo 256M, como el *kernel*, *uboot*, y otros componentes clave.

## 2. Activación de las variables de entorno

Una vez clonado el SDK, es necesario activar las variables de entorno para que el sistema de compilación funcione correctamente:

```
source build/cvisetup.sh
```

Este script configura las variables del entorno que permiten que las herramientas y configuraciones de Buildroot funcionen correctamente, facilitando la compilación del sistema para la Milk-V Duo 256M.

## 3. Selección de la placa objetivo

El siguiente paso es seleccionar la configuración de la placa. Para ello, utilizamos el comando:

```
defconfig cv1812cp_milky_duo256m_sd
```

Esto asegura que todas las configuraciones específicas para la Milk-V Duo 256M sean aplicadas, como las configuraciones de CPU, memoria, periféricos, etc.

## 4. Configuración de uboot

A continuación, se configura *uboot*, el cargador de arranque que gestiona el inicio del sistema operativo en la Milk-V Duo 256M. Las configuraciones clave incluyen (12):

- *Activación de opciones de boot:* Activamos el soporte para arrancar desde SD/EMMC:  
[\*] Support for booting from SD/EMMC
- *Configuración del entorno:* Especificamos que el entorno se almacenará en un sistema de archivos FAT en la partición SD:  
[\*] Environment is in a FAT filesystem

Luego, configuraremos los parámetros específicos del entorno, como la ubicación del entorno en la SD y el nombre del archivo:

```
(0:1) Device and partition for where to store the environment in FAT  
(uboot.env) Name of the FAT file to use for the environment  
(0) mmc device number  
(1) mmc partition number
```

- *Soporte de sistemas de archivos:* Activamos el soporte de escritura para ext4 en el sistema:

```
[*] Enable ext4 filesystem write support
```

Esto permite usar el comando **saveenv** dentro de *uboot* para guardar las configuraciones del entorno en la partición SD.

```

Environment
navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularize
Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

[ ] Enable overwriting environment
[ ] Environment is not stored
[ ] Environment in EEPROM
[*] Environment is in a FAT filesystem
[ ] Environment is in a EXT4 filesystem
[ ] Environment in flash memory
[ ] Environment in an MMC device
[ ] Environment in a NAND device
[ ] Environment in a non-volatile RAM
[ ] Environment is in OneNAND
[ ] Environment is in remote memory space
[ ] Environment is in SPI flash
[ ] Enable redundant environment support
  (mmc) Name of the block device for the environment
  (0:1) Device and partition for where to store the environment in FAT
  (uboot.env) Name of the FAT file to use for the environment
  [ ] Relocate gd->env_addr
  (0) mmc device number
  (1) mmc partition number
  [ ] Create default environment from file
  [ ] Add run-time information to the environment
  [ ] Amend environment by FDT properties
  [ ] Always append the environment with new data
  [ ] Permit write access only to listed variables
  [ ] Block forced environment operations
  [ ] Add a 'ver' environment variable with the U-Boot version

```

Figura 12: Imagen de las variables de entorno.

#### 5. Configuración del Kernel

El siguiente paso es configurar el *kernel*. Se utiliza el siguiente comando para acceder al menú de configuración:

`menuconfig_kernel`

Dentro de este menú, configuramos opciones similares a las de *uboot*, como el soporte para arrancar desde SD/EMMC, y activamos las opciones de *boot media* y entorno en sistema de archivos FAT.

#### 6. Compilación del sistema

Una vez configurado todo, se compila el sistema usando el comando:

`build_all`

Este comando compila todos los componentes del sistema, pero en algunos casos, puede generar un error en algún paso del proceso. Si esto ocurre, se puede forzar la compilación del *kernel* de forma independiente usando el siguiente comando:

`build_kernel`

Este paso asegura que los archivos clave estén disponibles en las siguientes rutas:

```

./linux_5.10/build/sg2002_duo_sd/arch/riscv/boot/Image
./linux_5.10/build/sg2002_duo_sd/arch/riscv/boot/dts/cvitek/sg2002_duo_sd.dtb
./fsbl/build/cv1812cp_milkv_duo256m_sd/fip.bin

```

Estos archivos deben ser copiados a la partición FAT de la tarjeta SD para que el proceso de arranque funcione correctamente.

#### 7. Configuración de uboot para arrancar el sistema

Para sobrescribir el proceso de arranque normal de la placa y utilizar los archivos generados por la compilación, es necesario modificar las variables de entorno de *uboot*. Esto se hace accediendo al menú de *uboot* durante el arranque de la placa, presionando cualquier tecla para acceder a las opciones de configuración. Las variables de entorno necesarias son:

```

setenv cain_boot 'setenv bootargs ${reserved_mem} ${root} ${mtdparts} console=ttyS0,115200n8
setenv dtb_addr 0x8b300000
setenv sddev 0
setenv bootcmd run_cainboot

```

Estas configuraciones permiten que *uboot* cargue el *kernel*, el *device tree* (dtb) y el *file system* desde la tarjeta SD, asegurando que el sistema se inicie correctamente con los archivos que hemos configurado.

Por ultimo se uso la herramienta de fiptool, para crear el archivo .fip apoyandonos en el repositorio (<https://github.com/sophgo/fiptool/blob/master/README.md>). Además de estas modificaciones generales, fue necesario habilitar el bus SPI utilizado por el módulo LoRa y comprobar su funcionamiento desde espacio de usuario.

### 1. Configuración del archivo DTS (Device Tree Source)

El archivo DTS (Device Tree Source) es una estructura de datos que describe el hardware de la placa. Es necesario configurarlo adecuadamente para que el sistema reconozca los dispositivos y periféricos durante el arranque.

#### *Ubicación del archivo DTS:*

El archivo que necesitas modificar se encuentra en la siguiente ruta dentro del SDK:

```
linux_5.10/build/cv1812cp_milkv_duo256m_sd/arch/riscv/boot/dts/cvitek
```

#### *Modificación del archivo DTS:*

Abre el archivo DTS que corresponde a tu placa (`cv1812cp_milkv_duo256m_sd.dts`) y realiza las modificaciones necesarias. Esto podría implicar cambiar parámetros relacionados con memoria, dispositivos SPI, frecuencia de reloj, y otros dispositivos periféricos específicos de la placa. Ejemplo de parte relevante de un archivo DTS:

```

&spi1 {
    clock-frequency = <40000000>;
    pinctrl-0 = <&spi1_pins_a &spi1_pins_b>;
    pinctrl-1 = <&spi1_pins_c>;
    pinctrl-names = "default", "sleep";
    spi_slave_mode = <0>;
    spi_cs_number = <1>;
    spi_cs_bitmap = <1>;
    status = "okay";
};

spidev@0 {
    compatible = "rohm,dh2228fv";
    status = "okay";
    spi-max-frequency = <0x2625A00>;
    reg = <0>;
};

```

En este archivo, se configura el SPI y se establece la frecuencia del reloj y otros parámetros para el dispositivo LoRa u otros periféricos.

### 2. Configuración de uboot

Una vez configurado el DTS, es necesario compilar y configurar *uboot*, el cargador de arranque que inicializa el hardware y carga el sistema operativo. El proceso para compilar *uboot* y configurar las variables de entorno es esencial para que el sistema se inicie correctamente.

#### *Acceso al menú de configuración de uboot:*

Dentro del SDK, se deben realizar ciertas configuraciones adicionales para preparar *uboot*.

Para ello, accede al menú de configuración usando el siguiente comando:

```
make menuconfig
```

En este menú, seleccionas las opciones para habilitar el soporte para SPI y otras configuraciones necesarias, como el soporte de memoria y sistemas de archivos. Asegúrate de activar las opciones necesarias, como:

- *SPI support*: Esto es necesario para que *uboot* pueda gestionar la comunicación SPI.
- *SPI Master Controller*: Configuración para que *uboot* actúe como maestro SPI.
- *Support for booting from SD/EMMC*: Esto asegura que *uboot* puede arrancar desde una tarjeta SD o EMMC.

Esta configuración es crucial para que el sistema de arranque funcione correctamente en la Milk-V Duo 256M.

Por último se ejecuta:

```
make build_uboot
```

En primer lugar se verificó, tras el arranque, la presencia de los dispositivos SPI en /dev, así como la información de los controladores registrados, tal como se ilustra en la Fig. 13. A continuación se detuvo el servicio de WiFi, ya que comparte el mismo bus SPI, y se configuraron los pines multifunción del puerto P mediante escrituras directas a registros *GPIO* usando *devmem*:

```
/etc/init.d/S30wifi stop    # Detener WiFi si usa el mismo SPI  
  
. /devmem 0x030010D0 b 0x1  # GPIO P18 -> CS del LoRa  
. /devmem 0x030010DC b 0x1  # GPIO P21 -> MISO  
. /devmem 0x030010E0 b 0x1  # GPIO P22 -> MOSI  
. /devmem 0x030010E4 b 0x1  # GPIO P23 -> SCK
```

Cada llamada a *devmem* fuerza el modo de cada pin a la función alternativa asociada al SPI (chip select, MISO, MOSI y reloj), de modo que el módulo LoRa queda conectado lógicamente al controlador expuesto como /dev/spidevX.Y.

Para comprobar que el bus SPI estaba operativo se instaló la utilidad *spidev\_test* y se realizó una transferencia de prueba:

```
spidev_test -D /dev/spidev1.0 -v
```

Seguido se creó, un archivo spi.c el cual nos permitía evaluar después de compilar el estado de los spi, usando miso y mosi, donde el código proporcionado es una prueba básica para comunicación SPI (Serial Peripheral Interface) en una plataforma basada en Linux, como la Milk-V Duo 256M. En esta prueba, se utiliza la interfaz SPI para enviar y recibir datos a través de un dispositivo SPI (/dev/spidev0.0). El objetivo principal de esta prueba es asegurarse de que la comunicación SPI esté funcionando correctamente y que los parámetros del driver SPI se configuren adecuadamente.

1. Apertura del dispositivo SPI para establecer la comunicación con el dispositivo conectado a la interfaz SPI.
2. Configuración del modo SPI (*SPI\_MODE\_0*) usando *ioctl*.

3. Definición de la transferencia SPI en la estructura `spi_ioc_transfer`, donde se especifican los *buffers* de transmisión y recepción, la velocidad SPI y el número de bits por palabra.
4. Ejecutar la transferencia SPI con `ioctl`.
5. Imprimir los datos recibidos para verificar que la transferencia se realizó correctamente.
6. Cerrar el archivo SPI una vez completada la operación.

La compilación se realizó de la siguiente manera:

```
~/duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-  
-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0  
spi.c -o /tmp/spi
```

La salida del comando (Fig. 14) mostró una secuencia de bytes transmitidos y recibidos sin errores, confirmando que la configuración del kernel, del *device tree* y de los pines era correcta y que el sistema estaba listo para integrar el driver de LoRa en el siguiente nivel de software.

