Application Note

Shared IRQ Line Considerations

AN-PM-059

Abstract

When IRQ line-sharing between multiple devices has been imposed by the target hardware design, a system failure may occur that is intrinsic to the Linux kernel. This document outlines recommendations to avoid such issues. Several solutions have been identified and each should be considered on its merits for the target platform under examination

Contents

Figures

Tables

1 Terms and Definitions

CPU	Central Processing Unit
GPIO	General Purpose Input/Output
I^2C	Inter-Integrated Circuit (Bus)
ISR	Interrupt Service Routine
IRQ	Interrupt Request (Line)
OTP	One Time Programmable (Memory)
OS	Operating System
PMIC	Power Management Integrated Circuit
POR	Power On Request
RTC	Real Time Clock
SoC	System on Chip
SPI	Serial Peripheral Interface

2 References

[1] DA9063, Datasheet, Dialog Semiconductor

[2] AN-PM-047, Renesas R-Car E2 platform for automotive infotainment, Application Note, Dialog Semiconductor

[3] AN-PM-049, Renesas R-Car M2 platform for automotive applications, Application Note, Dialog Semiconductor

[4] AN-PM-050, Renesas R-Car H2 platform for automotive applications, Application Note, Dialog Semiconductor

3 Introduction

This document describes a system failure intrinsic to the Linux kernel when IRQ line-sharing has been imposed by the target hardware design.

4 Shared Interrupt Line

Figure 1 diagram shows a typical system where a Dialog PMIC delivers power to the SoC and shares the IRQx input with two other devices.

Each device has its own software interrupt service routine (ISR). These ISRs are called in turn by the operating system once the IRQx is received.

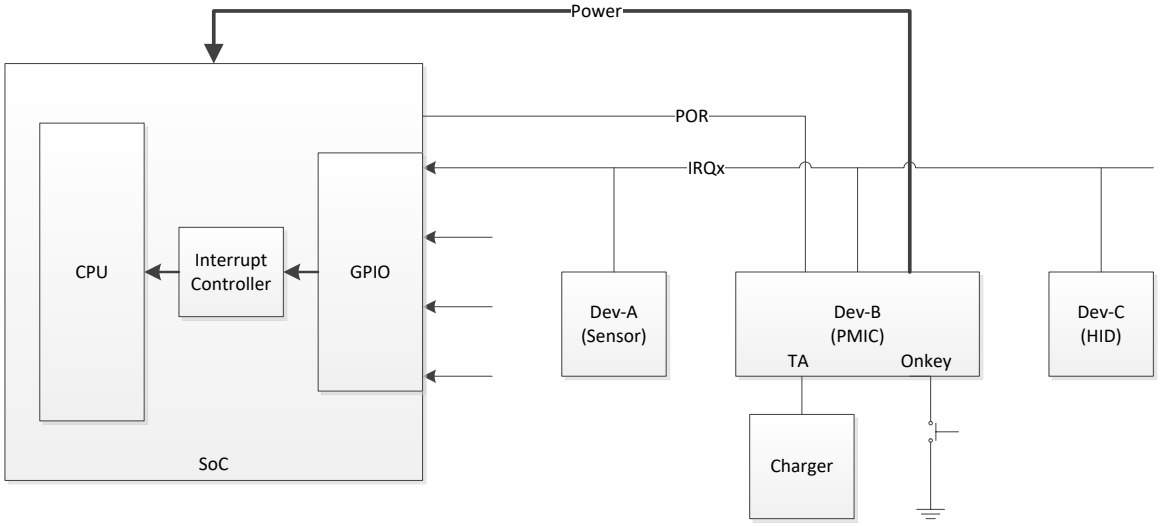

Figure 1: Shared Interrupt Line Block Diagram Example

4.1 Symptoms and Mode of Failure in the Linux Kernel

The first sign of a failure within the Linux kernel is an unhandled interrupt. This behavior can be seen by polling the filesystem's proc interface (if this filesystem has been enabled in the kernel).

```
cat /proc/interrupts
```

Checking the console output for a rapidly increasing number of interrupt calls can provide the first indication that an IRQ is not being handled correctly. If this first symptom is ignored, the next direct manifestation will be a kernel message of the form:

```
irq N: nobody cared (try booting with the "irqpoll" option)
Disabling IRQ #N
```

The next kernel action is to disable the IRQ that remains un-serviced. There is a risk that the kernel will disable all shared interrupts on the IRQ line, including those that may be from correctly functioning devices.

