

Práctica 1: Introducción al entorno

Arquitectura de Sistemas Integrados

Alejandro Solá
asola01@ucm.es

Pablo Suárez
pasuar03@ucm.es

Febrero 2023



Índice

Índice		2.3. Entregable 3	7
1. Introducción	3	3. Sesión 2	9
		3.1. Entregable 1	10
2. Sesión 1	3	3.2. Entregable 2	10
2.1. Entregable 1	3		
2.2. Entregable 2	7	4. Conclusión	11

1. Introducción

En esta primera práctica, hemos caracterizado varios de los aspectos más relevantes de un sistema embebido:

- Potencia
- Bloques funcionales
- Funcionalidad de sus módulos
- Interacción de un **SoC** real con el entorno de trabajo.

Producto de haber estudiado los módulos que componen el sistema, así como su arquitectura, hemos ido familiarizándonos con el entorno de trabajo a lo largo de las dos sesiones correspondientes a esta primera práctica.

2. Sesión 1

En esta primera sesión, hemos cargado el `project1` en Vivado, una vez cargado el proyecto hemos estudiado los módulos del diseño implementado, simplificado el diagrama de bloques y caracterizado la potencia de cada etapa, teniendo en cuenta la arquitectura del **SoC**.

2.1. Entregable 1

A continuación se identifican las diferentes partes del **SoC**. El esquemático general tiene la siguiente forma:

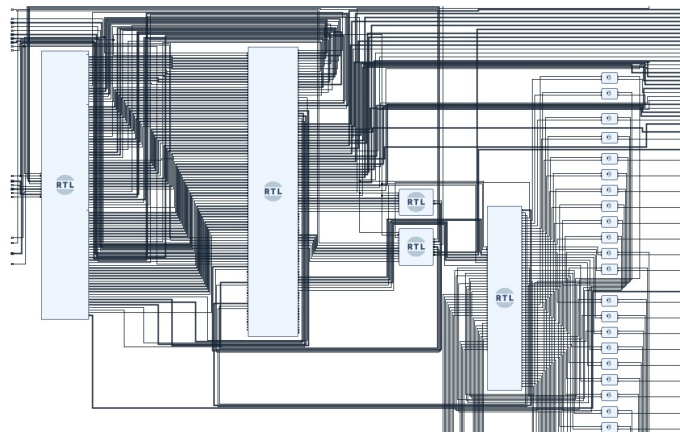


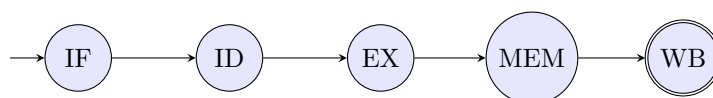
Figura 1: Esquemático del **SoC** implementado en Vivado

Como se puede ver, se trata de un sistema complejo a nivel de conexiones, pero podemos distinguir claramente cada uno de los módulos que componen el **SoC**. No está de más mencionar que las conexiones más gruesas son de mayor ancho de banda ya que son capaces de transportar un mayor número de bits, y estas se corresponden con buses.

Analizando cada uno de los bloques se tiene lo siguiente ¹:

- Módulo **SWERV**: este módulo se corresponde con la **CPU** del sistema.

Como sabemos por teoría, la capa de aplicaciones software dependen de este, así como de la **GPU** este módulo se afronta desde la capa de *Aplicaciones software*. Anteriormente en Estructura de Computadores, vimos la arquitectura **MIPS** con las 5 etapas clásicas:



¹Todos los módulos que se explican a continuación cuentan con dos pines comunes entre todos ellos: el reloj/*clock* y el pin destinado al reset, por ello, no son mencionados en cada uno de los módulos, puesto que no son pines propios de un módulo en específico