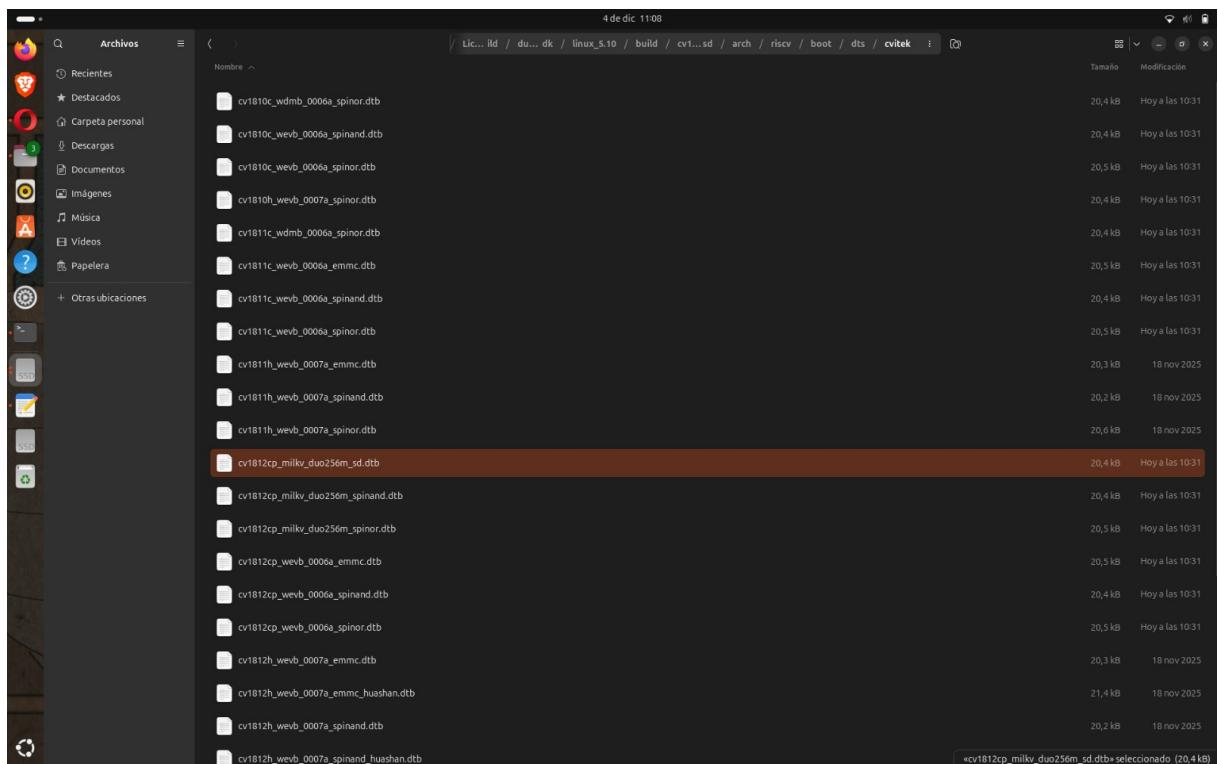


Figura 13: Consola de la Milk-V Duo 256M tras el arranque desde la imagen generada. Se observan los dispositivos en `/dev`, incluidos los nodos `spidevX.Y` correspondientes al bus SPI.

Figura 14: Ejecución de `spidev_test` sobre `/dev/spidev1.0`. La herramienta intercambia bytes a 500 kHz, verificando el correcto funcionamiento del bus SPI antes de conectar el módulo LoRa.

```
# ls -la /dev/spi*
crw----- 1 root      root      153,    0 Jan  1 00:00 /dev/spidev0.0
crw----- 1 root      root      153,    1 Jan  1 00:00 /dev/spidev1.0
#
```

Figura 15: Reconocimiento de spidev

#### 4.14. Toolchain y compilador

Durante la preparación del entorno de compilación fue necesario localizar y habilitar el tool-chain adecuado para generar ejecutables compatibles con la arquitectura de la tarjeta Milk-V Duo 256 M. Dado que la placa utiliza un procesador CV1812, basado en la arquitectura RISC-V de 64 bits, los binarios deben ser compilados mediante un compilador cruzado (cross-compiler), ya que el sistema anfitrión (el PC) ejecuta una arquitectura x86-64.

Por lo tanto primero se carga el SDK, se busca acceder a las herramientas provistas por Buildroot. El compilador cruzado se encuentra dentro de hostools y en la sección de gcc y fue posible listar todos los ejecutables del toolchain, entre los que se identificó el compilador principal.

```
apa@apa-vtobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ ls host-tools/
gcc .git
apa@apa-vtobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ ls host-tools/gcc/
gcc-linaro-6.3.1-2017.05-x86_64_aarch64-elf/           gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/   riscv64-linux-x86_64/
gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/     riscv64-elf-x86_64/
gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-gnueabihf/  riscv64-linux-musl-x86_64/
apa@apa-vtobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ ls host-tools/gcc/riscv64-linux-musl-x86_64/
bin lib lib64 libexec riscv64-unknown-linux-musl share sysroot
apa@apa-vtobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ ls
build          build.sh freertos host-tools isp_tuning middleware osdrv README-ja.md README-zh.md
buildroot-2021.05 device fsbl install linux_5.10 opensbi ramdisk README.md u-boot-2021.10
apa@apa-vtobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ ls host-tools/gcc/riscv64-linux-musl-x86_64/
bin/           lib/           libexec/      share/
include/       lib64/        riscv64-unknown-linux-musl/ sysroot/
apa@apa-vtobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk$ ls host-tools/gcc/riscv64-linux-musl-x86_64/bin/
riscv64-unknown-linux-musl-addr2line riscv64-unknown-linux-musl-gcc           riscv64-unknown-linux-musl-gdb           riscv64-unknown-linux-musl-objdump
riscv64-unknown-linux-musl-ar         riscv64-unknown-linux-musl-gcc-10.2.0  riscv64-unknown-linux-musl-gdb-add-index  riscv64-unknown-linux-musl-ranlib
riscv64-unknown-linux-musl-as         riscv64-unknown-linux-musl-gcc-ar      riscv64-unknown-linux-musl-gprof       riscv64-unknown-linux-musl-readelf
riscv64-unknown-linux-musl-c++        riscv64-unknown-linux-musl-gcc-nn      riscv64-unknown-linux-musl-ld          riscv64-unknown-linux-musl-size
riscv64-unknown-linux-musl-c++filt   riscv64-unknown-linux-musl-gcc-ranlib   riscv64-unknown-linux-musl-lld         riscv64-unknown-linux-musl-strings
riscv64-unknown-linux-musl-cpp       riscv64-unknown-linux-musl-gcov       riscv64-unknown-linux-musl-lto-dump    riscv64-unknown-linux-musl-strip
riscv64-unknown-linux-musl-elfedit   riscv64-unknown-linux-musl-gcov-dump   riscv64-unknown-linux-musl-ld          riscv64-unknown-linux-musl-ld
riscv64-unknown-linux-musl-g++       riscv64-unknown-linux-musl-gcov-tool   riscv64-unknown-linux-musl-lm          riscv64-unknown-linux-musl-objcopy
```

Figura 16: Ubicacion del compilador gcc

Para facilitar su uso, se añadió el directorio del toolchain al PATH, con ello fue posible ejecutar directamente confirmando que el compilador estaba correctamente instalado y era funcional.

```
sp@sp-OptiPlex-5090:~/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot$ cd host-tools/gcc/riscv64-linux-musl-x86_64/bin/
sp@sp-OptiPlex-5090:~/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot$ cd host-tools/gcc/riscv64-linux-musl-x86_64/bin$ pwd
/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot$cd host-tools/gcc/riscv64-linux-musl-x86_64/bin
sp@sp-OptiPlex-5090:~/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot$cd host-tools/gcc/riscv64-linux-musl-x86_64/bin$ 
sp@sp-OptiPlex-5090:~/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot$cd host-tools/gcc/riscv64-linux-musl-x86_64/bin$ ls
riscv64-unknown-linux-musl-addr2line  riscv64-unknown-linux-musl-gcc      riscv64-unknown-linux-musl-objdump
riscv64-unknown-linux-musl-ar        riscv64-unknown-linux-musl-gc-10.2.0  riscv64-unknown-linux-musl-ranlib
riscv64-unknown-linux-musl-ld        riscv64-unknown-linux-musl-gdb-add-index riscv64-unknown-linux-musl-readelf
riscv64-unknown-linux-musl-ldc       riscv64-unknown-linux-musl-gprof      riscv64-unknown-linux-musl-readelf
```

Figura 17: Compilador añadido al PATH

Para validar el funcionamiento del compilador se creó un archivo sencillo en C (`hello.c`). En el primer intento de compilación se presentó un error debido al uso incorrecto del parámetro `-c`, el cual instruye al compilador a generar únicamente un archivo objeto (`.o`) sin proceder al proceso de enlace. Esto provocaba que el ejecutable final no pudiera generarse. Una vez identificado el origen del problema, se corrigió eliminando la opción `-c`, permitiendo así que `gcc` realizara tanto la compilación como el enlace en un solo paso.

```
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin$ cd  
apa@apa-vivobook:$  
apa@apa-vivobook:$  
apa@apa-vivobook:$ export PATH=$PATH:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin  
apa@apa-vivobook:$ vi hello.c  
apa@apa-vivobook:$  
apa@apa-vivobook:$  
apa@apa-vivobook:$  
apa@apa-vivobook:$  
apa@apa-vivobook:$ riscv64-unknown-linux-musl-gcc -c hello.c -o hello  
hello.c:3:15: error: expected ';' ',' or ')' before '{' token  
    3 | void main(void[  
      | ^  
apa@apa-vivobook:$ vi hello.c  
apa@apa-vivobook:$ riscv64-unknown-linux-musl-gcc -c hello.c -o hello  
apa@apa-vivobook:$ ls
```

Figura 18: Linea erronea de compilación

```
apa@apa-vivobook:/media/apa/Work/LicheeRV-Nano-Build/duo-buildroot-sdk5 riscv64-unknown-elf-gcc hello.c -o /tmp/hello
```

Figura 19: Ejecución funcional de compilación

#### 4.14.1. Integración de librerías y resolución de dependencias

Durante la compilación de aplicaciones más complejas, como las pruebas de SPI y los módulos asociados al módulo LoRa, se presentaron errores derivados de dependencias no resueltas. En

particular, el compilador no lograba encontrar ciertos archivos de cabecera ni funciones específicas al intentar incluir librerías como:

```
include <linux/spi/spidev.h >
```

Este comportamiento se debe a que, en el entorno Buildroot, dichas librerías forman parte del sysroot propio del SDK y requieren parámetros adicionales de compilación para ser visibles al compilador cruzado. Para identificar estos parámetros se analizó el código de referencia spidevtest.c, incluido dentro del kernel y utilizado tradicionalmente para validar la interfaz SPI en Linux.

A partir de esta revisión se determinó que era necesario compilar los programas por lo que al momento de realizar la compilación se realizó de la siguiente manera:

```
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc \
-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 \
-O2 -g0 \
spi.c -o /tmp/spi
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 hello.c -o /tmp/hello
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 hello.c -o /tmp/hello
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ cp -r hello /media/apa/boot/
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ riscv64-unknown-elf-gcc hello.c -o /tmp/hello
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 spi.c -o /tmp/spi
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ riscv64-unknown-elf-gcc hello.c -o /tmp/hello
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 loras.c -o /tmp/loras
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 loras.c -o /tmp/loras
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 loras.c -o /tmp/loras
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 loras.c -o /tmp/loras
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 lorass.c -o /tmp/lorass
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 lorass.c -o /tmp/lorass
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 lorass.c -o /tmp/lorass
apa@apa-vivebook:/media/apa/Work/licheeRV-Nano-Build/duo-buildroot-sdk$ ./duo-buildroot-sdk/host-tools/gcc/riscv64-linux-musl-x86_64/bin/riscv64-unknown-linux-musl-gcc -D_LARGEFILE_SOURCE
-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 Pruebac.c -o /tmp/Pruebac
```

Figura 20: Compilación con dependencias

Este mismo procedimiento se aplicó para los diferentes programas desarrollados (SPI, LoRa, pruebas básicas), logrando finalmente generar ejecutables funcionales y compatibles con la arquitectura RISC-V de la Milk-V Duo 256M

#### 4.14.2. Transferencia y despliegue de ejecutables en la Milk-V Duo

Una vez generados los ejecutables en el entorno de desarrollo, fue necesario transferirlos hacia la Milk-V Duo 256M para su validación en la plataforma real. Debido a las restricciones del sistema de archivos —donde la partición raíz (/) se encuentra montada en modo de solo lectura—, la única ubicación accesible para escritura permanente en la tarjeta SD es la partición /boot, montada con permisos de lectura y escritura mediante el sistema de archivos FAT. Tras establecer la conexión serial a través de minicom, se accedió al sistema embebido y se navegó hacia el único directorio con permisos de escritura. Desde este punto, los archivos enviados desde el host pueden almacenarse y ejecutarse sin afectar las particiones protegidas del sistema. Para realizar la transferencia se empleó el protocolo ZMODEM, disponible por defecto en minicom y adecuado para entornos embebidos sin servicios de red. El procedimiento seguido fue el siguiente:

