



UNIVERSIDAD DE ZARAGOZA

SISTEMAS EMPOTRADOS 2

Práctica 4

Medida de latencia de interrupción en Linux

AUTOR:

Adrián Martín Marcos 756524

Zaragoza, España

Curso 2020–2021

Índice

| | |
|--|----------|
| 1. Configuración del kernel | 1 |
| 2. Primer apartado | 2 |
| 2.1. Compilación e instalación del módulo | 2 |
| 2.2. Preparación de las conexiones al osciloscopio | 2 |
| 2.3. Medición de la latencia de interrupción | 3 |
| 2.4. Solución al problema de desmontaje del módulo | 4 |
| 3. Segundo apartado | 5 |
| 3.1. Añadir segundo canal | 5 |
| 3.2. Modificaciones del código fuente | 5 |
| 3.3. Preparación de las conexiones al osciloscopio | 6 |
| 3.4. Medición de la latencia de interrupción | 6 |
| Referencias | 8 |



1. Configuración del kernel

En este primer apartado se va a comentar cuál es la configuración del kernel de la distribución sobre la que se ha realizado la práctica.

Los fuentes de Linux se han obtenido del fichero *kernel-rt.xz*, que se encuentra en el directorio de prácticas de la asignatura, en el servidor Hendrix [2]. Sobre ellos ya se ha aplicado el parche tiempo real, por lo que al ejecutar *uname -r* en la distribución, el release que aparece es *3.8.13-rt14*.

No obstante, durante la configuración previa a la compilación, se eligió uno de los modelos de expulsión que se remarcán en la siguiente imagen:

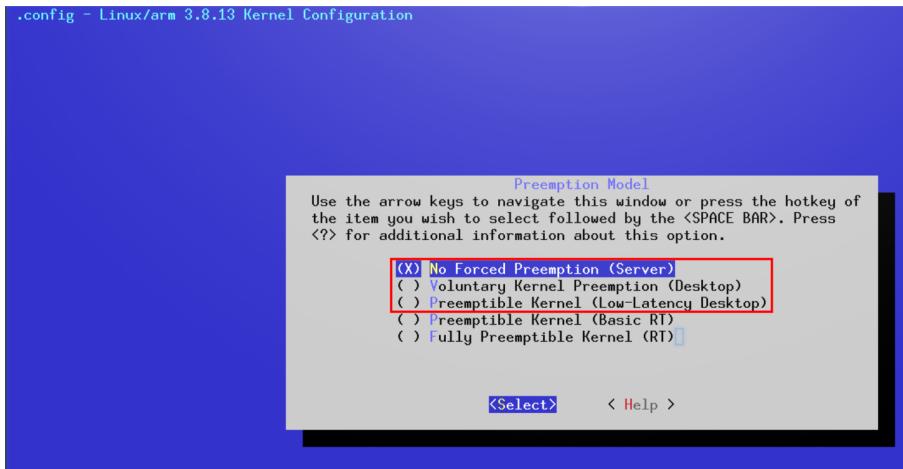


Figura 1: Captura del menú de configuración del modelo de expulsión

Por lo tanto, si bien el kernel usado tiene el parche tiempo real, el modelo de expulsión no es *Full Preemption*, y por lo tanto no se fuerza a que todas las actividades del kernel sean planificables. Por ese motivo, los tiempos medidos son menores que los que se esperaba obtener, porque desde el *irq_handler* se salta directamente a la *ISR*, eliminándose así la latencia de pasar entre medio por el planificador. Sobre el diagrama de las diapositivas de clase [1], puede verse de la forma siguiente:

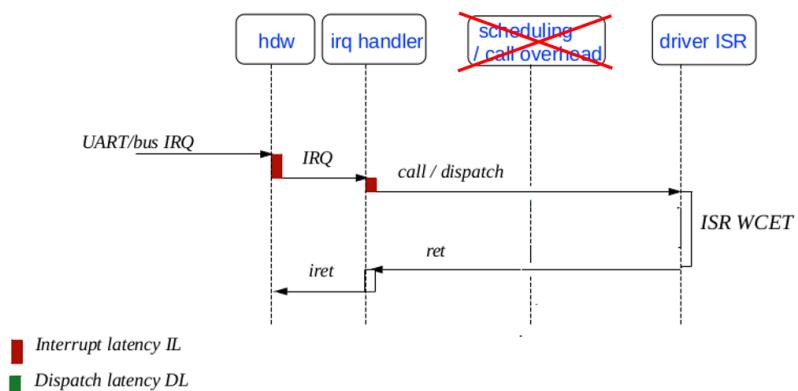


Figura 2: Diagrama de clase editado para evidenciar que no se pasa por el scheduler



2. Primer apartado

En este apartado, se ha realizado la medida de latencia de interrupción con un osciloscopio, usando el módulo cuyo código fuente se nos proporcionaba en el directorio de prácticas de la asignatura. [3].

2.1. Compilación e instalación del módulo

Una vez descargados los fuentes del módulo, se ha procedido a su compilación. Para ello, tras renombrar el fichero *test-irq-latency.c* como *test_irq_latency.c*, ha bastado con seguir los pasos que se indicaban en el guión; se nos proporcionaba un *Makefile*, por lo que solo era necesario modificar el valor de ciertas variables de shell para que todo funcionase.

Una vez con el fichero *test_irq_latency.ko* generado, se ha instalado en la distribución, copiándolo en el directorio raíz de la partición *rootfs*.

2.2. Preparación de las conexiones al osciloscopio

El siguiente paso fue realizar las conexiones necesarias entre los pines de la BeagleBone y el osciloscopio. Una vez hechas, quedan tal que:

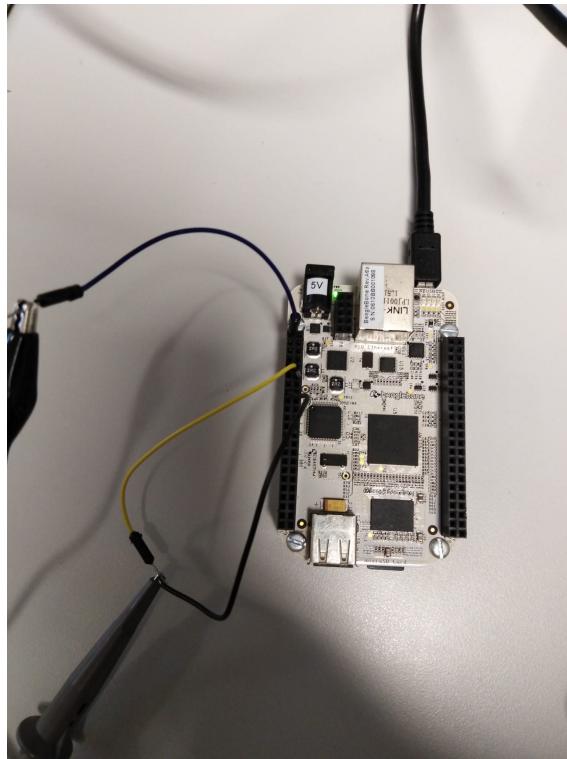


Figura 3: Fotografía a la placa con las conexiones necesarias al osciloscopio

En esta práctica se trabaja con el *expansion header P9*, que son los pines de la izquierda al colocar la placa como en la fotografía. Se puede observar cómo el pin 1, que es la tierra, está conectado a la tierra del osciloscopio, mientras los pines 12 y 15 están unidos entre sí y a la sonda del osciloscopio.



2.3. Medición de la latencia de interrupción

Una vez llegado este punto, se procede a medir la latencia de interrupción por software y con el osciloscopio.

