Trabajo Práctico

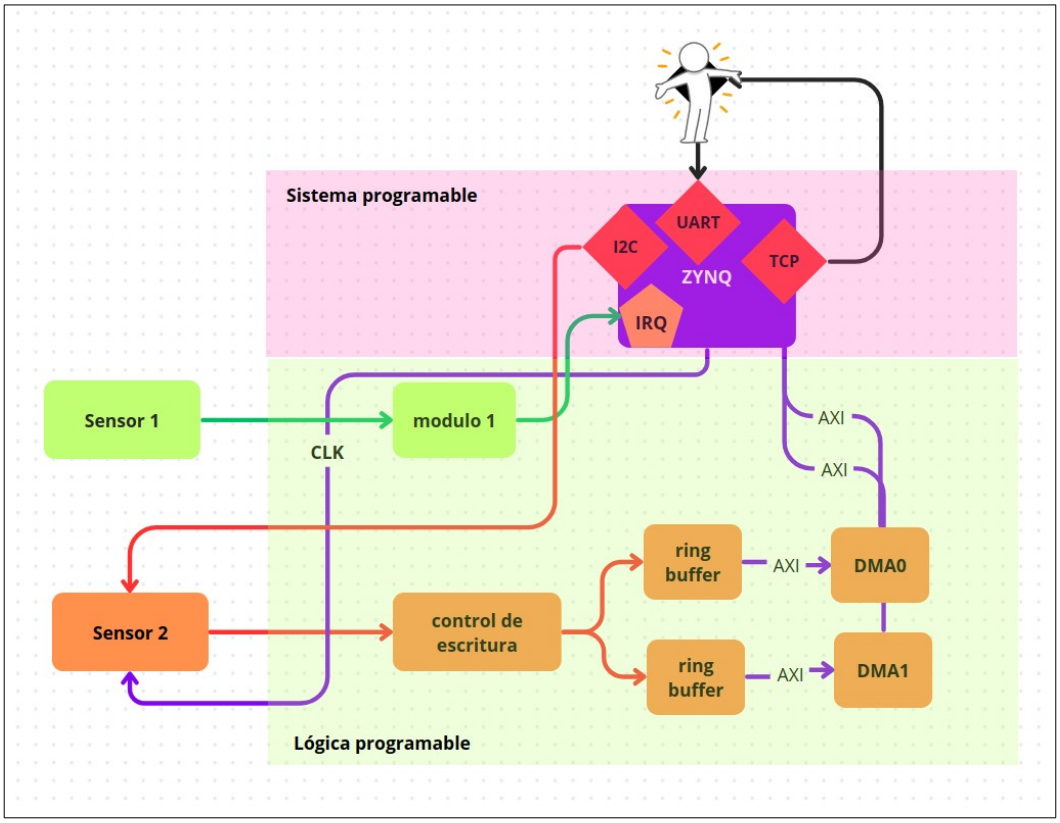
Título: Implementación de una aplicación de tiempo real en un sistema embebido a elección

Objetivo:
Diseñar e implementar una aplicación en sistemas embebidos sobre una plataforma de recursos limitados, en relación con los contenidos de la asignatura.

Alcances:

1. **Definir claramente el objetivo y el contexto de uso de la aplicación de tiempo real seleccionada.**

Adquirir, procesar y transmitir datos de múltiples sensores en tiempo real, garantizando baja latencia, sincronización precisa y comunicación confiable con un usuario externo a través de UART/TCP.



Esto implica (disclaimer):

- Captura continua de señales de Sensor 1 (I2C) y Sensor 2 (vía PL+DMA).
- Procesamiento inmediato de la información para obtener resultados útiles (filtros, cálculos, correlaciones).
- Transmisión en tiempo real hacia el usuario o un sistema supervisor (por TCP/IP o UART).
- Supervisión del sistema (estado, fallos, watchdog, configuración dinámica).

Contexto de uso

El sistema se aplicará en un entorno embebido de tiempo real, con las siguientes condiciones:

1. Restricciones temporales (tiempo real):

- Los datos deben procesarse y estar disponibles dentro de un intervalo máximo definido (ej. <10 ms desde la adquisición hasta la transmisión).
- Se requiere respuesta inmediata a eventos críticos (ej. interrupción de DMA).