1. En la Milk-V, activar el receptor ZMODEM ejecutando:  
**rz**
2. En el host (PC), dentro de minicom:
  - a) presionar **Ctrl + A**
  - b) luego **S** para abrir el menú de envío
  - c) seleccionar el protocolo *zmodem*
  - d) navegar hacia la carpeta donde se almacenó el ejecutable compilado

- e) confirmar el archivo a enviar
3. Durante la transferencia, la Milk-V muestra el progreso en consola y almacena automáticamente el archivo recibido en el directorio actual (/boot).

```

| Directorio de llamadas telefónicas.. D Ejecutar script....G | Lim
|piar pantalla.....C
| Enviar ficheros....S Recibir ficheros...R | Configurar Minicom.O
| Parámetros de comunicación...P Añadir salto de línea..A | Suspend
|er minicom...J
| Capturar encendido/apagado.L Desconectar.....H | Salir y reini
|cializar.....X
| Enviar parada.....F Inicializar Modem..M | Salir sin reiniciali
|zar...Q
| Config. Terminal.. T Ejecutar Kermit....K | Tecla de modo con cur
|sor....I
| Ajuuste de linea activado/desactivado.....W Eco local encendido/a
|pagado.Y | Pantalla de ayuda.....Z
| Pegar fichero.....Y Conmutar sello de tiempo...N | Retroceso
|.....B
| Añadir RSeleccione función o presione Enter para salir.■

```

Figura 21: Navegación hacia ZMODEM

1. Una vez finalizada la transferencia, se comprobó la presencia del archivo:

```
ls -l /boot
```

2. La ejecución se realizó directamente desde la partición FAT:

```
/boot/nombre_ejecutable
```

#### 4.15. Errores encontrados y lecciones aprendidas

Durante la creación de la imagen se cometieron varios errores que quedaron registrados en los comandos ejecutados y en el intercambio con un asistente de IA. Los más relevantes y lo aprendido de ellos fueron:

- **Ejecución de comandos en directorios equivocados:** intentar correr `make clean` o `source build/cvisetup.sh` fuera de la raíz del proyecto provocó errores sobre archivos inexistentes o reglas ausentes. La lección fue identificar siempre el directorio correcto de trabajo (`/work` en el contenedor).
- **No cargar el entorno antes de usar defconfig:** el error `bash: defconfig: command not found` se resolvió recordando que primero debe ejecutarse `source build/cvisetup.sh`.
- **Confusión entre Docker y Singularity:** inicialmente se mezclaron ambos enfoques (imágenes Docker y archivos `.sqfs`). Finalmente se optó por seguir exactamente la ruta recomendada en el `README`, evitando pasos adicionales que introducían complejidad.
- **Reintentos de compilación:** algunos fallos de compilación se solucionaron repitiendo el script de construcción ya con el entorno bien configurado, confirmando que parte de los errores se debían a estados intermedios incoherentes.

Documentar estos errores y sus soluciones es importante porque demuestra el proceso de depuración seguido y facilita que otra persona pueda reproducir el flujo sin repetir los mismos fallos.

#### 4.15.1. Relación con los objetivos del proyecto

El resultado de este capítulo de desarrollo de software es una imagen Linux funcional y personalizable para la Milk-V Duo 256M, que:

- Sirve como plataforma estable para desplegar las aplicaciones de monitoreo y control del invernadero.
- Permite habilitar y probar las interfaces de hardware necesarias (SPI para LoRa, UART, GPIO, etc.).
- Cumple con los criterios de la rúbrica relacionados con el uso de herramientas de desarrollo profesional, la documentación del proceso y la reproducibilidad del entorno.

En subcapítulos posteriores se detalla el software de más alto nivel (firmware de los nodos de sensado, scripts de recepción y el servidor web) que se ejecuta sobre esta imagen de base.

## 5. Prototipado y pruebas

En esta sección se describe el proceso de construcción física del sistema RoseWatch y las pruebas realizadas para verificar el funcionamiento de los distintos módulos: nodo de sensado (Arduino + sensores + LoRa), pasarela embebida basada en la Milk-V Duo 256M y enlace de comunicaciones entre ambos.

### 5.1. Prototipado de hardware

#### 5.1.1. Nodo de sensado

El primer prototipo se construyó sobre una placa Arduino UNO, integrando los sensores DHT11 (temperatura y humedad del aire), BH1750 (iluminancia) y HW-080 (humedad de suelo), junto con un módulo LoRa SX1278 a 433 MHz. Debido a que el Arduino trabaja a 5 V y el LoRa a 3,3 V, se añadió un *level shifter* de cuatro canales para adaptar las líneas SCK, MOSI, CS y RESET desde 5 V a 3,3 V, mientras que las líneas MISO y DIO0 se conectaron directamente al Arduino.

En la Fig. 22 se muestra el montaje del nodo de sensado. Se aprecia el cableado hacia los sensores y el módulo LoRa, así como la disposición del *level shifter* entre el Arduino y el transceptor.

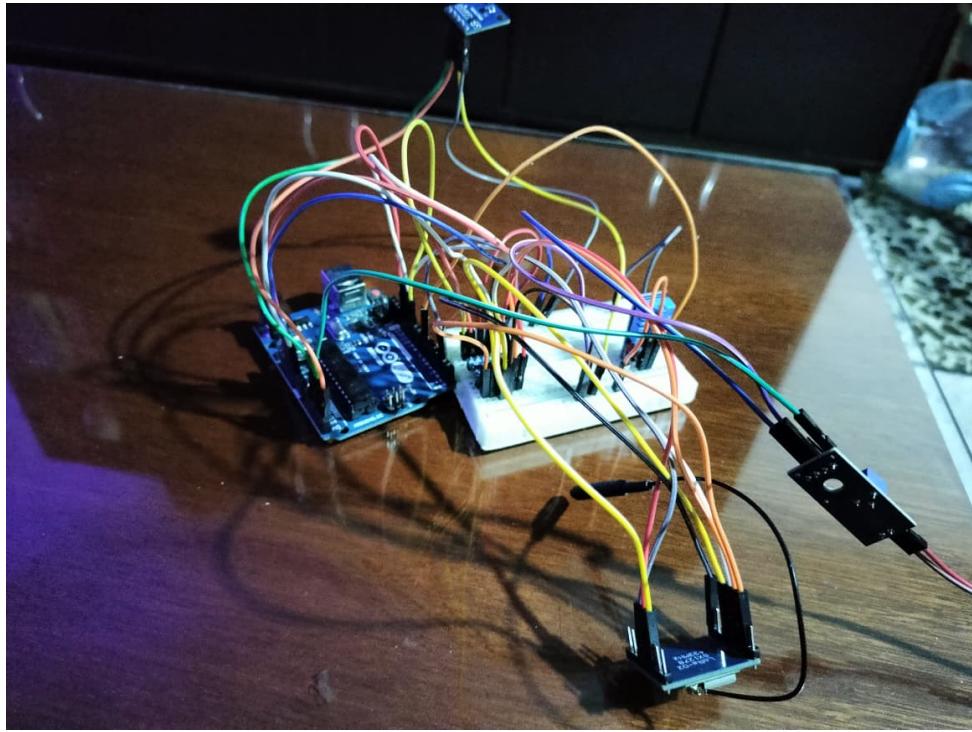


Figura 22: Prototipo físico del nodo de sensado: Arduino UNO con DHT11, BH1750, sensor de humedad de suelo HW-080, módulo LoRa SX1278 y *level shifter* para adaptación de niveles lógicos.

### 5.1.2. Pasarela con Milk-V Duo 256M

El segundo prototipo corresponde a la pasarela embebida basada en la Milk-V Duo 256M. Sobre la imagen de Linux generada en las secciones anteriores se conectó un módulo LoRa mediante el bus SPI expuesto en el puerto P de la tarjeta. La configuración de pines se realizó escribiendo directamente en los registros de multiplexación mediante la herramienta `devmem` y verificando la presencia de los dispositivos `spidev`.

En la Fig. ?? se observa la Milk-V Duo 256M con la tarjeta microSD, el módulo LoRa conectado por SPI y la alimentación externa utilizada durante las pruebas.

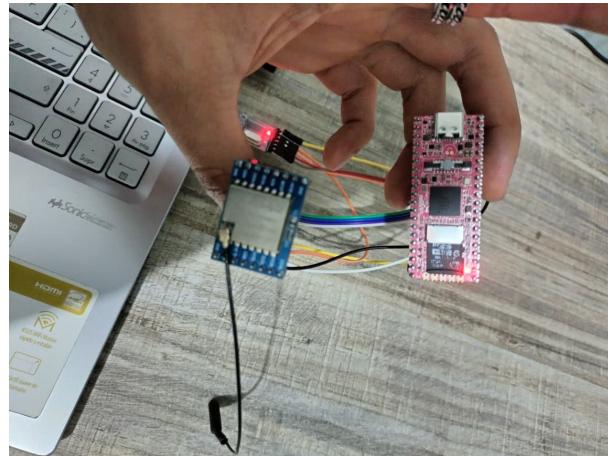


Figura 23: Prototipo de la pasarela: Milk-V Duo 256M con la imagen de Linux generada, módulo LoRa conectado por SPI y configuración de pines mediante `devmem`.

## Mapa de conexiones LoRa–Milk-V Duo 256M

La pasarela se implementó conectando un módulo LoRa SX1278 a la interfaz SPI de la Milk-V Duo 256M. En la Tabla y el siguiente esquemático se resume el mapeo de pines empleado en el prototipo, así como la justificación de cada conexión.

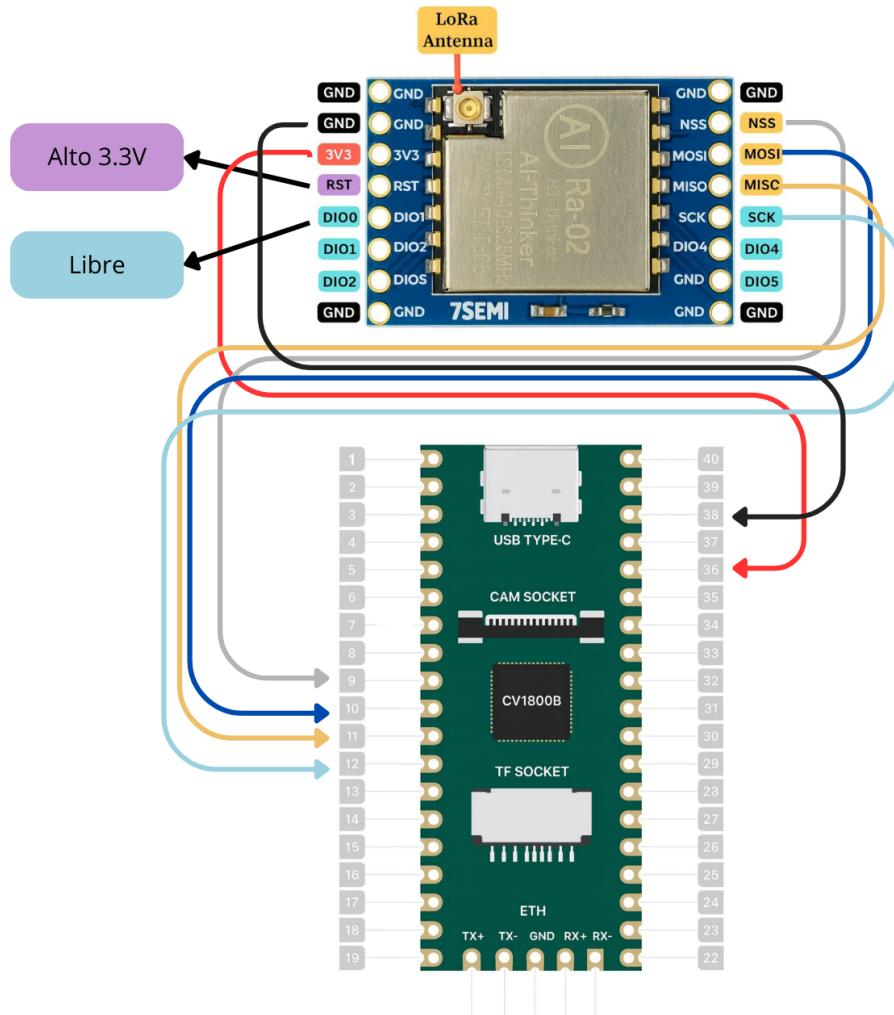


Figura 24: Esquemático de conexiones entre el Lora y la Milk-v duo 256.