Dentro de la distribución, se ejecuta `insmod test_irq_latency.ko`, que cargará el módulo y hará que se ejecute su función de init, `test_irq_latency_init_module`. En ella se empieza escribiendo en los registros de control de los pines P9/12 y P9/15, para que queden configurados como GPIOs (función `setup_pinmux`, configura ambos pines en modo 7). Despues, se solicita el uso de esos dos pines al kernel, y si todo transcurre sin problemas, al pin P9/15 se le asocia una IRQ, que tendrá como handler la función `test_irq_latency_interrupt_handler`. Despues, se configura un timer para que ejecute la función `test_irq_latency_timer_handler` de manera periódica hasta que se cumpla un tiempo de expiración.

Tras ello, pasará lo siguiente: el pin P9/12 empieza teniendo un valor constante igual a 1, pero de manera periódica, el timer que se ha programado ejecuta la función `test_irq_latency_timer_handler`, y ésta permuta ese valor a 0. Por otra parte, cuando se detecta el valor 0 en el pin P9/15, salta la interrupción que éste tiene asociada, y es atendida por la función `test_irq_latency_interrupt_handler`, que pone el pin P9/12 a 1 de nuevo.

Además de eso, ambas funciones comparten un struct, que es global al módulo. Sirve, por ejemplo, para intercambiar información acerca de si la última permutación del valor del pin P9/12 ha causado una interrupción que ha sido atendida correctamente, y así detectar errores. También almacena variables como `test_count`, que es el contador de eventos generados, y sirve como condición de parada en las mediciones.

En el tiempo que transcurre hasta que se generan `NUM_TESTS` eventos periódicos en el pin P9/12, se puede observar en el canal del osciloscopio lo siguiente:

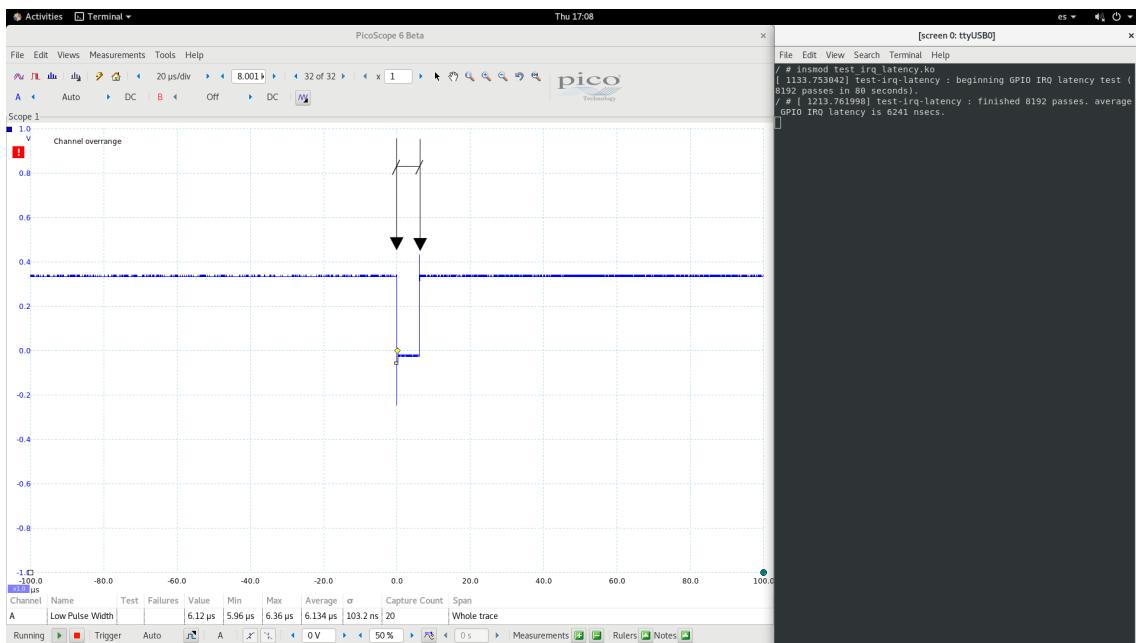


Figura 4: Captura de pantalla del momento en el que terminan los eventos temporizados

Las dos flechas verticales indican el de tiempo que dura la latencia de interrupción, que es el tiempo que pasa desde que la función `test_irq_latency_timer_handler` pone a 0 el valor del pin P9/12 y la función `test_irq_latency_interrupt_handler` lo vuelve a poner a 1.