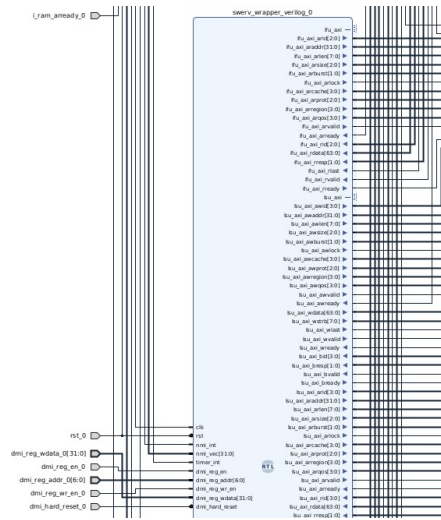


Figura 2: Módulo `swerv_wrapper_verilog_0`

- Conexiones de **entrada**:

- **nmi**: hace referencia a *non-maskable interrupt*. El uso de estos pines está relacionado con las interrupciones hardware que las técnicas estándar de enmascaramiento de interrupciones del sistema no pueden ignorar. Los **nmis** se utilizan a menudo cuando el tiempo de respuesta es crítico o cuando una interrupción nunca debe ser desactivada durante el funcionamiento del chip. Dentro de los usos habituales se incluyen la notificación de errores de hardware no recuperables, la depuración y creación de perfiles del sistema así como los reinicios del sistema.
- **timer.int**: hace referencia a *timer interrupt* y son similares a los pines de interrupciones externas, pero en lugar de activarse por un evento externo, lo hacen por un temporizador. Se llaman así porque interrumpen el hilo de ejecución una vez finalizada la instrucción actual y ejecutan su código, volviendo a la siguiente instrucción desde donde se quedó.
- **dmi**: estas siglas hacen referencia a *Debug Module Interface*, que es un módulo de interfaz ligado a la depuración. Los **dmi** son pines de conexión externa, tal y como podemos ver en el diseño implementado en Vivado.

- Conexiones de **salida**:

- **ifu**: estos pines se corresponden con la unidad de **instruction fetch** i.e: **IF**.
- **lsu**: estos pines se corresponden con la unidad de **load store**.
- **sb** estos pines se corresponden con el **store byte module**.

Hemos estudiado el **RISC-V** una arquitectura prima hermana de la anterior (2.1), luego estudiamos el algoritmo de planificación dinámica llamado Tomasulo. Finalmente hemos estudiado la arquitectura del **SoC** del laboratorio que será la que implementaremos en las prácticas. La siguiente figura muestra de manera clara y detallada la arquitectura del pipeline de dicho sistema:

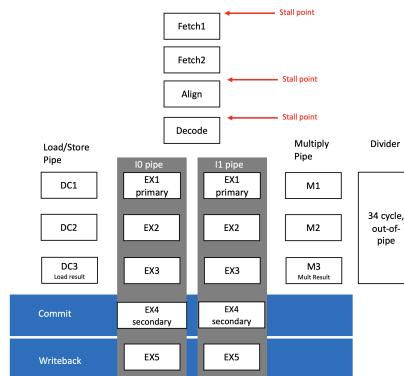


Figura 3: Pipeline del **SoC** del laboratorio

Módulo `int_con`: este módulo se encarga de las interrupciones de entrada/salida. También tiene la función de interconexión, este conexionado lo vemos presente entre la **CPU** (i.e: el módulo `swerv`) y los módulos: `boot-rom-wrapper`, `syscon_wrapper`, `GPIO` y `bidirec's`.