Cuadro 1: Conexiones entre el módulo LoRa SX1278 y la Milk-V Duo 256M.

Pin LoRa	Señal	Pin Milk-V	Motivo de selección
VCC	3,3 V	3V3_OUT	El SX1278 trabaja a 3,3 V y la Milk-V no tolera 5 V en sus GPIO.
GND	Referencia	GND	Masa común entre la Milk-V, el LoRa y el resto del sistema.
SCK	Reloj SPI	SPI0_SCK (p. 9)	Señal de reloj del bus SPI0 usado para comunicar con el LoRa.
MOSI	Datos M→E	SPI0_MOSI (p. 10)	Datos enviados por la Milk-V hacia el SX1278 (escritura de registros).
MISO	Datos E→M	SPI0_MISO (p. 11)	Datos enviados por el SX1278 hacia la Milk-V (lectura de registros).
NSS / CS	Chip Select	SPI0_CS0 (p. 12)	Selecciona al módulo LoRa como esclavo del bus /dev/spidev0.0.
RST	Reset	3,3 V	Se mantiene en nivel alto para evitar resets por hardware durante la operación.
DIO0	RxDone (IRQ)	GPIO libre (p. ej. p. 5)	Reservado para futuras mejoras con interrupciones; en esta versión se usa sondeo por SPI.

Este mapeo es coherente con la configuración de pines realizada mediante `devmem` para habilitar el bus SPI (CS, MISO, MOSI y SCK) y con el dispositivo de usuario `/dev/spidev0.0` empleado por el programa receptor en C.

## 5.2. Prototipado de software y pruebas del nodo de sensado

Una vez montado el hardware del nodo de sensado se cargó el firmware descrito en la Sección ??, que realiza la lectura periódica de los sensores y el envío de las mediciones por LoRa en forma de cadena de texto.

Antes de integrar la pasarela, se validó el funcionamiento del firmware utilizando el monitor serie del Arduino IDE. La Fig. ?? muestra un ejemplo de salida, donde se observan:

- La temperatura y la humedad relativa entregadas por el DHT11.
- El valor crudo del sensor de humedad de suelo y el porcentaje obtenido tras la calibración (entre suelo seco y suelo húmedo).
- La iluminancia en lux medida por el BH1750.
- La trama final enviada por LoRa con el formato T,H\_aire,H\_suelo,Lux.

```
19:34:08.625 -> -----
19:34:08.659 -> Temperatura: 25.3 °C
19:34:08.659 -> Humedad ambiente: 37.5 %
19:34:08.691 -> Humedad suelo: 900 raw → 0 %
19:34:08.723 -> Iluminancia: 498 lx
19:34:08.756 -> Enviando por LoRa: 25.3,37.5,0,498
```

Figura 25: Salida del monitor serie del Arduino UNO durante las pruebas del nodo de sensado. Se muestran las lecturas de los sensores y el mensaje completo que se transmite a través del módulo LoRa.

En estas pruebas se forzaron cambios en el entorno para comprobar la respuesta de cada sensor:

- Aumentar la temperatura y la humedad cerca del DHT11 (por ejemplo, acercando la mano o un flujo de aire caliente).
- Iluminar y oscurecer el BH1750 para observar variaciones en lux.
- Humedecer y secar el sustrato donde se encontraba la sonda HW-080.

En todos los casos se observaron variaciones coherentes con la condición física, lo que permitió validar el firmware y la calibración básica del nodo de sensado antes de integrarlo con la pasarela.

### 5.3. Pruebas de la interfaz SPI y del módulo LoRa en la pasarela

En la pasarela Milk-V Duo 256M, las primeras pruebas se enfocaron en habilitar y verificar el bus SPI destinado al módulo LoRa. Tras el arranque desde la imagen generada, se comprobó la existencia de los dispositivos `/dev/spidevX.Y` y se inspeccionaron los controladores SPI registrados, tal como se muestra en la Fig. 26.

Posteriormente se detuvo el servicio de WiFi y se configuraron los pines P18, P21, P22 y P23 como señales de CS, MISO, MOSI y SCK del SPI, respectivamente, mediante las siguientes instrucciones:

```
/etc/init.d/S30wifi stop
./devmem 0x030010D0 b 0x1 # GPIO P18 -> CS del LoRa
./devmem 0x030010DC b 0x1 # GPIO P21 -> MISO
./devmem 0x030010E0 b 0x1 # GPIO P22 -> MOSI
./devmem 0x030010E4 b 0x1 # GPIO P23 -> SCK
```

Finalmente se utilizó la herramienta `spidev_test` para realizar una transferencia de prueba a través del dispositivo `/dev/spidev1.0`. La Fig. 27 muestra la salida del comando, donde se aprecian los bytes enviados y recibidos sin errores, confirmando el correcto funcionamiento del bus SPI antes de integrar el driver de LoRa.

Figura 26: Verificación de los dispositivos SPI en la Milk-V Duo 256M tras el arranque desde la imagen personalizada.

Figura 27: Salida de `spidev_test` sobre `/dev/spidev1.0`, que confirma el correcto funcionamiento de la interfaz SPI utilizada para el módulo LoRa en la pasarela.

### 5.4. Síntesis de resultados de las pruebas

Las pruebas de prototipo realizadas permiten concluir que:

- El nodo de sensado basado en Arduino UNO mide de forma coherente las variables de temperatura, humedad ambiente, humedad de suelo e iluminancia, y construye correctamente la trama de texto que se envía por LoRa.
- La pasarela Milk-V Duo 256M arranca desde la imagen generada, expone el bus SPI en `/dev/spidev1.0` y es capaz de realizar transferencias de prueba, lo que valida la configuración del kernel, del *device tree* y de los pines.
- El sistema se encuentra preparado para la integración completa del enlace LoRa nodo-pasarela y para el desarrollo de la capa de aplicación en la Milk-V (recepción, almacenamiento y visualización de las mediciones).

#### 5.4.1. Implementación del firmware y software

##### Receptor LoRa y registro de datos en la Milk-V Duo 256M

##### Firmware de recepción y registro de datos en la Milk-V Duo 256M

El archivo `lora_rx_log_v2.c` implementa el firmware de la pasarela basada en la Milk-V Duo 256M. Este programa se encarga de: (i) detectar de forma automática la configuración correcta de la interfaz SPI hacia el módulo LoRa SX1278, (ii) configurar los parámetros de modulación para que coincidan con el nodo transmisor, (iii) recibir los paquetes LoRa enviados por el Arduino y (iv) decodificar y registrar las variables agroclimáticas en un archivo CSV con marca de tiempo, RSSI y SNR. A continuación se describe el funcionamiento del código por bloques.

**Definición de registros y constantes.** En la primera parte del archivo se incluyen las cabeceras estándar de C (`stdio.h`, `stdint.h`, `unistd.h`, etc.), así como `linux/spi/spidev.h` para poder acceder al dispositivo `/dev/spidev0.0`. Posteriormente se definen las direcciones de los registros del SX1278 (`REG_FIFO`, `REG_OP_MODE`, `REG_FRF_MSB`, `REG_MODEM_CONFIG_1`, `REG_MODEM_CONFIG_2`, `REG_VERSION`, entre otros) y las constantes asociadas a los modos de operación del chip (`MODE_SLEEP`, `MODE_STDBY`, `MODE_RX_CONTINUOUS`) y a las banderas de interrupción (`IRQ_RX_DONE`, `IRQ_CRC_ERROR`). También se declaran variables globales para almacenar el descriptor del SPI (`spi_fd`), una bandera de ejecución (`running`), la configuración SPI detectada (`working_mode`, `working_speed`) y el puntero al archivo de log (`log_file`).

**Manejo de señales y apagado ordenado.** La función `signal_handler()` atiende las señales `SIGINT` y `SIGTERM` (por ejemplo, al pulsar `Ctrl+C`). Su única función es poner a cero la bandera `running` e imprimir un mensaje de salida. De esta forma, el bucle principal del programa se detiene de manera controlada y permite ejecutar las rutinas de cierre: pasar el SX1278 a modo `SLEEP`, cerrar el descriptor SPI y cerrar el archivo CSV.

**Capa de acceso SPI a bajo nivel.** El bloque *SPI básico* encapsula las operaciones de acceso al bus SPI de Linux:

- `spi_open_with_config(device, mode, speed)` abre el dispositivo `/dev/spidev0.0` con el modo SPI indicado (0–3) y la velocidad deseada. Adicionalmente fija 8 bits por palabra y el orden de bits MSB-first, utilizando las llamadas `ioctl` de `spidev`.
- `spi_read_register(reg, speed)` implementa la lectura de un registro del SX1278. Para ello envía dos bytes: la dirección con el bit más significativo en 0 (lectura) y un byte vacío; la respuesta en el segundo byte (`rx[1]`) corresponde al contenido del registro.
- `spi_write_register(reg, value, speed)` implementa la escritura en un registro. En este caso se envía la dirección con el bit más significativo en 1 (escritura) seguida del valor a escribir.

Estas funciones abstraen el detalle del uso de `SPI_IOC_MESSAGE` y permiten que el resto del firmware trabaje directamente en términos de registros del SX1278.

**Auto-detección de la configuración SPI.** Una particularidad del firmware es que no supone a priori un modo SPI ni una velocidad concreta, sino que los descubre de forma automática mediante la función `auto_detect_spi_config()`. Esta función recorre todas las combinaciones de:

- Modos SPI: `SPI_MODE_0`, `SPI_MODE_1`, `SPI_MODE_2` y `SPI_MODE_3`.

- Velocidades: 50 kHz, 100 kHz, 200 kHz y 500 kHz.

Para cada combinación: (i) abre el dispositivo `/dev/spidev0.0` con ese modo y velocidad, (ii) lee el registro `REG_VERSION` y comprueba si devuelve un valor típico de la familia SX127x (0x11, 0x12 o 0x24) y (iii) realiza un test de escritura/lectura sobre el registro `REG_SYNC_WORD`. Solo si se consigue escribir un valor de prueba (0x55), leerlo correctamente y restaurar el valor original (0x12), se considera que la configuración es válida. En ese caso se almacenan en `working_mode` y `working_speed` el modo y la velocidad que serán usados en el resto del programa. Si ninguna combinación funciona, el programa informa del error y se detiene.