Cuando se superan los NUM_TESTS eventos generados, la función `test_irq_latency_timer_handler` muestra por pantalla un mensaje con la latencia media que se ha medido por software. Como se puede observar en la figura, la medición hecha con el osciloscopio y la medición software son prácticamente iguales.

2.4. Solución al problema de desmontaje del módulo

El problema se producía al ejecutar `rmmod test_irq_latency.ko`, pues no se obtenía ningún mensaje, pero tampoco se desmontaba el módulo (ejecutando `lsmod` se podía comprobar que el módulo seguía montado).

La solución era ejecutar el comando sin la extensión en el nombre del módulo, es decir, ejecutar `rmmod test_irq_latency`. De esa forma, el módulo quedaba correctamente desmontado.

Otra posible solución era hacer un reset de la placa, de forma que al volver a arrancar el sistema, el módulo ya no estuviese cargado.



3. Segundo apartado

3.1. Añadir segundo canal

En este apartado, se ha realizado la medida de latencia de interrupción usando dos canales del osciloscopio, tras modificar el código fuente del módulo utilizado en el apartado anterior.

3.2. Modificaciones del código fuente

El objetivo que se persigue es el mismo: medir la latencia de interrupción. No obstante, ahora se pretende cambiar ligeramente la forma de hacerlo; en el apartado anterior se asume que los pines P9/12 y P9/15 están conectados, de forma que al permutar el valor del pin P9/12 en una función temporizada, se causa una interrupción en el pin P9/15, en cuyo handler se restaura el valor del pin P9/12, de forma que monitorizando el valor de la señal de la conexión entre los pines se tiene en intervalo de tiempo que está el valor permutado la medida de la latencia de interrupción. Pues bien, con esas mismas conexiones entre pines, ahora la función temporizada permutará y restaurará el valor del pin P9/12, causante de la interrupción en el pin P9/15. Por otra parte, el manejador de interrupción permutará y restaurará otro pin del P9, que será monitorizado por otro canal del osciloscopio, de tal forma que el intervalo temporal entre los flancos de subida de las ondas registradas por los dos canales será la medida de la latencia de interrupción.

El pin elegido para ser permutado por el handler de la ISR es el P9/23. Se ha optado por él porque es el que se propone en el guión de la práctica.

Con el número del pin, P9/23, se va al manual de referencia de la BeagleBone [4]. Se busca el apartado relativo al *expansion header P9*, que está en las páginas 59 a 63 (apartado 6.13.3). En él hay varias tablas, entre las cuales se empieza mirando la 11 (página 59), en la que se nos indica que el nombre de la señal de la placa asociada al pin P9/23 es GPIO1_17. Tal y como se explica en el guión de la práctica, esto nos permite saber el número con el que ese pin está representado en la interfaz de GPIOs del kernel de Linux; dado que el nombre de la señal es GPIO1_17, calculando $1 \cdot 32 + 17 = 49$, se tiene que el pin P9/23 se representa con el número 49. En la tabla 13 (página 62), se indica que para usarlo como GPIO, hay que ponerlo en modo 7 (al igual que pasaba con los pines P9/12 y P9/15). Por otra parte, en la tabla 12 (página 60), está la información de los modos 0 a 3. El modo 0 indica el nombre del registro de control asociado al pin en el SoC de la placa, y en el caso del P9/23, es *gpmc_a1*. Con ese nombre, se va al manual de referencia del SoC de la BeagleBone [5]. En el apartado 9.3 se listan los registros de control de entrada/salida y su offset sobre la dirección base con el que están mapeados en memoria. Así, en la tabla 10 (página 1459), se indica que el offset del registro de control del GPIO *conf_gpmc_a1* es *844h*.