The exact conditions are set in the Linux kernel core but, if approximately 100 000 interrupts have not been handled, the assumption is made that the IRQ is held in some manner and is not being serviced correctly. A diagnostic message and stack trace are sent to the console, then the kernel attempts to turn off the offending IRQ. Future IRQ notifications on the line are then ignored.

4.2 Example of Causation

The following related events are required to cause the fault behavior.

1. An event wakes up the PMIC, such as ONKEY, RTC alarm, or charger insertion.
2. The PMIC establishes the power rails.
3. If a corresponding wake-up event is unmasked, the Dev-B (PMIC) may assert its nIRQ output (low).
4. The system loads and runs the OS.
5. During initialization, the kernel installs the drivers and registers the corresponding ISR for each device in a predetermined order (this ordering is system dependent and assumed unknown). The first handler registered causes the IRQ line to the CPU to be enabled.
6. The SoC GPIO and interrupt controller are configured to receive interrupt request IRQx. The registered ISRs for the IRQx are called by the OS in turn until the IRQx becomes de-asserted.
7. If an ISR for either Dev-A or Dev-C is called before the PMIC device driver has been installed, the CPU may repeatedly call the Dev-A and/or Dev-C ISR as the interrupt line will still be held by Dev-B (PMIC).

4.3 General Causes

- The PMIC issues an interrupt request before the system is ready to receive it.
- The PMIC OTP does not mask all interrupts by default.
- An inability to control the order of Linux device driver installation.
- Interrupt line-sharing has not been considered during the system integration of the device drivers.
- The driver has not registered all of its IRQ entries with the kernel core and therefore the IRQs are not properly controlled or masked by the kernel.

5 Solutions

There are several software solutions to handling a shared interrupt line which can be categorized as:

- configuration or hardware-based
- run-time solutions executed in software

5.1 Allocate a Dedicated IRQ Line

Whilst not always possible, allocating a dedicated IRQ input for the Dialog PMIC device resolves the line-sharing issue and does not require any workarounds or software intervention.

5.2 OTP IRQ Mask Programming for all Interrupts

Preventing the Dialog PMIC from generating an IRQ during the system start-up sequence is suited to all OS applications and does not require any software workarounds in the bootloader or Linux kernel.

Programming the IRQ mask bits in the PMIC OTP could resolve the issue. However, these mask bits are generally left disabled in some PMICs to retain compatibility with other system requirements. Programming these OTP bits therefore cannot be considered a general solution.

See Section 6 for device-specific recommendations.

In some PMICs, masking all interrupts can have side-effects and may affect wake-up operations: for example, the DA9063 wake-up port (CHG_WAKE) feature would be disabled since, ‘The IRQ assertion and wake-up event can be suppressed via the interrupt mask M_WAKE’. See [1], page 45.

5.3 Bootloader IRQ Masking

Early masking of interrupts within the PMIC can be performed during the bootloader section of startup. This is the software equivalent solution of Section 5.2 and requires a bootloader to exist in the system and the capability to modify the software in the bootloader accordingly.

This software workaround requires a customer- or platform-specific implementation and its suitability is therefore limited to a fully-integrated target system.

5.4 Kernel Platform Quirk IRQ Masking

A kernel-only solution can be implemented using a ‘kernel quirk’. A kernel quirk is usually a platform-specific modification which is called during the start-up code of the Linux kernel.

This solution has the disadvantage of requiring specific workaround quirks for each target platform and also ignores the IRQ problem until the kernel has started to load. It has the advantage of being available to all identified platforms as part of the integrated kernel distribution.

See the Linux kernel example for platform solutions for the compatible targets ‘renesas,koelsch’ and ‘renesas,lager’:

```
arch/arm/mach-shmobile/regulator-quirk-rcar-gen2.c
```

The PMICs used in these target systems are DA9063 and DA9210. The OTP settings used for these platforms have been set such that the PMIC cold boot or restart has unmasked interrupts. Any interrupts that are triggered upon PMIC start-up are a cause of an interrupt storm.

Without the quirk, an IRQ line-sharing failure occurs as soon as one driver is installed by the Linux kernel and requests the IRQ. It immediately gets stuck in an infinite loop since it can only de-assert its own interrupt request line, and because the other driver has not yet installed an interrupt handler. The quirk masks the interrupts in both the DA9063 and DA9210: it must execute after the I^2C master driver but before the I^2C slave drivers are initialized.

See Appendix A for further details.

5.5 Disable the OS from Receiving Interrupts

Specific code can be added during driver initializations to prevent the OS from receiving a PMIC interrupt until the PMIC device driver has been installed correctly. This disallows IRQ sources until all driver IRQ handlers have been serviced.