**Configuración del transceptor LoRa.** Una vez establecida la comunicación SPI, la función `lora_init()` se encarga de configurar el SX1278 en modo LoRa con los mismos parámetros que el nodo transmisor basado en Arduino:

- En primer lugar se coloca el chip en modo SLEEP escribiendo `MODE_LONG_RANGE_MODE | MODE_SLEEP` en `REG_OP_MODE`, lo que garantiza que no haya actividad durante la reconfiguración.
- A continuación se fija la frecuencia central a 433 MHz mediante la función `lora_set_frequency()`, que calcula el valor de *FRF* correspondiente y lo reparte en los registros `REG_FRF_MSB`, `REG_FRF_MID` y `REG_FRF_LSB`.
- La función `lora_set_spreading_factor(12)` ajusta el factor de dispersión a SF12, sobrescribiendo los bits [7:4] del registro `REG_MODEM_CONFIG_2` y manteniendo el resto de opciones.
- La función `lora_set_bandwidth(62500)` selecciona un ancho de banda de 62,5 kHz ( $BW = 62,5 \text{ kHz}$ ), actualizando los bits de *BW* en el registro `REG_MODEM_CONFIG_1`.
- Mediante `lora_set_coding_rate(8)` se configura la tasa de codificación a  $CR = 4/8$ , modificando los bits [3:1] de `REG_MODEM_CONFIG_1`.
- En `REG_MODEM_CONFIG_3` se escribe el valor 0x0C, que activa la optimización para bajas tasas de datos (*Low Data Rate Optimize*) y el control automático de ganancia (AGC).
- El firmware desactiva el CRC de carga útil limpiando el bit correspondiente en `REG_MODEM_CONFIG_2`, de forma coherente con la configuración por defecto de la librería `LoRa.h` empleada en el Arduino transmisor.
- Se fija la palabra de sincronismo (*sync word*) en 0x12, se habilita el *LNA boost* para mejorar la sensibilidad y se configuran las direcciones base de los buffers FIFO de transmisión y recepción en 0x00.

Finalmente, el SX1278 pasa a modo **STDBY** (*standby*), quedando listo para entrar en recepción continua cuando se invoque a la función de lectura de paquetes.

**Recepción de paquetes y cálculo de RSSI/SNR.** El bloque *LORA* incluye la función `lora_receive_packet(uint8_t *buf, int max_len, int *out_rssi, int *out_snr)`, que implementa la lógica de recepción de un paquete:

1. Se limpian todas las banderas de interrupción escribiendo 0xFF en `REG_IRQ_FLAGS` y se coloca el SX1278 en modo `RX_CONTINUOUS`.
2. Se entra en un bucle de espera acotado (50 iteraciones con retardo de 100 ms), donde en cada ciclo se lee `REG_IRQ_FLAGS`. Si está activa la bandera `IRQ_RX_DONE`, significa que se ha recibido un paquete.

3. Si además la bandera IRQ\_CRC\_ERROR está activa, el paquete se descarta, se limpian las banderas y la función devuelve un error.
4. En caso contrario, se lee el número de bytes recibidos desde REG\_RX\_NB\_BYTES y la dirección del FIFO en REG\_FIFO\_RX\_CURRENT. A partir de esa dirección se ajusta el puntero interno REG\_FIFO\_ADDR\_PTR y se realiza una lectura en ráfaga del FIFO.
5. El contenido recibido se copia al buffer buf y se recuperan las métricas de enlace leyendo REG\_PKT\_RSSI\_VALUE y REG\_PKT\_SNR\_VALUE. El RSSI en dBm se obtiene mediante una aproximación del tipo RSSI = valor\_crudo - 164 (para 433 MHz) y el SNR se calcula a partir del valor crudo dividido entre 4.
6. Finalmente se limpian de nuevo las banderas de interrupción y la función devuelve la longitud del paquete recibido. En caso de *timeout* (no llega ningún paquete en el número de iteraciones previsto), se devuelve cero.

**Registro de datos en archivo CSV.** El bloque *LOG A CSV* incorpora la función `open_log_file()`, que intenta abrir un archivo `lora_datos.csv` en el directorio actual. Si no es posible, prueba con la ruta `/tmp/lora_datos.csv`. Si el archivo no existía, se escribe una cabecera con los nombres de las columnas: `fecha_hora,temp_C,humAmb_pct,humSuelo_pct,lux,rssi_dBm,snr_dB`. Posteriormente, en cada paquete correctamente decodificado, el programa genera una marca de tiempo con resolución de milisegundos (`gettimeofday()` y `localtime()`), parsea la trama recibida en formato CSV (`temp,humAmb,humSuelo,lux`) mediante `sscanf()`, e imprime en consola las variables con sus etiquetas (*Temperatura, Humedad ambiente, etc.*). Además, si el archivo de log está disponible, añade una nueva fila al CSV con la fecha y hora, las cuatro variables medidas y los valores de RSSI y SNR.

**Función principal.** En la función `main()` se registra el manejador de señales, se llama a `auto_detect_spi_config()` para encontrar una configuración válida de SPI y, si tiene éxito, se invoca a `lora_init()` para configurar el SX1278. A continuación se intenta abrir el archivo de log mediante `open_log_file()` y se entra en un bucle principal controlado por la bandera `running`. En cada iteración del bucle se llama a `lora_receive_packet()`; si se recibe un paquete, se almacena temporalmente en un buffer, se parsea la trama para obtener las variables agroclimáticas, se muestran por consola con sus etiquetas y se registran en el archivo CSV junto con RSSI y SNR. Periódicamente se imprime un mensaje de estado indicando el número total de paquetes recibidos. Cuando el usuario solicita la salida (p. ej., con `Ctrl+C`), la bandera `running` pasa a cero, el bucle termina, el transceptor LoRa se coloca en modo `SLEEP` y se cierran tanto el dispositivo SPI como el archivo de log, completando un apagado ordenado del sistema.

```

caption={Fragmento de la función de recepción de paquetes LoRa en la Milk-V.},
label={lst:lora-receive-frag}
int lora_receive_packet(uint8_t *buf, int max_len,
                      int *out_rssi, int *out_snr) {
    spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
    lora_set_mode(MODE_RX_CONTINUOUS);

    for (int i = 0; i < 50 && running; i++) {
        uint8_t irq = spi_read_register(REG_IRQ_FLAGS, working_speed);

        if (irq & IRQ_RX_DONE) {
            if (irq & IRQ_CRC_ERROR) {
                printf("[WARN] CRC error - descartando paquete\n");
                spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
                return -1;
            }
        }
    }
}

```

```

int len = spi_read_register(REG_RX_NB_BYTES, working_speed);
uint8_t fifo_addr =
    spi_read_register(REG_FIFO_RX_CURRENT, working_speed);

if (len > max_len) len = max_len;

spi_write_register(REG_FIFO_ADDR_PTR, fifo_addr, working_speed);

uint8_t tx[256] = {REG_FIFO & 0x7F};
uint8_t rx[256] = {0};

struct spi_ioc_transfer tr = {
    .tx_buf = (unsigned long)tx,
    .rx_buf = (unsigned long)rx,
    .len = len + 1,
    .speed_hz = working_speed,
    .bits_per_word = 8,
    .delay_usecs = 100,
};

if (ioctl(spi_fd, SPI_IOC_MESSAGE(1), &tr) < 1) {
    printf("[ERROR] Fallo al leer FIFO\n");
    spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
    return -1;
}

memcpy(buf, &rx[1], len);

int rssi = spi_read_register(REG_PKT_RSSI_VALUE,
                             working_speed) - 164;
int snr = (int8_t)spi_read_register(REG_PKT_SNR_VALUE,
                                    working_speed) / 4;

if (out_rssi) *out_rssi = rssi;
if (out_snr) *out_snr = snr;

spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
return len;
}

usleep(100000); // 100 ms
}

return 0; // timeout
}

```

En el `main()`, el programa entra en un bucle donde llama repetidamente a `lora_receive_packet()`, parsea la trama recibida en el formato `temp,humAmb,humSuelo,lux`, imprime las variables con sus etiquetas y registra cada muestra en un archivo CSV junto con la hora, el RSSI y el SNR del enlace. De esta forma, la pasarela no solo visualiza los datos en tiempo real, sino que también genera un registro persistente para su análisis posterior.

#### 5.4.2. Plan de pruebas

Se definió un plan de pruebas sencillo, pero alineado con los objetivos del prototipo RoseWatch y con los bloques principales del sistema (nodo de sensado, enlace LoRa y pasarela Milk-V Duo 256M). El plan se estructuró en cuatro grupos:

- Pruebas unitarias de sensores.** Verificar de forma individual que cada sensor entrega valores coherentes al modificar intencionalmente las condiciones de medida:

- DHT11: acercar una fuente de calor o respirar sobre el sensor para observar cambios en temperatura y humedad relativa.
- BH1750: variar la iluminación incidente (ambiental, interna de celular, cubrir el sensor) y comprobar la variación en lux.
- HW-080: humedecer y secar la zona de la sonda para observar el cambio en la lectura analógica y en el porcentaje de humedad de suelo calculado.

**2. Pruebas del enlace LoRa.** Comprobar que el módulo LoRa transmisor (en el Arduino) y el módulo LoRa receptor (en la Milk-V Duo 256M) se comunican correctamente:

- Enviar tramas de prueba con valores conocidos (por ejemplo, secuencias fijas o incrementales).
- Repetir la prueba a diferentes distancias dentro de un recinto y en espacios abiertos, verificando que se reciben las tramas sin errores aparentes.

**3. Pruebas de integración nodo–pasarela.** Validar el flujo completo desde la adquisición de datos hasta la recepción en la Milk-V:

- Enviar periódicamente la trama CSV con temperatura, humedad ambiente, humedad de suelo e iluminancia.
- Comprobar que el programa en C de la Milk-V recibe la trama, la separa en campos y etiqueta cada valor (por ejemplo, *Temperatura: X, Humedad: Y, ...*).
- Verificar que la información se registra correctamente en archivo o en la consola para su posterior análisis.

**4. Pruebas de visualización.** En la medida de lo posible, comprobar la presentación de los datos al usuario:

- Revisar que los valores mostrados (en consola o en la interfaz web local) sean coherentes con las mediciones observadas directamente en el invernadero o en el entorno de prueba.
- Confirmar que las actualizaciones se realizan con la frecuencia esperada.

#### 5.4.3. Resultados de las pruebas

##### 5.4.3. Resultados de las pruebas

Las pruebas realizadas sobre el prototipo permitieron verificar el funcionamiento básico de cada bloque del sistema:

- **Sensores.** El DHT11 respondió a variaciones de temperatura y humedad relativa generadas de forma controlada (calentamiento local y exhalación de aire), mostrando incrementos/-decrementos coherentes en las lecturas. El BH1750 registró cambios claros al pasar de iluminación ambiente a iluminación directa con una linterna y al cubrir completamente el sensor. El HW-080 mostró variaciones significativas al humedecer y secar el sustrato, lo que confirma su utilidad para discriminar entre suelo seco, intermedio y húmedo.

Ademas se comprobó todo esto por medio del seria monitor del id de arduino, con el fin de verificar el correcta toma de los datos y de las variaciones de estos mismos.

```

08:25:27.200 -> Temperatura: 23.5 °C
08:25:27.200 -> Humedad ambiente: 42.4 %
08:25:27.217 -> Humedad suelo: 900 raw → 0 %
08:25:27.250 -> Iluminancia: 196 lx

```

Figura 28: En esta imagen vemos la lectura correcta de los datos de cada uno de los sensores con lecturas de un ambiente promedio del interior de un laboratorio.