Con esa información, en la función *setup_pinmux* se configura ahora también el pin P9/23, accediendo a su registro de control mediante *AM33XX_CONTROL_BASE + 0x844*, y poniéndolo en modo 7, PULLUP, OUTPUT (que es como está configurado el pin P9/12). Es necesario añadir al struct *irq_latency_test* un campo en el que se almacene el número del pin P9/23 en el interfaz de GPIOs del kernel, de manera análoga a como se hace con los otros dos pines. Después, en la función *test_irq_latency_init_module* hay que programar la solicitud de uso del nuevo pin al kernel, mediante *gpio_request_one*, con una invocación análoga a la que se hace con el pin P9/12. Por último, en *test_irq_latency_exit_module* hay que comunicar al kernel que se deja de hacer uso del pin P9/23, invocando la función *gpio_free* con el número 49 (que es con el que ese pin se representa).

Con esas modificaciones queda hecha toda la configuración relativa al nuevo pin. Solo falta modificar la función *test_irq_latency_timer_handler* para que restaure el valor del pin P9/12 después de permutarlo. Para que la anchura de la onda se alargue, se ha añadido un bucle que realiza 5000



iteraciones de instrucciones null. Por último, en la función `test_irq_latency_interrupt_handler`, en vez de manipular el pin P9/12, ahora todas las acciones se realizan sobre el pin P9/23, permutando su valor al iniciar la función y restaurándolo antes de hacer el retorno.

3.3. Preparación de las conexiones al osciloscopio

La forma de realizar las conexiones del primer canal del osciloscopio es la misma que en el apartado anterior.

Al añadir el segundo canal, éste debe quedar como se muestra en la siguiente imagen:

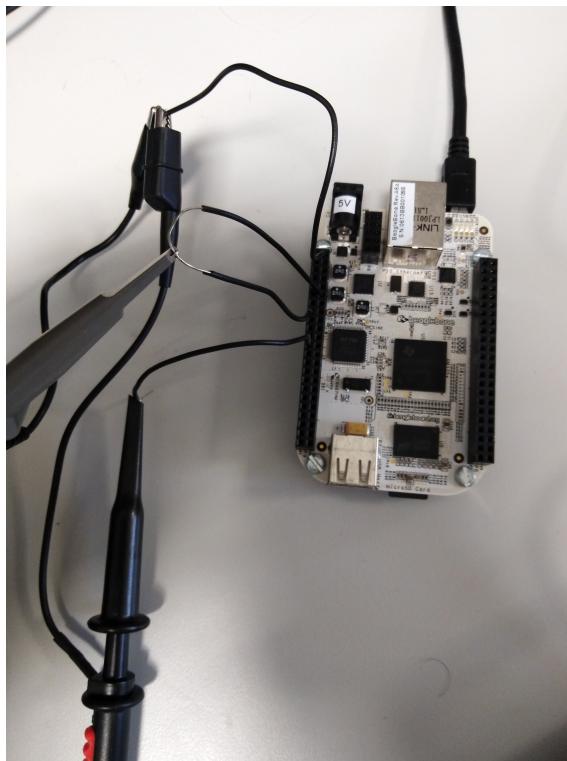


Figura 5: Fotografía a la placa con las conexiones necesarias al osciloscopio

Se puede observar cómo el pin 1, que es la tierra, está conectado a la tierra del segundo canal del osciloscopio, mientras que su sonda está monitorizando el pin 23.

3.4. Medición de la latencia de interrupción

Para medir la latencia de interrupción se ha procedido de la misma forma que en el anterior apartado, y se han obtenido los siguientes resultados:

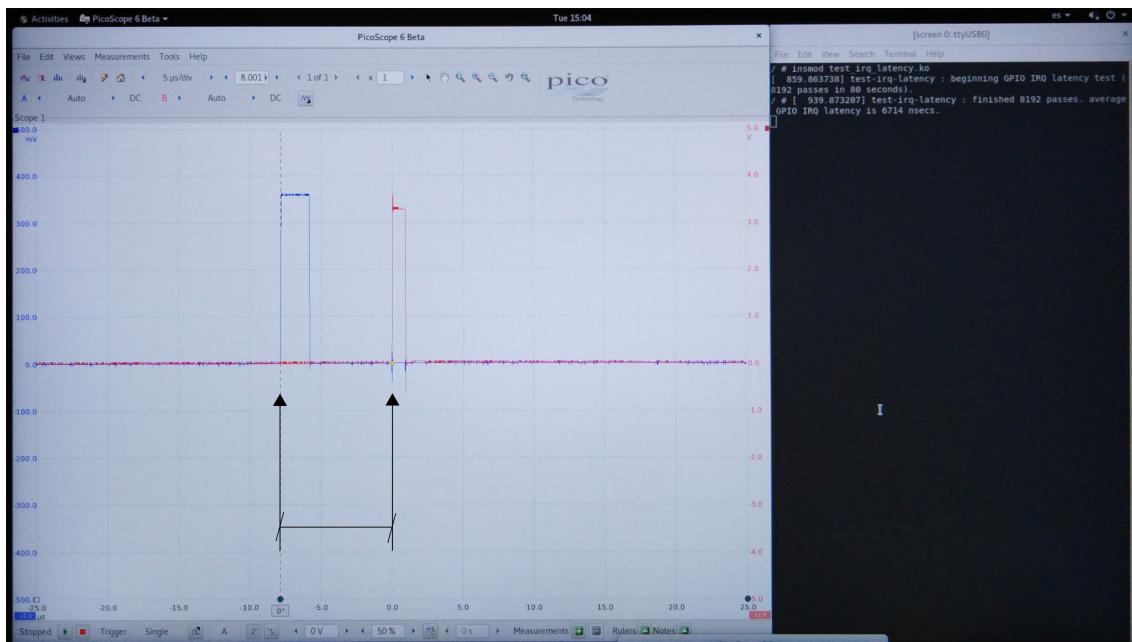


Figura 6: Fotografía de la pantalla del momento en el que terminan los eventos temporizados

Las dos flechas verticales indican el de tiempo que dura la latencia de interrupción, que es el tiempo que pasa desde que la función `test_irq_latency_timer_handler` pone a 1 el valor del pin P9/12, medido por el canal 1 (color azul), y la función `test_irq_latency_interrupt_handler` pone a 1 el valor del pin P9/23, medido por el canal 2 (color rojo).

Nótese cómo, por un descuido de programación, la permutación de valores es de 0 a 1, y no de 1 a 0, como ocurría en el apartado anterior.

Debido a problemas con el programa picoscope, no se pudo hacer una medición precisa de la latencia sobre la gráfica, pero proporcionalmente, en base la regla en la parte inferior de la ventana, se puede observar como los tiempos de medición de latencia que se tienen son muy parecidos a los que se obtienen por software, al igual que pasaba en el apartado anterior.



Referencias

- [1] *Diapositivas de la asignatura del tema sobre excepciones.* URL: https://moodle.unizar.es/add/pluginfile.php/2824553/mod_resource/content/3/SE2_09_Exceptions.pdf (visitado 15-01-2021).
- [2] *Fuentes de Linux con el parche tiempo real aplicado.* URL: <sftp://hendrix.cps.unizar.es/misc/practicas/SE-II/PR3/kernel-rt.xz> (visitado 15-01-2021).
- [3] *Fuentes del módulo con el que se ha medido la latencia de interrupción.* URL: <sftp://hendrix.cps.unizar.es/misc/practicas/SE-II/PR4> (visitado 15-01-2021).
- [4] *Manual de la placa BeagleBone.* URL: https://moodle.unizar.es/add/pluginfile.php/2824583/mod_resource/content/1/BeagleBoneR6.SysRefMan.pdf (visitado 15-01-2021).
- [5] *Manual de referencia del SoC de la placa BeagleBone.* URL: <https://www.ti.com/lit/ug/spruh73q/spruh73q.pdf> (visitado 15-01-2021).