- Pines de **entrada**:
 - **i_ifu**: estos pines se corresponden con la unidad de **instruction fetch** i.e: **IF**
 - **i_lsu**: estos pines se corresponden con la unidad de **load store**.
 - **i_ram**: estos pines están destinados al *Random Access Module* i.e: **RAM**. Todos estos pines son direccionados como pines de salida, ya que en este **SoC** la mayor parte de la RAM se encuentra fuera del mismo. Decimos la mayor parte, porque también hay algo de RAM en la CPU (**swerv module**)
 - **i_sb**: estos pines se corresponden con el **store byte module**.
 - **wb**: estos pines se corresponden con los buses **Wishbone**, este tipo de buses lógicos permite la comunicación entre módulos del sistema integrado, por ello en cualquiera de los módulos donde están contemplados, estos pines conectan con otro módulo interno del **SoC**.
- Pines de **salida**:
 - **o_ifu**: estos pines se corresponden con la unidad de **instruction fetch** i.e: **IF**
 - **o_lsu**: estos pines se corresponden con la unidad de **load store**.
 - **o_ram**: estos pines están destinados al *Random Access Module* i.e: **RAM**. Todos estos pines son direccionados como pines de salida, ya que en este **SoC** la mayor parte de la RAM se encuentra fuera del mismo. Decimos la mayor parte, porque también hay algo de RAM en la CPU (**swerv module**).
 - **o_sb**: estos pines se corresponden con el **store byte module**.
 - **wb**: estos pines se corresponden con los buses **Wishbone**, este tipo de buses lógicos permite la comunicación entre módulos del sistema integrado, por ello en cualquiera de los módulos donde están contemplados, estos pines conectan con otro módulo interno del **SoC**.

Este es un módulo de memoria, y forma parte del apartado de los periféricos junto al GPIO y el `syscon_wrapper` (ver figura 5a). Este módulo consta de pines destinados al `clock`, `reset` y al bus *wishbone* a la entrada y a la salida.

La función de dicho módulo, como bien indican sus siglas es la de controlar el sistema (**system_controller**). Como bien se ha comentado antes, este módulo es parte de la sección de los periféricos del sistema

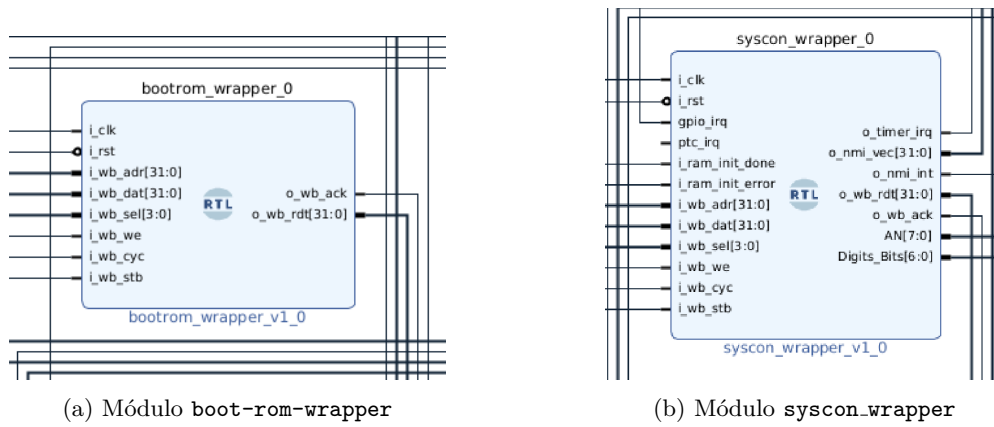


Figura 5: Módulos boot-rom-wrapper y boot-rom-wrapper

(ver figura 5b). Al igual que en el anterior módulo, en este podemos encontrar la misma familia de pines a la entrada con la diferencia de que en este hay tres pines dedicados a la RAM, al módulo GPIO (imaginamos que servirá para controlar los leds y los switches) y un pin denominado *ptc_irq*². A la salida tenemos un pin para un *timer*, 2 pines de tipo *nmi*, 2 pines *wb*, un pin de salida de tipo AN y otro de salida denominado *Digits_Bits*.

■ Módulo GPIO

El módulo GPIO involucra los pines disponibles del sistema, y por ende forma parte del apartado de los periféricos del sistema tal y como se ha mencionado antes (ver figura 6).

