FH-OÖ Hagenberg/ESD
Embedded Operating Systems, WS 2014
Rainer Findenig, Florian Eibensteiner, Josef Langer © 2014

4. Übung: Embedded Linux: Driver Development, Part Two

Name(n): Punkte:

1 Vorbereitung

Für diese Übung sollten Sie die wichtigsten Teile von Kapitel 10 von *Linux Device Drivers* [CRK05], speziell die Abschnitte „Installing an Interrupt Handler“, „The /proc Interface“, „Top and Bottom Halves“ und „Implementing a Handler“, gelesen haben.

☐ Beantworten Sie dazu die folgenden Fragen:

- Kann ein *interrupt handler* Speicher allokieren? Begründen Sie Ihre Antwort!
- Was unterscheidet *Top* und *Bottom Halves*? Wozu ist diese Unterscheidung notwendig?

2 Interrupts für Schalter 1

Erweitern Sie Ihren Treiber aus der letzten Übung derart, dass bei jedem Auftreten einer steigenden Flanke auf GPIO 15 (Schalter 1) ein möglichst kurzer Low-Puls auf GPIO 79 (LED 1) ausgegeben wird.

Sie können die Nummer des Interrupts mit Hilfe der PXA-spezifischen Funktion `IRQ_GPIO` (aus `asm/arch/irqs.h`) bestimmen:

```
1 #include <asm/arch/irqs.h>
2
3 #define SWITCH0_IRQ IRQ_GPIO(15)
```

Die Funktion `set_irq_type` ermöglicht es, den Interrupt nur für die steigende Flanke zu aktivieren:

```
1 set_irq_type(SWITCH0_IRQ, IRQT_RISING);
```

☐ Testen Sie nun Ihren *interrupt handler*! Prüfen und erklären Sie dabei die Informationen in der Datei `/proc/interrupts`!

3 Tasklets

Erweitern Sie Ihren Treiber um ein Tasklet, das im Interrupt gestartet (`tasklet_schedule()`) wird. Dieses Tasklet soll, ähnlich wie der *interrupt handler*, einen möglichst kurzen Puls auf GPIO 36 (LED 2) ausgeben.

4 Workqueues

Erweitern Sie Ihren Treiber um eine Workqueue, deren Funktion ebenfalls im Interrupt gestartet (`schedule_work()`) wird. Die Funktion soll einen möglichst kurzen Puls auf GPIO 37 (LED 3) ausgeben.

Wichtig: Bei allen Tests soll entweder ein Tasklet oder eine Workqueue aktiv sein. Nie beide gleichzeitig. Das Verhalten von Tasklets und Workqueues soll miteinander verglichen werden.

5 Real-Time-Verhalten von Linux

Der Test, der in diesem Beispiel aufgebaut werden soll, ist an jenen Tacke und Ricci [TR02] angelehnt, dieser wiederum baut auf dem Messverfahren von Dupré und Baracos [DB01] auf. Daher erscheint es sinnvoll, dass Sie diese beiden Paper genau lesen!

Sie sollen mit Hilfe des in Abbildung 1 dargestellten vereinfachten Testaufbaus die Latency und den Jitter bestimmen der Systems bestimmen. Verwenden Sie dazu einen Funktionsgenerator, um Interrupts am System auszulösen. Die Anschlussbelegung ist in Tabelle 1 zusammengefasst. Messen
□ Sie die (durchschnittlichen) Interrupt-, Tasklet und Workqueue-Latenzen!

Name	**GPIO**	**X2**
`OUT_ISR` (LED 1)	79	15
`OUT_TL` (LED 2)	36	14
`OUT_WQ` (LED 3)	37	13
`IN` (Schalter 1)	15	16
`GND`		20

Table 1: Anschlussbelegung.

Anmerkung: Abbildung 2 zeigt, wie sich Jitter und Latency mit einem Oszilloskop, wie in Abbildung 1, gut aufnehmen lassen: Durch das Aktivieren der Option „∞ Persist“ wird jedes Auftreten der Signale OUT ISR (in diesem Bild Signal D_0) und OUT TL/WQ (D_1) gespeichert und (in diesem Fall) in grau dargestellt.

Wiederholen Sie diese Messungen unter Systemlast (zB. `md5sum /dev/urandom`) und vergle-
□ ichen Sie die Latenzen bzw. den Jitter mit den Werten des unbelasteten Systems.

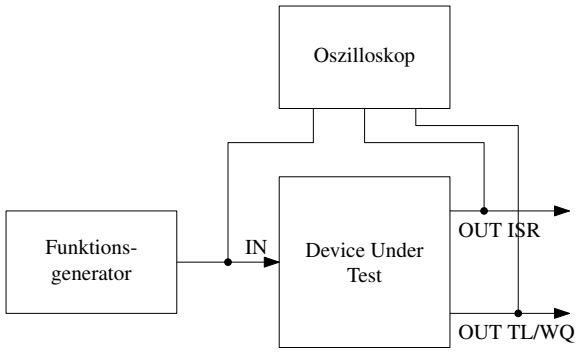

Figure 1: Testaufbau für Aufgabe 1.

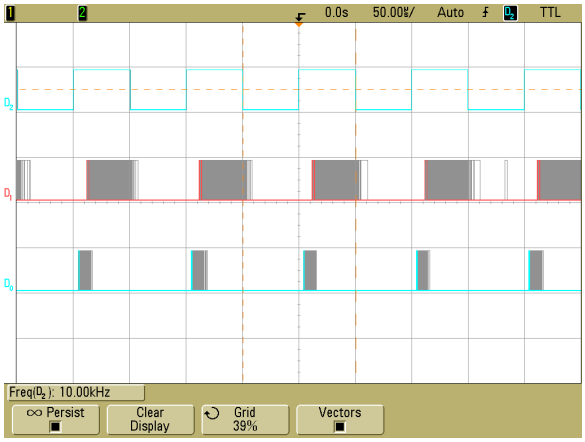

Figure 2: Messung der Latency und des Jitters mit der Option „∞ Persist“.

References

[CRK05] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media, Inc., 3rd edition, February 2005.

[DB01] Joseph K. Dupré and Paul Baracos. Benchmarking real-time determinism. Technical report, International Society for Measurement & Control, 2001.

[TR02] Chris Tacke and Lawrence Ricci. Benchmarking real-time determinism in Windows CE. Technical report, Applied Data Systems, 2002.

“ / These are the most dangerous and useful defines. They do printk() during*
** the interrupt processing routine(s), so if you manage to get “flooded” by*
** irq's, start thinking about the “Power off/on” button...*
**/”*

linux-2.2.16/drivers/sbus/char/aurora.h