- **Enlace LoRa.** El nodo de sensado basado en Arduino UNO transmitió correctamente las tramas de texto con el formato definido (temperatura, humedad ambiente, humedad de suelo e iluminancia, separados por comas). En las pruebas realizadas en interior y a distancias comparables a las de un invernadero típico, el receptor en la Milk-V Duo 256M recibió las tramas de manera estable, sin pérdidas apreciables en las condiciones de prueba.

```
08:28:28.052 -> Enviando por LoRa: 23.5,42.7,0,201
```

Figura 29: imagen de envío de los datos por medio del Modulo Lora de transmisión según el formato definido

- **Integración nodo–pasarela.** El programa en C ejecutado sobre la Milk-V logró: (i) inicializar el módulo LoRa receptor a través de la interfaz SPI, (ii) detectar la llegada de cada paquete, (iii) leer el contenido de la trama y (iv) dividirla en sus campos, asignando etiquetas a cada variable (por ejemplo, *Temperatura*, *Humedad*, *Humedad de suelo*, *Luz*). De esta forma se confirmó que el flujo de datos nodo–gateway funciona de extremo a extremo.
- **Visualización.** En la etapa actual, la visualización se realiza principalmente a través de la consola de la Milk-V, donde se imprimen las lecturas acompañadas de su etiqueta y se observa su evolución en el tiempo. Estas lecturas resultan consistentes con los cambios introducidos en el entorno de prueba, lo que valida el formato de los datos y la lógica de presentación básica. La integración con una interfaz web más completa se plantea como trabajo futuro.

```

[STATUS] Escuchando... (20 recibidos)
[DATA] 15 bytes | RSSI: -53 dBm | SNR: 10 dB
[DATO] 21.4,48.9,0,288
08:03:18.999 --> Temperatura: 21.4 °C
08:03:18.999 --> Humedad ambiente: 48.9 %
08:03:18.999 --> Humedad suelo: 0 %
08:03:18.999 --> Iluminancia: 288 lx

[INFO] Total: 16 paquetes
[A] [RX] 15 bytes | RSSI: -49 dBm | SNR: 10 dB
[DATO] 21.5,48.8,0,288
08:03:28.960 --> Temperatura: 21.5 °C
08:03:28.960 --> Humedad ambiente: 48.8 %
08:03:28.960 --> Humedad suelo: 0 %
08:03:28.960 --> Iluminancia: 288 lx

[INFO] Total: 17 paquetes
[?] [RX] 15 bytes | RSSI: -37 dBm | SNR: 11 dB
[DATO] 21.5,48.8,0,227
08:03:39.012 --> Temperatura: 21.5 °C
08:03:39.012 --> Humedad ambiente: 48.7 %
08:03:39.012 --> Humedad suelo: 0 %
08:03:39.012 --> Iluminancia: 227 lx

[INFO] Total: 18 paquetes
[G] [RX] 15 bytes | RSSI: -48 dBm | SNR: 10 dB
[DATO] 21.6,48.5,0,227
08:03:48.960 --> Temperatura: 21.6 °C
08:03:48.960 --> Humedad ambiente: 48.5 %
08:03:48.960 --> Humedad suelo: 0 %
08:03:48.960 --> Iluminancia: 227 lx

[INFO] Total: 19 paquetes
[B] [RX] 15 bytes | RSSI: -48 dBm | SNR: 10 dB
[DATO] 21.7,48.4,0,222
08:03:59.012 --> Temperatura: 21.7 °C
08:03:59.012 --> Humedad ambiente: 48.4 %
08:03:59.012 --> Humedad suelo: 0 %
08:03:59.012 --> Iluminancia: 222 lx

[INFO] Total: 20 paquetes
[STATUS] Escuchando... (20 recibidos)
CTRL-A Z For help | 315288 BNI | NOR | Minicom 2.9 | VT102 | Desconectado | ttys008

```

Figura 30: Datos recibidos por la Milk-v Duo 256 e imprimidos por el minicom.

En conjunto, estos resultados muestran que el prototipo RoseWatch es capaz de medir las variables agroclimáticas definidas, transmitirlas mediante un enlace LoRa punto a punto y reci-

birlas en la pasarela basada en la Milk-V Duo 256M, constituyendo una base sólida para extender el sistema a escenarios con más nodos y capacidades adicionales.

## 6. manejo de información

Para realizar la compilación de los datos de nuestro proyecto Primero almacenamos absolutamente todos los datos en la memoria de nuestra Milk-v duo 256 y posterior utilizamos distintos comandos para subir los documentos a nuestro repositorio de utilizamos Github.

El link de nuestro repositorio es: <https://github.com/Cbetin/EmbebidosRoseWatch>

## 7. Resultados y análisis

En esta sección se presentan y analizan los resultados obtenidos con el prototipo RoseWatch, integrando tanto las pruebas individuales de los sensores como las pruebas del enlace LoRa y del firmware de recepción en la Milk-V Duo 256M. El objetivo no fue lograr un producto comercial final, sino demostrar experimentalmente la viabilidad técnica del sistema propuesto y comprobar que los bloques de hardware y software interactúan de forma coherente.

Desde el punto de vista de la captación de datos, las pruebas realizadas en laboratorio permitieron verificar el comportamiento esperado de cada sensor. El DHT11 respondió de forma clara a variaciones controladas de temperatura y humedad relativa, como el acercamiento de una fuente de calor o la exhalación de aire, mostrando incrementos y decrementos consistentes en las lecturas. El BH1750 registró cambios significativos al pasar de condiciones de iluminación ambiente a iluminación directa (por ejemplo, con una linterna de celular) y al cubrir completamente el sensor, lo que confirma su capacidad para discriminar distintos niveles de iluminancia. Finalmente, el sensor HW-080 de humedad de suelo mostró variaciones apreciables en la señal analógica al humedecer y secar el sustrato, lo que permitió mapear el valor crudo a un porcentaje de humedad y diferenciar entre suelo seco, intermedio y húmedo. Estas observaciones se corroboraron mediante el monitor serie del entorno de Arduino, donde se visualizaron las lecturas en tiempo real.

En cuanto al enlace LoRa, el nodo de sensado basado en Arduino UNO fue capaz de transmitir de manera estable tramas de texto con el formato definido (`temperatura, humAmbiente, humSuelo, lux`). En las pruebas realizadas en interior y a distancias comparables a las de un invernadero típico, el receptor implementado en la Milk-V Duo 256M recibió las tramas de forma consistente, sin pérdidas apreciables en las condiciones de prueba. El uso de parámetros de modulación robustos (433 MHz, SF12, BW = 62,5 kHz y CR = 4/8) se tradujo en enlaces con buena sensibilidad a costa de una menor velocidad de transmisión, lo cual es adecuado para aplicaciones de monitoreo donde el periodo de muestreo es del orden de segundos y no se requiere un gran caudal de datos.

La integración nodo–pasarela también arrojó resultados satisfactorios. El programa en C ejecutado sobre la Milk-V pudo: (i) detectar de forma automática la configuración correcta de la interfaz SPI hacia el SX1278, (ii) inicializar el transceptor LoRa con parámetros compatibles con el Arduino transmisor, (iii) recibir paquetes en modo de recepción continua, (iv) extraer el contenido del FIFO y (v) parsear la trama CSV en las cuatro variables agroclimáticas de interés. Además, se logró calcular e imprimir para cada paquete los valores de RSSI y SNR reportados por el propio SX1278, lo que permite contar con indicadores básicos de calidad de enlace. Las tramas recibidas se almacenaron correctamente en un archivo CSV junto con una marca de tiempo, facilitando el análisis posterior de la evolución de las variables y del comportamiento del enlace.

Desde la perspectiva de la visualización, en el estado actual del proyecto los datos se presentan principalmente a través de la consola de la Milk-V Duo 256M, donde se imprimen las variables acompañadas de sus etiquetas (*Temperatura*, *Humedad ambiente*, *Humedad de suelo*, *Iluminancia*) y se observa su evolución en el tiempo. Aunque esta interfaz es básica, resultó suficiente para verificar la coherencia entre los valores medidos y las condiciones reales del entorno de prueba, y para confirmar que el sistema funciona de extremo a extremo.

En conjunto, los resultados obtenidos permiten concluir que el prototipo RoseWatch es capaz de medir las variables agroclimáticas definidas, transmitirlas mediante un enlace LoRa punto a punto y recibirlas en la pasarela basada en la Milk-V Duo 256M, con registro persistente en archivos CSV. Aunque las pruebas se realizaron principalmente en un entorno de laboratorio y con un único nodo de sensado, el comportamiento observado indica que la arquitectura propuesta es viable y constituye una base sólida para su posterior despliegue y ampliación en escenarios reales de invernaderos.

## 8. Conclusiones y trabajo futuro

### 8.1. Conclusiones

El desarrollo de RoseWatch permitió integrar en un solo prototipo varios conceptos claves de los sistemas embebidos modernos: sensado distribuido, comunicaciones de largo alcance de baja tasa (LoRa), pasarelas embebidas basadas en Linux y registro de datos para análisis posterior. A partir del trabajo realizado se pueden extraer las siguientes conclusiones principales:

En primer lugar, se demostró la viabilidad de utilizar un nodo de sensado de bajo costo, basado en Arduino UNO y sensores comerciales (DHT11, BH1750 y HW-080), para medir variables agroclimáticas relevantes en un invernadero de rosas. Las pruebas en laboratorio mostraron que, aunque estos sensores no alcanzan la precisión de equipos industriales, sí permiten capturar de manera cualitativa y consistente las variaciones de temperatura, humedad relativa, humedad de suelo e iluminancia que resultan críticas para la detección temprana de condiciones de riesgo, como las heladas.

En segundo lugar, se validó que la tecnología LoRa constituye una alternativa adecuada para transmitir estas variables a lo largo de distancias comparables a las dimensiones de un invernadero, sin necesidad de infraestructura de red convencional ni acceso a Internet. La configuración utilizada (433 MHz, SF12, BW = 62,5 kHz, CR = 4/8) permitió establecer enlaces robustos en las condiciones de prueba, con tasas de datos suficientes para el envío periódico de tramas de monitorización. Esto confirma que LoRa es una tecnología pertinente para aplicaciones de monitoreo agroclimático en entornos rurales o con conectividad limitada.

En tercer lugar, la utilización de la Milk-V Duo 256M como pasarela demostró que es posible implementar sobre una plataforma de bajo costo y arquitectura RISC-V un sistema de recepción, procesamiento ligero y registro de datos en Linux. El trabajo de configuración de la imagen del sistema, habilitación del SPI de usuario (`spi-dev`) y programación en C del receptor LoRa permitió adquirir experiencia práctica en temas como compilación cruzada, manejo de periféricos desde espacio de usuario y diseño de firmware orientado a registro de datos (data-logging).

En cuarto lugar, el prototipo desarrollado cumplió los objetivos planteados para este proyecto de curso: se logró cerrar el ciclo completo desde la medición en el nodo de sensado hasta la recepción, etiquetado y almacenamiento de los datos en la pasarela. Aunque el sistema actual solo contempla un nodo de sensado y una interfaz de visualización básica en consola, la arquitectura hardware y software implementada es extensible tanto a escenarios con múltiples nodos como a

interfaces de usuario más amigables, lo cual se plantea explícitamente como trabajo futuro.

Finalmente, más allá de los resultados técnicos, el proyecto evidenció la importancia de conectar problemas reales del entorno (en este caso, la vulnerabilidad de los invernaderos de rosas a las heladas) con soluciones tecnológicas concretas basadas en sistemas embebidos. RoseWatch no pretende sustituir de inmediato los sistemas industriales existentes, pero sí ofrece una prueba de concepto funcional y una base de conocimiento sobre la cual se pueden construir futuras versiones más robustas, escalables y cercanas a las necesidades de productores y técnicos del sector florícola.

## 8.2. Trabajo futuro

El prototipo desarrollado en este proyecto demuestra la viabilidad de utilizar un enlace LoRa punto a punto para el monitoreo agroclimático en invernaderos de rosas. Sin embargo, aún existen múltiples líneas de mejora y extensión que pueden abordarse en trabajos futuros.

En primer lugar, resulta fundamental evolucionar desde la arquitectura actual de un solo nodo de sensado hacia una **red con múltiples nodos distribuidos** dentro del invernadero o incluso en varios invernaderos. Para ello sería necesario definir un esquema de direccionamiento de nodos (por ejemplo, incluyendo un identificador de nodo en la trama transmitida), mecanismos simples de acceso al medio para evitar colisiones (ventanas de tiempo, turnos de transmisión, etc.) y estrategias de retransmisión en caso de pérdida de paquetes. Esta ampliación permitiría obtener un mapa espacial mucho más rico de temperatura, humedad e iluminación, mejorando la capacidad de detección de condiciones críticas para el cultivo.

En segundo lugar, es deseable desarrollar una **interfaz de visualización más amigable y completa** sobre la pasarela basadas en la Milk-V Duo 256M. Actualmente los datos se muestran principalmente en consola; como trabajo futuro se propone diseñar un panel web local que permita visualizar las variables en tiempo real, consultar históricos, graficar tendencias por fecha y por nodo, y configurar umbrales de alerta desde un navegador, sin necesidad de acceder por terminal al dispositivo. Esta capa de presentación facilitaría la adopción del sistema por parte de usuarios no técnicos (productores, encargados de invernadero, etc.).