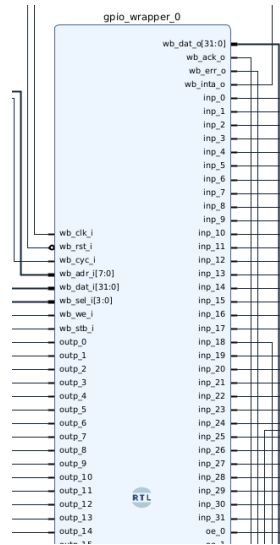


Figura 6: Módulo GPIO

Este módulo consta de dos familias de pines a la entrada, ya que hay pines de tipo *wb* y pines de salida, que van directos a los módulos *bidirecs* los 16 de ellos van a los bidireccionales destinados a los switches de la placa y los otros 16 van a los leds.

■ Módulos *bidirec*'s

Estos módulos que se encuentran repartidos en la parte derecha del esquemático y están posicionados en vertical cubriendo de arriba a abajo la sección izquierda del esquemático. Su función está relacionada con el manejo de los leds así como de los switches (ver figura 14).

- 16 módulos *bidirec* dedicados a los **switches**.
- 16 módulos *bidirec* dedicados a los **leds**.

²No estamos seguros de su funcionalidad, pero creemos que puede estar relacionado con los termistores, ya que las siglas *PTC* hacen referencia a: "Coeficiente de temperatura positivo". Los termistores PTC son resistencias con un coeficiente de temperatura positivo, lo que significa que la resistencia aumenta al aumentar la temperatura.

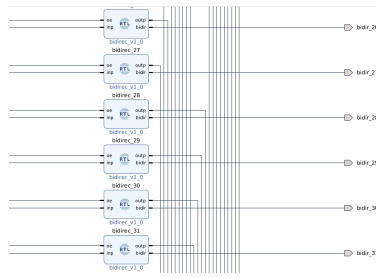


Figura 7: Módulos bidirecs

2.2. Entregable 2

Estudiando las imágenes tomadas durante la primera sesión del laboratorio, podemos inferir el siguiente diagrama de bloques simplificado del **SoC**.