2. Restricciones de recursos:
   - CPU limitada (ARM Cortex-A9 single-core del Zynq).
   - Memoria limitada 512 MB.
   - Uso del PL (FPGA) para offload de procesamiento intensivo y transferencia rápida mediante DMA.
3. Entorno operativo:
   - Puede ser un sistema de adquisición industrial, biomédico o de IoT avanzado.
   - Conectado a sensores heterogéneos que entregan datos en diferentes protocolos (I2C y entradas rápidas al PL).
   - Necesita comunicación confiable con un usuario (interfaz humana o servidor remoto).
4. Robustez y tolerancia a fallos:
   - Debe manejar errores de comunicación (reintentos en I2C, pérdida de paquetes TCP).
   - Supervisión mediante watchdog y tareas de monitoreo.

2. **Identificar los posibles modos de implementación y establecer los requerimientos funcionales y no funcionales asociados.**

Modo de implementación	Descripción	Ventajas	Desventajas	Requerimientos funcionales	Requerimientos no funcionales
PS (Software puro en FreeRTOS)	Todo se ejecuta en el Cortex-A9: adquisición, procesamiento, comunicación y supervisión.	- Fácil de desarrollar y depurar.- Menor complejidad de integración.- No requiere HDL en el PL.	- Alta carga en CPU.- Riesgo de no cumplir tiempos de respuesta.- Latencias mayores en DMA y procesamiento.	- Lectura I2C (Sensor 1).- Lectura DMA gestionada por software.- Procesamiento básico.- Comunicación UART/TCP.- Supervisión con watchdog.	- Latencia total ≤10 ms difícil de garantizar.- Uso intensivo de CPU.- Memoria del PS limitada.
Híbrido (PS + PL)	El PL maneja DMA, buffers y opcionalmente procesamiento. El PS con FreeRTOS gestiona control, sincronización y comunicación.	- Buen balance entre rendimiento y complejidad.- Reduce carga en CPU.- PL acelera tareas críticas.- Más escalable.	- Mayor complejidad de diseño.- Requiere integración PS–PL (AXI, drivers).	- Lectura I2C (Sensor 1).- Lectura Sensor 2 vía DMA + buffers en PL.- Procesamiento en PS + PL.- Comunicación UART/TCP.- Supervisión con watchdog.	- Latencia total ≤10 ms garantizable.- Uso eficiente de DMA y ring buffers.- Requiere manejo de interrupciones.- Modularidad de software.
PL centrado (hardware acelerado)	El PL hace casi todo el procesamiento y entrega resultados al PS. El PS solo comunica y supervisa.	- Máximo rendimiento y baja latencia.- CPU casi libre.- Alta escalabilidad.	- Desarrollo complejo en Verilog o HLS.- Dificultad de depuración.- Poco flexible a cambios en algoritmos.	- Implementar adquisición + procesamiento en PL.- PS solo UART/TCP + supervisión.	- Latencia mínima.- Alto uso de recursos en FPGA.- Menor flexibilidad en actualización de funciones.

Analizando las ventajas y desventajas anteriormente planeadas con respecto a los modos de implementación, se considera que la mayor ventaja se obtiene de un sistema híbrido en el que se desacople el sistema de bajo nivel de hardware de la de alto nivel en lenguaje C.

Los requerimientos funcionales definen lo que el sistema debe hacer para cumplir con su propósito (adquisición, procesamiento y comunicación de datos de sensores en tiempo real).

ID	Descripción	Componentes	Notas de implementación
FR1	Adquirir datos del Sensor 1 vía I2C a tasas configurables (p. ej., 10–50 ms).	Sensor 1, I2C, Módulo 1	Usar tarea periódica en FreeRTOS o un IP en PL; validar integridad de datos.
FR2	Adquirir datos del Sensor 2 vía interfaz GPIO/ADC.	Sensor 2	Tarea basada en interrupciones en FreeRTOS o muestreador hardware en PL.
FR3	Procesar datos de sensores (p. ej., filtrado, fusión en el bloque “Procesamiento”).	Procesamiento, Delay	Implementar como tarea en FreeRTOS o como IP en HLS; salida a ring buffers.
FR4	Gestionar el almacenamiento temporal de datos y la transferencia usando canales DMA.	Ring Buffers, DMA0/DMA1, AXI	Tarea en FreeRTOS configura el DMA; el PL acelera las transferencias.
FR5	Sincronizar temporización y relojes del sistema.	Alimentación Clock, CLK	Usar temporizador TTC en FreeRTOS o contadores en PL para mayor precisión.
FR6	Manejar interrupciones de eventos (p. ej., datos listos, errores).	IRQ	ISR notifica a tareas de FreeRTOS mediante semáforos.
FR7	Comunicar datos procesados vía UART, I2C (externo) y TCP.	UART, I2C, TCP	Tarea de baja prioridad en FreeRTOS con pila lwIP; soportar comandos de reconfiguración.
FR8	Registrar estado del sistema y errores.	Sistema general	Tarea de monitoreo envía salidas vía UART.