Una tercera línea de trabajo consiste en mejorar sustancialmente la **lógica de detección y alerta de heladas**. En lugar de basarse únicamente en umbrales fijos de temperatura, se plantea diseñar y validar un *modelo matemático* que combine todas las variables medidas (temperatura, humedad relativa, humedad del suelo e iluminancia) e incorpore conceptos como punto de rocío, gradientes de temperatura y velocidad de cambio. A partir de series históricas de datos del invernadero y de registros de eventos reales de heladas, dicho modelo podría calibrarse para estimar un índice de riesgo y emitir alertas más precisas y con mayor anticipación, reduciendo falsos positivos y falsos negativos.

Adicionalmente, se identifican otras mejoras posibles: la integración del sistema con una base de datos ligera para almacenamiento a largo plazo, la alimentación autónoma de los nodos mediante energía solar, el diseño de carcasa robustas para uso en ambiente de invernadero, y la evaluación de tecnologías complementarias (LoRaWAN, WiFi, 4G) para facilitar el acceso remoto a los datos desde fuera de la finca. Todas estas extensiones permitirían transformar el prototipo actual en una solución más completa y escalable para el monitoreo y la gestión preventiva de heladas en cultivos de rosas de exportación.

## Referencias

- [1] S. Zhdanov, “LicheeRV Nano — Board programming (Part 2),” *Medium*, 2025. [En línea]. Disponible en: <https://medium.com/@ret7020/licheerv-nano-board-programming-part-2-a10e33b0e110>
- [2] milkv-duo, “Milk-V Duo Official buildroot SDK,” GitHub repository. [En línea]. Disponible en: <https://github.com/milkv-duo/duo-buildroot-sdk>
- [3] Milk-V, “Milk-V Duo — Overview,” documentación oficial. [En línea]. Disponible en: <https://milkv.io/docs/duo/overview>
- [4] RT-Thread IoT OS, “Getting Started with a New RISC-V MPU Platform: RT-Thread Hands-on Milk-V Duo Guide,” *Medium*, 2024. [En línea]. Disponible en: <https://rt-thread.medium.com/getting-started-with-a-new-risc-v-platform-rt-thread-hands-on-milk-v-duo-guide-ba5c9eff5>
- [5] U-Boot Project, “LicheeRV Nano — SG2002 RISC-V SoC,” *U-Boot Documentation*. [En línea]. Disponible en: [https://docs.u-boot.org/en/stable/board/sophgo/licheerv\\_nano.html](https://docs.u-boot.org/en/stable/board/sophgo/licheerv_nano.html)
- [6] Sipeed, “LicheeRV-Nano-Build,” GitHub repository. [En línea]. Disponible en: <https://github.com/sipeed/LicheeRV-Nano-Build>
- [7] Semtech Corp., “SX1276/77/78/79 — 137 MHz to 1020 MHz Low Power Long Range Transceiver,” datasheet, Rev. 4, 2015. [En línea]. Disponible en: [https://cdn-shop.adafruit.com/product-files/3179/sx1276\\_77\\_78\\_79.pdf](https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf)
- [8] Semtech Corp., “SX1276/77/78/79 LoRa Modem Datasheet: Low Power Transceiver Overview,” reproducción en Studocu. [En línea]. Disponible en: <https://www.studocu.com/co/document/universidad-de-narino/sistemas-electricos/sx1276777879-lora-modem-datasheet-low-power-transceiver-overview/119529505>
- [9] sandeepmistry, “How to configurate LoRa for best distance?,” Issue #90, *arduino-LoRa* GitHub repository, 2018. [En línea]. Disponible en: <https://github.com/sandeepmistry/arduino-LoRa/issues/90>
- [10] Linux Kernel Documentation, “SPI userspace API (spidev),” [En línea]. Disponible en: <https://www.kernel.org/doc/Documentation/spi/spidev>

## A. Código fuente Arduino (transmisor)

```
#include <SPI.h>
#include <LoRa.h>
#include <DHT.h>
#include <Wire.h>
#include <BH1750.h>

// Configuración de pines
// LoRa (Arduino UNO)
const int csPin = 10; // CS / NSS
const int resetPin = 9; // RESET
const int irqPin = 2; // DIO0 / IRQ (pin con interrupción)

// DHT11
const int dhtPin = 7;
```

```

#define DHTTYPE DHT11
DHT dht(dhtPin, DHTTYPE);

// Sensor de humedad de suelo HW-080 (salida analgica)
const int sensorSueloPin = A2;

// Calibracion del sensor de suelo
const int rawSeco = 900;
const int rawHumedo = 500;

// BH1750 (IC: SDA=A4, SCL=A5 en UNO)
BH1750 lightMeter;

void setup() {
    Serial.begin(9600);
    delay(100);

    // Inicializar DHT11
    dht.begin();
    Serial.println("DHT11: inicializado");

    // Inicializar BH1750
    Wire.begin();
    if (lightMeter.begin()) {
        Serial.println("BH1750: OK");
    } else {
        Serial.println("BH1750: ERROR al iniciar");
    }

    // Inicializar LoRa
    LoRa.setPins(csPin, resetPin, irqPin);
    if (!LoRa.begin(433E6)) {
        Serial.println("LoRa: ERROR al iniciar");
        while (true);
    }
    LoRa.setTxPower(14); // Aumentamos potencia para compensar UNO
    LoRa.setSpreadingFactor(12); // SF12
    LoRa.setSignalBandwidth(62500); // 62.5 kHz
    LoRa.setCodingRate4(8); // CR 4/8

    Serial.println("== Sistema de sensores con LoRa (UNO) listo ==");
}

void loop() {
    // Lectura DHT11
    float temperatura = dht.readTemperature();
    float humedadAmb = dht.readHumidity();
    if (isnan(temperatura) || isnan(humedadAmb)) {
        Serial.println("DHT11: ERROR al leer");
    }

    // Lectura y calibracion HW-080
    int rawSoil = analogRead(sensorSueloPin);
    rawSoil = constrain(rawSoil, rawHumedo, rawSeco);
    int humedadSoilPct = map(rawSoil, rawHumedo, rawSeco, 100, 0);

    // Lectura BH1750
    float iluminanciaLux = lightMeter.readLightLevel();
    if (iluminanciaLux < 0) {
        Serial.println("BH1750: ERROR al leer");
    }

    // Construir mensaje
}

```

```

String mensaje = "";
mensaje = String(temperatura,1) + ",";
mensaje += String(humedadAmb,1) + ",";
mensaje += String(humedadSoilPct) + ",";
mensaje += String(iluminanciaLux,0);

// Depuración Serial
Serial.println(F("-----"));
if (!isnan(temperatura) && !isnan(humedadAmb)) {
    Serial.print("Temperatura: "); Serial.print(temperatura,1); Serial.println(" C");
    Serial.print("Humedad ambiente: "); Serial.print(humedadAmb,1); Serial.println(" %");
}
Serial.print("Humedad suelo: "); Serial.print(rawSoil); Serial.print(" raw ");
Serial.print(humedadSoilPct); Serial.println(" %");
if (iluminanciaLux >= 0) {
    Serial.print("Iluminancia: "); Serial.print(iluminanciaLux,0); Serial.println(" lx");
}
Serial.print("Enviando por LoRa: ");
Serial.println(mensaje);

// Transmisión LoRa
LoRa.beginPacket();
LoRa.print(mensaje);
LoRa.endPacket();

delay(7000); // Espera 7 s antes de la siguiente lectura
}

```

## B. Código fuente receptor / interfaz

```

// lora_rx_log_v2.c
// Receptor LoRa + log a CSV (Milk-V Duo 256M)

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/spi/spidev.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>

// ===== REGISTROS SX1278 =====
#define REG_FIFO 0x00
#define REG_OP_MODE 0x01
#define REG_FRF_MSB 0x06
#define REG_FRF_MID 0x07
#define REG_FRF_LSB 0x08
#define REG_PA_CONFIG 0x09
#define REG_LNA 0x0C
#define REG_FIFO_ADDR_PTR 0x0D
#define REG_FIFO_TX_BASE_ADDR 0x0E
#define REG_FIFO_RX_BASE_ADDR 0x0F
#define REG_FIFO_RX_CURRENT 0x10
#define REG_IRQ_FLAGS 0x12
#define REG_RX_NB_BYTES 0x13
#define REG_PKT_SNR_VALUE 0x19

```

```

#define REG_PKT_RSSI_VALUE 0x1A
#define REG_MODEM_CONFIG_1 0x1D
#define REG_MODEM_CONFIG_2 0x1E
#define REG_MODEM_CONFIG_3 0x26
#define REG_SYNC_WORD 0x39
#define REG_VERSION 0x42

// Modos
#define MODE_LONG_RANGE_MODE 0x80
#define MODE_SLEEP 0x00
#define MODE_STDBY 0x01
#define MODE_RX_CONTINUOUS 0x05

// IRQ
#define IRQ_RX_DONE 0x40
#define IRQ_CRC_ERROR 0x20

static int spi_fd = -1;
static volatile int running = 1;
static uint32_t working_speed = 0;
static uint8_t working_mode = 0;

// Log a archivo
static FILE *log_file = NULL;

// ====== MANEJO DE SEALES ======
void signal_handler(int sig) {
    (void)sig;
    running = 0;
    printf("\n[INFO] Saliendo...\n");
}

// ====== SPI BSICO ======
int spi_open_with_config(const char *device, uint8_t mode, uint32_t speed) {
    int fd = open(device, O_RDWR);
    if (fd < 0) {
        return -1;
    }

    uint8_t bits = 8;
    uint8_t lsb = 0; // MSB first

    if (ioctl(fd, SPI_IOC_WR_MODE, &mode) < 0) {
        close(fd);
        return -1;
    }

    if (ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits) < 0) {
        close(fd);
        return -1;
    }

    if (ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed) < 0) {
        close(fd);
        return -1;
    }

    // MSB first
    ioctl(fd, SPI_IOC_WR_LSB_FIRST, &lsb);

    return fd;
}

```

```

uint8_t spi_read_register(uint8_t reg, uint32_t speed) {
    usleep(200);
    uint8_t tx[2] = {reg & 0x7F, 0x00};
    uint8_t rx[2] = {0, 0};

    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
        .len = 2,
        .speed_hz = speed,
        .bits_per_word = 8,
        .delay_usecs = 100,
        .cs_change = 0,
    };

    if (ioctl(spi_fd, SPI_IOC_MESSAGE(1), &tr) < 1) {
        return 0;
    }

    usleep(200);
    return rx[1];
}

void spi_write_register(uint8_t reg, uint8_t value, uint32_t speed) {
    usleep(200);

    uint8_t tx[2] = {reg | 0x80, value};
    uint8_t rx[2] = {0, 0};

    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
        .len = 2,
        .speed_hz = speed,
        .bits_per_word = 8,
        .delay_usecs = 100,
        .cs_change = 0,
    };

    ioctl(spi_fd, SPI_IOC_MESSAGE(1), &tr);
    usleep(200);
}
}

// ====== AUTO-DETECCIN SPI ======
int auto_detect_spi_config(void) {
    printf("\n[INFO] Buscando configuración correcta de SPI...\n");
    printf(" Esto puede tardar unos segundos...\n\n");

    uint32_t speeds[] = {50000, 100000, 200000, 500000};
    uint8_t modes[] = {SPI_MODE_0, SPI_MODE_1, SPI_MODE_2, SPI_MODE_3};

    for (int m = 0; m < 4; m++) {
        for (int s = 0; s < 4; s++) {
            if (spi_fd >= 0) {
                close(spi_fd);
            }

            spi_fd = spi_open_with_config("/dev/spidev0.0", modes[m], speeds[s]);
            if (spi_fd < 0) {
                continue;
            }

            uint8_t version = spi_read_register(REG_VERSION, speeds[s]);