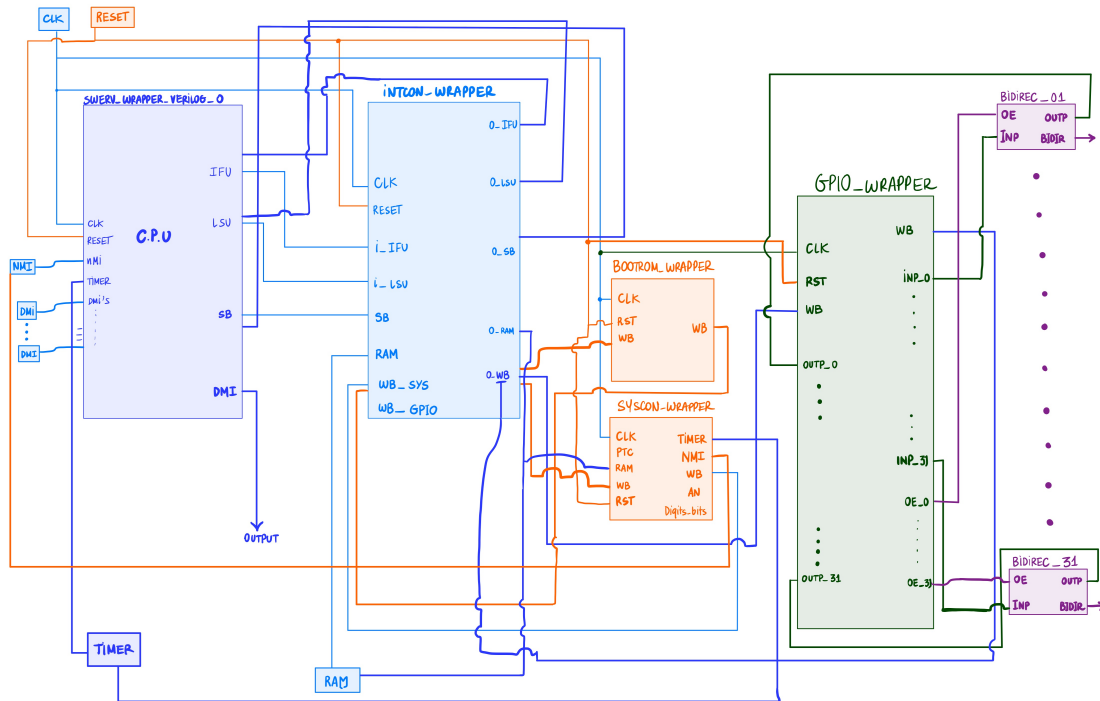


Figura 8: Esquemático simplificado

2.3. Entregable 3

Habiendo identificado las diferentes partes del **SoC**, podemos hacer un estudio sobre el consumo del **System on Chip** en términos de potencia. Además, sabiendo el espacio que ocupa cada etapa del sistema, podemos establecer una relación entre la potencia consumida, y el área cubierta por cada etapa.

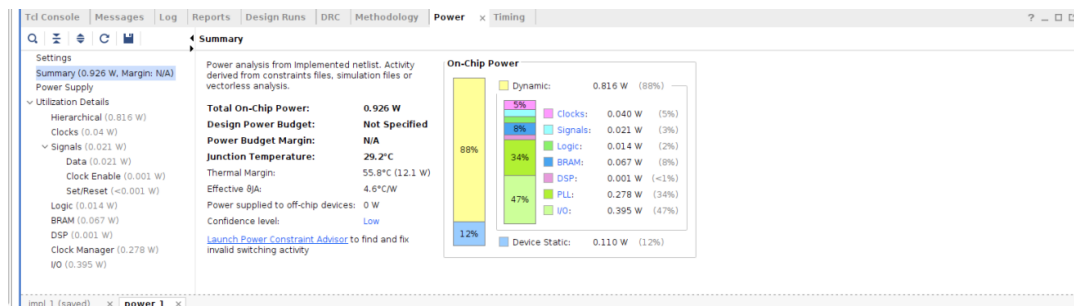


Figura 9: Summary de la potencia consumida

Este consumo de potencia, lo podemos relacionar directamente con el área ocupada por cada etapa, esto nos dará una idea de lo acertado que es el consumo de potencia en cada etapa respecto al área que ocupa cada una de ellas. Aunque normalmente se cumple que: $Potencia \propto \text{Área}$, pero a continuación veremos que no se trata de la norma general. Sabiendo que la potencia estática está ligada a la corriente de fuga de los transistores, cuanto más pequeño es el transistor, mayor corriente de fuga habrá. Mientras que la potencia dinámica hace referencia a la f_{clock} ya que: $(\uparrow f_{clock}) \rightarrow (\uparrow P[mW]) \rightarrow (\uparrow T[^\circ C])$

Como podemos observar en la figura 9, la parte dinámica del **SoC** consume casi el 90 % de la potencia del mismo, mientras que el resto es consumida por la potencia estática. Si nos fijamos en cada sección de la parte dinámica, podemos ver como la parte de entrada y salida es la que más va a consumir. Podemos ver también como la parte de BRAM(MEM) y de Logic(EX) consumen relativamente poca potencia en comparación con el resto. Relacionando estos datos con las superficies que ocupan en el **SoC** que mostramos en las figuras siguientes, vemos como la fase de memoria va a consumir poca potencia en comparación con la de ejecución, la cual ocupa mucha más superficie.