Por otro lado, los requerimientos no funcionales establecen cómo debe comportarse la aplicación una vez implementada, garantizando que el sistema híbrido propuesto cumpla con criterios de rendimiento, confiabilidad, escalabilidad, eficiencia energética, seguridad y portabilidad, de manera que la solución no solo satisfaga las funciones esperadas sino que también lo haga de forma robusta y eficiente en un entorno de tiempo real.

ID	Categoría	Descripción	Métricas / Objetivos	Notas de implementación
NFR1	Rendimiento (Tiempo)	Asegurar tiempo real estricto en adquisición y procesamiento de sensores (p. ej., sin incumplir plazos).	Latencia <10 ms en rutas críticas; análisis de WCET.	Priorizar tareas en FreeRTOS; usar PL para mayor determinismo.
NFR2	Rendimiento (Throughput)	Manejar tasas de datos de los sensores (p. ej., muestreo a 1 kHz).	Procesar 100+ muestras/seg sin desbordes.	Uso de DMA y colas en implementación híbrida.
NFR3	Confiabilidad	El sistema debe recuperarse de errores (p. ej., fallo de sensor) sin colapsar.	Disponibilidad >99%; diseño tolerante a fallos.	Temporizadores watchdog en FreeRTOS; tareas redundantes en multicore.
NFR4	Escalabilidad	Soportar la adición de más sensores o nuevas funcionalidades.	Tareas/IP modulares; mínimos cambios de código.	Modularidad del RTOS; reconfiguración del PL.
NFR5	Eficiencia energética	Operar dentro del presupuesto de consumo del Zynq para uso embebido.	<5 W de consumo.	Modos de bajo consumo en FreeRTOS; gating en PL.
NFR6	Usabilidad / Mantenibilidad	Fácil de depurar y actualizar.	Uso de herramientas estándar (Vitis, Tracealyzer).	Tareas estructuradas; documentación clara.
NFR7	Seguridad	Proteger contra accesos no autorizados (p. ej., vía TCP).	Autenticación básica en comunicaciones.	Opcional en prototipo; ampliar con métodos avanzados.
NFR8	Portabilidad	Ejecutarse en distintos modelos Zynq (p. ej., serie 7000).	Compatible con BSPs de Xilinx.	Uso de puertos estándar de FreeRTOS.

3. **Implementar la solución en la plataforma embebida correspondiente.**
4. **Verificar y validar el correcto funcionamiento de la aplicación.**

Para iniciar la implementación, se requiere la utilización de módulos genéricos que simulen la lectura de los controladores DMA del sistema. Con este propósito, se desarrollaron dos módulos “dummy”: snow.v, que realiza un conteo ascendente, y white.v, encargado de un conteo descendente. Ambos módulos fueron codificados en Verilog y diseñados para operar bajo el protocolo de comunicación AXI Stream, elegido por la simplicidad de integración con el sistema Diligent.

Posteriormente, estos módulos se integraron con el System on Chip (SoC) mediante el diagrama en bloques de la herramienta de diseño Xilinx, lo que permite la conexión y el acceso al procesador Cortex-A9. La Figura 1 ilustra el diagrama en bloques del sistema completo de adquisición de datos.