```

```

        if (version == 0x12 || version == 0x11 || version == 0x24) {
            // Test de escritura
            spi_write_register(REG_SYNC_WORD, 0x55, speeds[s]);
            usleep(1000);
            uint8_t test_read = spi_read_register(REG_SYNC_WORD, speeds[s]);

            if (test_read == 0x55) {
                printf("[OK] Configuración encontrada!\n");
                printf(" Modo SPI: %d\n", modes[m]);
                printf(" Velocidad: %u Hz\n", speeds[s]);
                printf(" Versión chip: 0x%02X\n", version);

                working_mode = modes[m];
                working_speed = speeds[s];

                // Restaurar SYNC_WORD por defecto
                spi_write_register(REG_SYNC_WORD, 0x12, speeds[s]);
                return 0;
            }
        }
    }

    printf("[ERROR] No se encontró configuración válida\n");
    return -1;
}

// ===== LORA =====
void lora_set_mode(uint8_t mode) {
    spi_write_register(REG_OP_MODE, MODE_LONG_RANGE_MODE | mode, working_speed);
    usleep(10000);
}

void lora_set_frequency(long frequency) {
    uint64_t frf = ((uint64_t)frequency << 19) / 32000000;

    spi_write_register(REG_FRF_MSB, (uint8_t)(frf >> 16), working_speed);
    spi_write_register(REG_FRF_MID, (uint8_t)(frf >> 8), working_speed);
    spi_write_register(REG_FRF_LSB, (uint8_t)(frf >> 0), working_speed);

    printf("[OK] Frecuencia: %ld Hz\n", frequency);
}

void lora_set_spreading_factor(int sf) {
    if (sf < 6 || sf > 12) sf = 7;

    uint8_t config2 = spi_read_register(REG_MODEM_CONFIG_2, working_speed);
    config2 = (config2 & 0x0F) | ((sf << 4) & 0xF0);
    spi_write_register(REG_MODEM_CONFIG_2, config2, working_speed);

    printf("[OK] SF: %d\n", sf);
}

void lora_set_bandwidth(long bw) {
    int bw_val;

    if (bw <= 7800) bw_val = 0;
    else if (bw <= 10400) bw_val = 1;
    else if (bw <= 15600) bw_val = 2;
    else if (bw <= 20800) bw_val = 3;
    else if (bw <= 31250) bw_val = 4;
    else if (bw <= 41700) bw_val = 5;
}

```

```

    else if (bw <= 62500) bw_val = 6;
    else if (bw <= 125000) bw_val = 7;
    else if (bw <= 250000) bw_val = 8;
    else bw_val = 9;

    uint8_t config1 = spi_read_register(REG_MODEM_CONFIG_1, working_speed);
    config1 = (config1 & 0x0F) | (bw_val << 4);
    spi_write_register(REG_MODEM_CONFIG_1, config1, working_speed);

    printf("[OK] BW: %ld Hz\n", bw);
}

void lora_set_coding_rate(int cr) {
    if (cr < 5) cr = 5;
    if (cr > 8) cr = 8;

    int cr_val = cr - 4;

    uint8_t config1 = spi_read_register(REG_MODEM_CONFIG_1, working_speed);
    config1 = (config1 & 0xF1) | (cr_val << 1);
    spi_write_register(REG_MODEM_CONFIG_1, config1, working_speed);

    printf("[OK] CR: 4/%d\n", cr);
}

int lora_init(void) {
    printf("\n[INFO] Inicializando LoRa...\n");

    lora_set_mode(MODE_SLEEP);

    // Config igual que el Arduino
    lora_set_frequency(433000000); // 433 MHz
    lora_set_spreading_factor(12); // SF12
    lora_set_bandwidth(62500); // 62.5 kHz
    lora_set_coding_rate(8); // CR 4/8

    // MODEM_CONFIG_3: AGC + LowDataRateOptimize
    spi_write_register(REG_MODEM_CONFIG_3, 0x0C, working_speed);
    printf("[OK] MODEM_CONFIG_3: AGC + LowDataRateOptimize\n");

    // CRC OFF para coincidir con Arduino (LoRa.h por defecto sin CRC)
    uint8_t config2 = spi_read_register(REG_MODEM_CONFIG_2, working_speed);
    config2 &= ~0x04; // RxPayloadCrcOn = 0
    spi_write_register(REG_MODEM_CONFIG_2, config2, working_speed);
    printf("[OK] CRC: OFF\n");

    // Sync word
    spi_write_register(REG_SYNC_WORD, 0x12, working_speed);

    // LNA boost
    uint8_t lna = spi_read_register(REG_LNA, working_speed);
    lna |= 0x03;
    spi_write_register(REG_LNA, lna, working_speed);

    // FIFO base
    spi_write_register(REG_FIFO_TX_BASE_ADDR, 0x00, working_speed);
    spi_write_register(REG_FIFO_RX_BASE_ADDR, 0x00, working_speed);

    lora_set_mode(MODE_STDBY);

    printf("[OK] LoRa inicializado\n\n");
    return 0;
}

```

```

// NUEVO: devolvemos tambin RSSI y SNR
int lora_receive_packet(uint8_t *buf, int max_len, int *out_rssi, int *out_snr) {
    spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
    lora_set_mode(MODE_RX_CONTINUOUS);

    for (int i = 0; i < 50 && running; i++) {
        uint8_t irq = spi_read_register(REG_IRQ_FLAGS, working_speed);

        if (irq & IRQ_RX_DONE) {
            if (irq & IRQ_CRC_ERROR) {
                printf("[WARN] CRC error - descartando paquete\n");
                spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
                return -1;
            }

            int len = spi_read_register(REG_RX_NB_BYTES, working_speed);
            uint8_t fifo_addr = spi_read_register(REG_FIFO_RX_CURRENT, working_speed);

            if (len > max_len) {
                printf("[WARN] Paquete demasiado grande: %d bytes (max: %d)\n", len, max_len);
                len = max_len;
            }

            spi_write_register(REG_FIFO_ADDR_PTR, fifo_addr, working_speed);
            usleep(1000);

            uint8_t tx[256] = {REG_FIFO & 0x7F};
            uint8_t rx[256] = {0};

            struct spi_ioc_transfer tr = {
                .tx_buf = (unsigned long)tx,
                .rx_buf = (unsigned long)rx,
                .len = len + 1,
                .speed_hz = working_speed,
                .bits_per_word = 8,
                .delay_usecs = 100,
                .cs_change = 0,
            };

            if (ioctl(spi_fd, SPI_IOC_MESSAGE(1), &tr) < 1) {
                printf("[ERROR] Fallo al leer FIFO\n");
                spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
                return -1;
            }
        }

        memcpy(buf, &rx[1], len);

        int rss = spi_read_register(REG_PKT_RSSI_VALUE, working_speed) - 164;
        int snr = (int8_t)spi_read_register(REG_PKT_SNR_VALUE, working_speed) / 4;

        if (out_rssi) *out_rssi = rss;
        if (out_snr) *out_snr = snr;

        printf("[RX] %d bytes | RSSI: %d dBm | SNR: %d dB\n", len, rss, snr);

        spi_write_register(REG_IRQ_FLAGS, 0xFF, working_speed);
        return len;
    }

    usleep(100000);
}

```

```

        return 0; // timeout
    }

// ===== LOG A CSV =====
int open_log_file(void) {
    const char *paths[] = {
        "lora_datos.csv", // 1) Directorio actual
        "/tmp/lora_datos.csv", // 2) Carpeta temporal (si lo anterior falla)
        NULL
    };

    for (int i = 0; paths[i] != NULL; i++) {
        const char *path = paths[i];
        int exists = 0;

        FILE *test = fopen(path, "r");
        if (test) {
            exists = 1;
            fclose(test);
        }

        log_file = fopen(path, "a");
        if (!log_file) {
            // No matamos el programa, probamos con la siguiente ruta
            perror("[WARN] No se pudo abrir/crear archivo de log");
            continue;
        }

        if (!exists) {
            // Cabecera CSV
            fprintf(log_file, "fecha_hora,temp_C,humAmb_pct,humSuelo_pct,lux,rssi_dBm,snr_dB\n");
            ffflush(log_file);
        }
    }

    printf("[LOG] Guardando datos en: %s\n", path);
    return 0; // xito
}

// Si llegamos aqui, ninguna ruta funcion
fprintf(stderr,
    "[WARN] No se pudo crear archivo de log en ninguna ruta.\n"
    " Se continuar solo con salida por consola.\n");
log_file = NULL;
return -1;
}

// ===== MAIN =====
int main(void) {
    signal(SIGINT, signal_handler);
    signal(SIGTERM, signal_handler);

    printf("=====\\n");
    printf(" Receptor LoRa Ra-02 + Log CSV\\n");
    printf(" Milk-V Duo 256M\\n");
    printf("=====\\n");

    if (auto_detect_spi_config() != 0) {
        fprintf(stderr, "\\n[ERROR] No se pudo detectar el chip LoRa\\n");
        return 1;
    }

    if (lora_init() != 0) {

```

```

        close(spi_fd);
        return 1;
    }

    if (open_log_file() != 0) {
        // seguimos sin archivo de log
        fprintf(stderr, "[WARN] Continuando sin registro en CSV (solo consola).\n");
    }

    printf("[INFO] Esperando paquetes... (Ctrl+C para salir)\n\n");

    uint8_t buffer[256];
    int count = 0;

    while (running) {
        int rssi = 0, snr = 0;
        int len = lora_receive_packet(buffer, sizeof(buffer) - 1, &rssi, &snr);

        if (len > 0) {
            buffer[len] = '\0';
            count++;

            // 1) Ver el string crudo
            printf("[DATO] %s\n", buffer);

            // 2) Parsear: temp,humAmb,humSuelo,lux
            float temp = 0.0f;
            float humAmb = 0.0f;
            int humSuelo = 0;
            int lux = 0;

            int parsed = sscanf((char *)buffer, "%f,%f,%d,%d",
                                &temp, &humAmb, &humSuelo, &lux);

            if (parsed == 4) {
                struct timeval tv;
                gettimeofday(&tv, NULL);
                struct tm *tm_info = localtime(&tv.tv_sec);

                int h = tm_info->tm_hour;
                int m = tm_info->tm_min;
                int s = tm_info->tm_sec;
                long ms = tv.tv_usec / 1000;

                char fecha[64];
                strftime(fecha, sizeof(fecha), "%Y-%m-%d %H:%M:%S", tm_info);

                // 3) Imprimir estilo pestaa Serial Arduino
                printf("%02d:%02d:%02d.%03ld -> Temperatura: %.1f C\n",
                       h, m, s, ms, temp);
                printf("%02d:%02d:%02d.%03ld -> Humedad ambiente: %.1f %%\n",
                       h, m, s, ms, humAmb);
                printf("%02d:%02d:%02d.%03ld -> Humedad suelo: %d %%\n",
                       h, m, s, ms, humSuelo);
                printf("%02d:%02d:%02d.%03ld -> Iluminancia: %d lx\n\n",
                       h, m, s, ms, lux);

                // 4) Guardar en CSV
                if (log_file) {
                    fprintf(log_file, "%s,%.1f,%.1f,%d,%d,%d,%d\n",
                            fecha, temp, humAmb, humSuelo, lux, rssi, snr);
                    fflush(log_file);
                }
            }
        }
    }
}

```

```
    } else {
        printf("[WARN] No se pudieron parsear los 4 campos. Cadena: '%s'\n", buffer);
    }

    printf("[INFO] Total: %d paquetes\n\n", count);
}

static int status = 0;
if (++status >= 10) {
    printf("[STATUS] Escuchando... (%d recibidos)\n", count);
    status = 0;
}
}

printf("\n[INFO] Cerrando...\n");
lora_set_mode(MODE_SLEEP);
close(spi_fd);
if (log_file) fclose(log_file);
printf("[OK] Cerrado\n");

return 0;
}
```