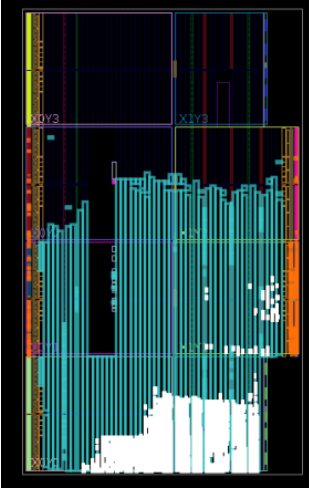


Figura 10: IF

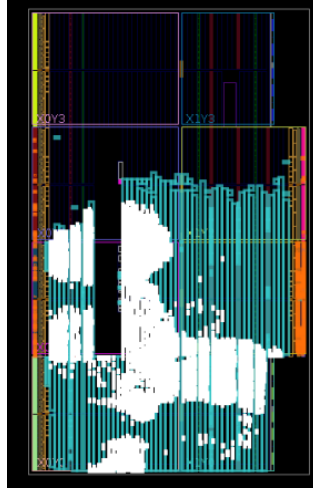


Figura 11: ID

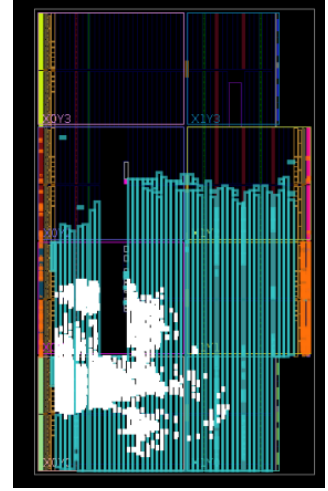
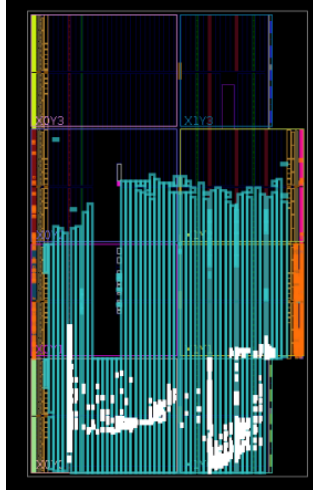
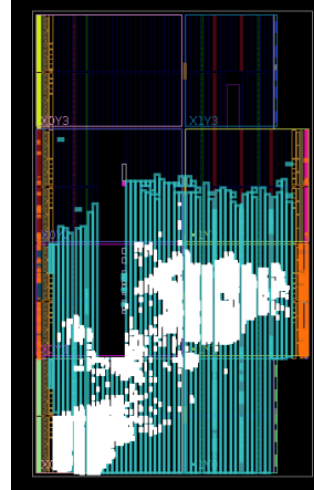


Figura 12: EX



(a) MEM



(b) WB

Figura 13: Módulos boot-rom-wrapper y boot-rom-wrapper

Podemos observar claramente como el **ID** es la etapa que más espacio ocupa, esto se debe principalmente a su responsabilidad a la hora de encontrar conflictos y problemas e.g: dependencias entre instrucciones. En lo que a solapamientos entre etapas se refiere, podemos ver uno entre las etapas de **ID** y **WB** ya que en el **instruction decode** se lee del registro y en la etapa del **write back** se escribe en el mismo. Es notable también el solapamiento que hay entre el **IF** con el **EX** debido a la actualización del contador de programa (PC) por parte del **IF**. Cabe destacar que el área ocupada por la etapa de **EX** se corresponde con todas las

En la siguiente imagen podemos ver la potencia consumida por etapa, esto nos ayudará a comprender mejor la eficiencia del **SoC**.