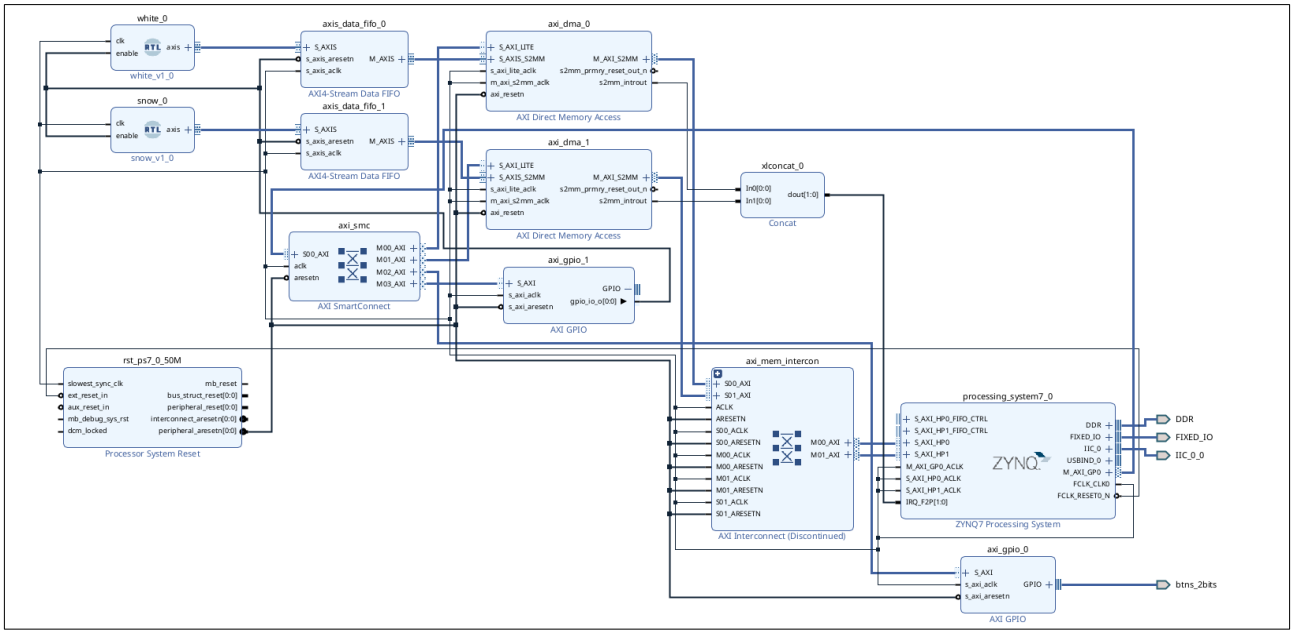


Figura 1. Diagrama en bloques del sistema de adquisisción de datos de dos sensores.

Además, en el Cortex-A9 se activaron y configuraron los protocolos TCP, I2C y UART para permitir la comunicación con el usuario y la transferencia de información entre los distintos módulos del sistema. En particular, I2C se implementó para la configuración de un expansor I2C, encargado de paralelizar los datos provenientes de los sensores.

La implementación completa del sistema indica que el mayor consumo de recursos se concentra en las Look-Up Tables (LUTs) de la FPGA, mientras que el Cortex-A9 representa el mayor consumo de procesamiento, tal como se había previsto en la primera fase del proyecto. Esta arquitectura permite centralizar la adquisición de datos en el Cortex, maximizando el uso del sistema en tiempo real provisto por la plataforma Xilinx.

La Figura 2 presenta el consumo total de memoria y potencia del sistema. Cabe destacar que los 1,254 W medidos constituyen un consumo relativamente bajo para este tipo de plataformas y son consistentes con el comportamiento esperado de la Cora Z7.