- La etapa **IF** consume; 7 mW, esto se traduce en un 1 % de la potencia total consumida por el módulo **swerv**.
- La etapa **ID** consume; 8 mW, esto se traduce en un 1 % de la potencia total consumida por el módulo **swerv**.
- La etapa **EX** consume; 14 mW, esta cifra resulta de sumar el apartado de **exu** con el del **lsu**, ya que la unidad del load/store es una unidad especial de ejecución destinada a la ejecución de todas las instrucciones de LD y STR, esto se traduce en un 2 % de la potencia total consumida por el módulo **swerv**.
- La etapa **MEM** consume; 44 mW, esto se traduce en un 5 % de la potencia total consumida por el módulo **swerv_eh1.2**. Pero observando más las cifras vemos que dentro del módulo **swerv** hay un apartado llamado **dma_ctrl** esto hace referencia a '*Direct Memory Access*', esto permite al dispositivo transferir los datos directamente a/desde la memoria sin ninguna interferencia de la CPU, por lo tanto: $44 + 1 = 45 \text{ mW} \sim 1 \%$ de la potencia total consumida por el módulo **swerv_eh1.2**.
- La etapa **WB** (**dbg** y **pic_ctrl_inst**) consume; 2 mW, esto se traduce en menos de un 2 % de la potencia total consumida por el módulo **swerv**.

3. Sesión 2

En la segunda sesión de la práctica, el foco se ha puesto en la integración del sistema físico con el entorno de desarrollo (Vivado). Con un `bitstream` previamente generado, hemos cargado un archivo de tipo `.elf` en el μ procesador, este archivo contiene código ensamblador que alberga la imagen más el **OS** ³. Además, también hemos llevado a cabo la carga del RTOS y la compilación de códigos C y ensamblador.

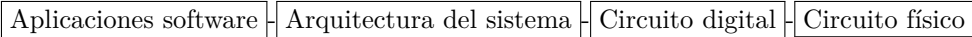
³En los entregables correspondientes a la segunda sesión del laboratorio esto se explica en mayor profundidad

3.1. Entregable 1

Cabe destacar que en este punto, aun no hemos introducido ningún sistema operativo. Las capas contenidas en la primera aplicación son las siguientes:

- **bitstream**: Forma parte de la arquitectura del sistema y de la del circuito digital, porque creábamos la arquitectura en Vivado, pero esa arquitectura es producto de un circuito digital.
- El **Blink.s** forma parte de la capa de aplicación, porque es un código que aplicamos al sistema empujado.
- Placa NEXYS 4 A7 Baremetal: hace referencia al circuito físico.

Por lo tanto, en este apartado solo tendríamos las siguientes capas:



3.2. Entregable 2

Para realizar el volcado de la placa hemos tenido que conectarnos por **telnet localhost 4444**, la conexión abierta es la siguiente:

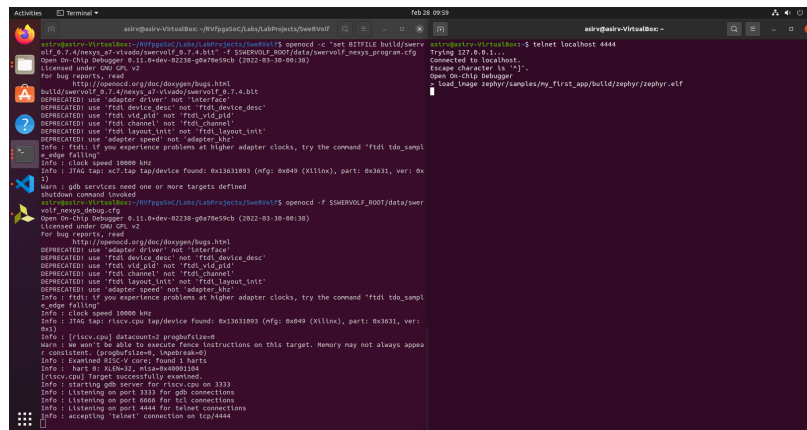


Figura 15: Terminal abierto en **telnet** escuchando en el puerto 4444

Instantes después introducimos el siguiente comando en la terminal;

```
load_image zephyr/samples/my_first_app/build/zephyr/zephyr.elf
```

Como sabemos por teoría, el archivo **zephyr.elf** es la imagen básica, pero hemos visto en clase que se puede modificar, añadiendo y/o quitando librerías, o incluso eliminando interfaces, con el fin de incrementar la eficiencia. Hay dos formas de deshabilitar módulos del **SoC**: mediante hardware (des · soldar el módulo) o firmware (quitar los drivers asociados a dicho módulo)