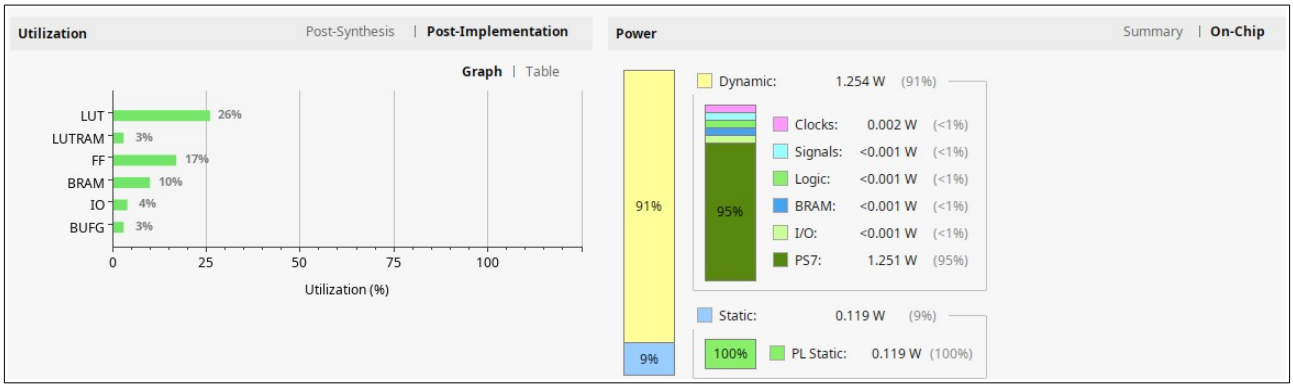


Figura 2. Consumos de memoria y potencia en la plataforma de evaluación Cora Z7.

El primer ensayo se centró en verificar la comunicación del servidor TCP implementado en el procesador. Para ello, se ejecutaron comandos básicos, como `ping 192.168.1.10`, cuyo resultado se muestra en la Figura 3. Esta prueba permitió confirmar que la conectividad entre la plataforma embebida y la PC cliente estaba correctamente establecida y operativa.

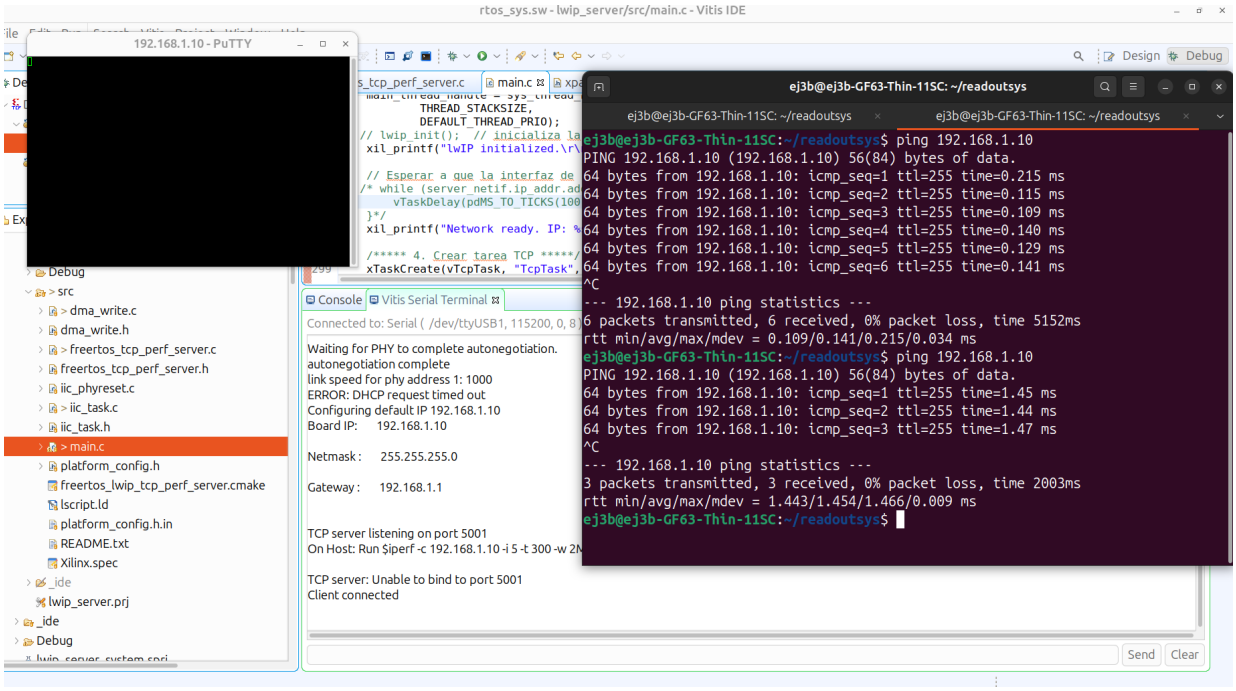


Figura 3. Comando PING utilizado apara testear la comunicación del sistema con la plataforma escogida.

Posteriormente, se configuró el primer sensor mediante el protocolo I2C. Una vez finalizada la configuración, se notificó por TCP que el sensor estaba listo mediante el mensaje “Configuration sensor 1 successful”, enviado por consola. La Figura 4 ilustra tanto la configuración del sensor vía I2C como la comunicación TCP, demostrando la correcta sincronización entre ambos protocolos.

La validación de la comunicación I2C se realizó mediante la función de callback del sistema. Debido a restricciones de tiempo, no se utilizó un Arduino UNO como esclavo; en su lugar, se dejó el pin desconectado y se monitorizó el estado del bus mediante UART. La comunicación I2C se

realizó a baja velocidad (100 kHz), y la Figura 4 muestra el estado del bus, incluyendo los NACKs generados conforme al protocolo de comunicación serial, lo que permitió verificar la integridad del control de transmisión.

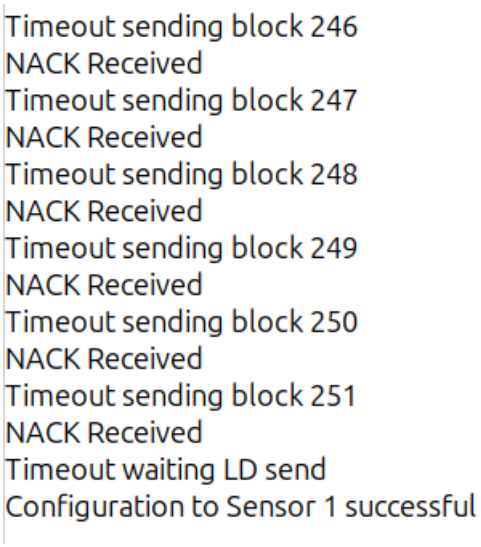


Figura 4. Configuración del sensor 1 por medio de I2C. Los mensajes resultan esperados para la comunicaión escogida.