Tras haber programado la placa le indicamos a la placa donde tiene que cargar el **PC**. Luego se activa el RTOS y la aplicación de muestra con el comando **resume**, esta aplicación ejecuta de manera simultánea el parpadeo de un led y el envío de un mensaje por puerto serie, para visualizar esto hemos tenido que abrir un terminal con Putty. Véase a continuación la ejecución de este último paso en la siguiente figura:

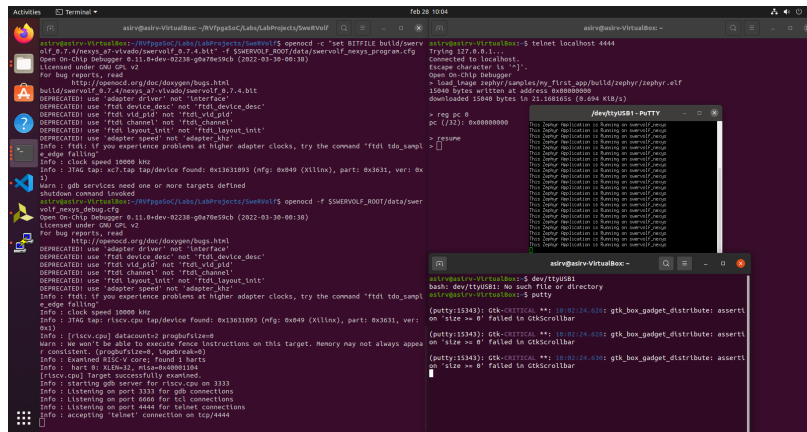
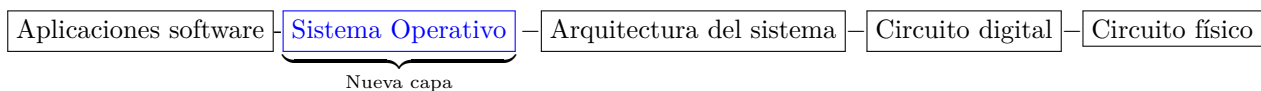


Figura 16: Putty corriendo la aplicación

- Hemos utilizado 2 ejecutables que intervienen en diferentes capas:
 - `.elf`: interviene en la capa de sistema operativo y en la capa de aplicación
 - `.cfg`: permite comunicación con telnet. Interviene en la capa de circuito digital y en la capa de arquitectura del sistema.
- Los puertos que nos encontramos abiertos son los siguientes:
 - 3333: puerto habilitado para el protocolo gdb que se utiliza para hacer la depuración del programa.
 - 4444: puerto habilitado para el protocolo telnet, para conectarnos a una máquina remotamente.
 - 6666: puerto habilitado para el TCL, es un protocolo que es implementado para el desarrollo y pruebas de aplicaciones script.
 - Puerto serie: habilitado en la placa a través de conexión USB para comunicarnos con ella y volcar el programa.

En este último punto, al haber cargado el sistema operativo en el sistema empotrado, hemos introducido la capa correspondiente al sistema operativo entre la capa del software y la capa de la aplicación. Sobre el **SoC** se encuentra el sistema operativo, y sobre el sistema operativo se encuentra el programa compilado. Cabe destacar que a la hora de cargar el workspace en VScode había numerosos archivos `.json`, estos archivos están ligados a la cross-compilación, concepto que hemos estudiado en clase.

Dicho lo anterior, podemos esbozar un pequeño esquema que recopila las capas que intervienen en esta segunda aplicación (teniendo en cuenta las anteriores)



4. Conclusión

En esta primera práctica nos hemos familiarizado con el entorno de trabajo, así como con el sistema empotrado. Ha sido una práctica interesante en la que hemos podido enlazar y poner en práctica todo aquello estudiado en el primer tema de teoría.