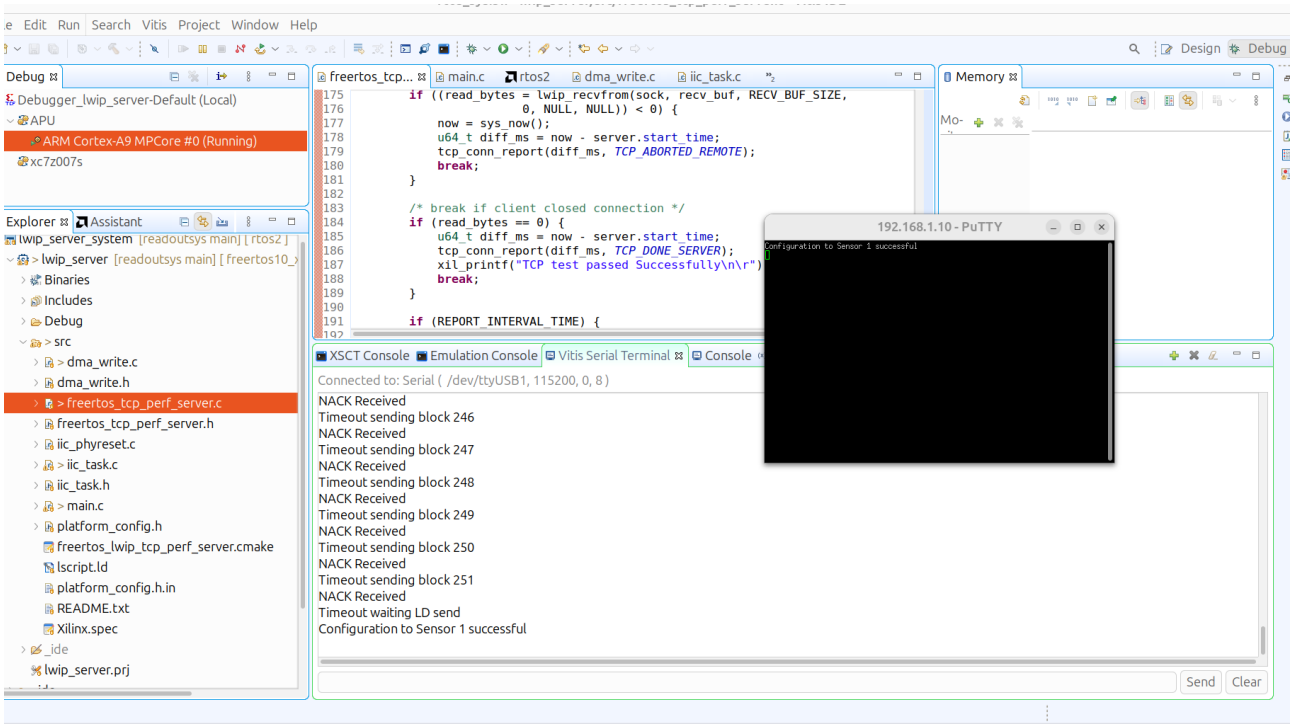


Figura 5. Configuración I2C para un sensor no conectado. Se observan los NACK de la comunicación en I2C y simultáneamente el mensaje en TCP indicando la configuración del sensor lo que prueba la sincronizacion entre ambos protocolos en FreeRTOS.

Posteriormente, se incorporó la lectura de los DMA para verificar la transferencia de datos desde el PL hacia el PS. Para ello se utilizaron los módulos de prueba snow.v y white.v, que generan

secuencias de conteo ascendente y descendente, respectivamente, dentro de un rango de 1024 unidades de datos (no en bits, sino en unidades discretas de información).

La validación de la transferencia se realizó mediante la comunicación UART, confirmando que los datos transmitidos desde el PL llegaban correctamente al PS. Los resultados obtenidos se muestran en la Figura 6, donde se observa la integridad y consistencia de las secuencias generadas por ambos módulos.

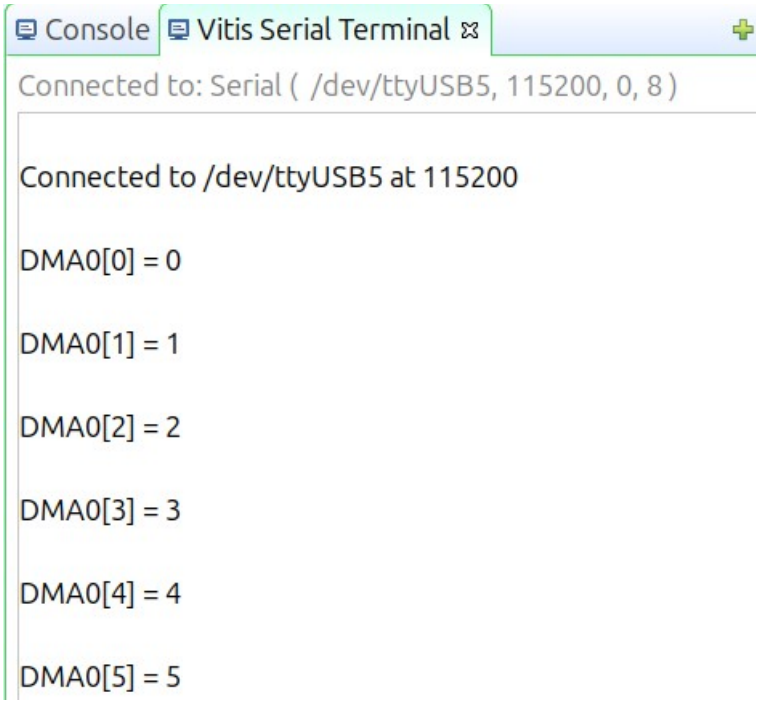


Figura 6. Lectura del DMA para el módulo de prueba de white.v (números ascendants).

5. Elaborar conclusiones a partir de la experiencia y los resultados obtenidos.

A. La arquitectura híbrida PS + PL proporciona un balance óptimo entre rendimiento y complejidad de diseño, asegurando baja latencia y eficiente manejo de DMA.

B. La implementación de módulos de prueba en Verilog permitió verificar la transferencia de datos y la sincronización entre los protocolos I2C, UART y TCP.

C. Las pruebas confirmaron que el sistema cumple con los requerimientos funcionales (adquisición, procesamiento y comunicación de sensores en tiempo real) y no funcionales (latencia <10 ms, eficiencia energética, confiabilidad y escalabilidad).

D. La experiencia demuestra que desacoplar las tareas críticas en el PL y mantener control y supervisión en el PS es la estrategia más efectiva para sistemas embebidos de tiempo real.

E. Los resultados obtenidos (consumo de 1.254 W, correcta sincronización de protocolos y transferencia DMA confiable) muestran que la solución es robusta, escalable y eficiente, y puede servir como base para futuras ampliaciones del sistema.