This is an intrusive solution and requires significant code alteration specific to the OS being used.

5.6 Disable the IRQ Input in the SoC GPIO

Interrupts can be masked by I^2C/SPI commands to the SoC, before installing any device drivers that use the IRQx line shared with the PMIC. This disallows IRQ sources until the driver IRQ handlers are all serviced. However, the alteration would be application-dependent and require deep understanding of the target system. This level of information may not be readily accessible when designing a system that is not open source.

5.7 Static Arrangement of Device Driver Installation Order

This solution involves statically arranging the device driver installation into a fixed order within the Linux kernel to avoid any driver installation race conditions that would otherwise cause an interrupt storm problem.

This could be used as a temporary fix in some cases, however it is not a general solution in most practical applications for several reasons. For example, if multiple devices flag an IRQ at start-up, an IRQ storm will still occur regardless of ordering. Also, when using later Linux kernels and a device tree solution, this would be totally dependent on the device tree implementation or would involve modification of the Linux core functions. In general, such modification is difficult to achieve for most large scale OS-based applications.

6 Specific PMIC Recommendations

The following recommendations are made for individual PMICs.

6.1 DA9210

There are several recommendations for DA9210:

- OTP default for setting all interrupt masks
- clearance of event bits upon start-up

6.1.1 Interrupt Masks

The standard OTP for DA9210 has all interrupts masked by default. However, the OTP mask settings should be verified for each system since they can be unmasked for some OTP variants. For specific details of Renesas R-Car E2, M2 and H2 platforms see [2], [3], and [4].

6.1.2 Event Bits

If OVCURR or NPWRGOOD occurs during the DA9210 boot-up, and they are deemed to be unintentional events, the event flag of E_OVCURR or E_NPWRGOOD is set even if the masking nIRQ toggle on MASK_B has been set by the OTP configuration settings.

The recommendation is for the system software to read the EVENT_B register and clear the flags on EVENT_B during the start-up flow.

Table 1: Event Bit Fields Associated with DA9210

Event Bit Field	Recommended for Clearance on Start-up
DA9210_E_OVCURR	Yes
DA9210_E_NPWRGOOD	Yes
DA9210_E_TEMP_WARN	No
DA9210_E_TEMP_CRIT	No
DA9210_E_VMAX	No

6.2 DA9063

The general recommendation for DA9063 is to clear the FAULT_LOG register on start-up.

6.2.1 Fault Log Clearance

If the DA9063 has been shutdown using the hardware ONKEY reset function (long press of ONKEY when the software fails to respond), the standard DA9063 ONKEY Linux device driver will not be able to reset the KEY_RESET within the FAULT_LOG register. The FAULT_LOG is a persistent register which holds its value across a reboot and therefore will affect the start-up sequence during the next system restart.

The KEY_RESET bit triggers a non-maskable interrupt which has no corresponding event bit in the event registers and therefore cannot be handled by the Linux framework in the normal way. This bit should be cleared upon start-up, either during the bootloader or the kernel driver probe operation.

See Appendix B for further details.

7 Conclusions

The software device driver issues caused by IRQ line-sharing between multiple devices can be avoided if the recommendations presented in this document are followed.

Each solution described should be considered on its own merits and the most appropriate way forward should be chosen for the target platform. No single solution can be identified to solve this problem.

Appendix A Linux Kernel Platform Quirk

This example is intended for the Linux kernel and should be modified for the platform device requirements. It assumes the following:

`arch/arm/mach-imx/platform-quirk.c`

The location for setting IRQ mask registers is indicated by /*`SET IRQ MASKS` */.

A.1 I²C Platform Quirk for Altering IRQ Mask Registers during Linux Startup

```
#include <linux/mfd/da9xxx/registers.h>

static int platform_i2c_bus_notify(struct notifier_block *nb,
                                   unsigned long action, void *data)
{
       struct device *dev = data;
       struct i2c_client *client;

       if (action != BUS_NOTIFY_ADD_DEVICE || dev->type == &i2c_adapter_type)
              return 0;

       client = to_i2c_client(dev);

       /* Assume I2C slave address 0x68 and da90xxx name */
       if ((client->addr == 0x68 && !strcmp(client->name, "da9xxx"))) {
              /* SET IRQ MASKS */
              i2c_smbus_write_byte_data(client, <REGISTER>, <VALUE> );
              bus_unregister_notifier(&i2c_bus_type, nb);
       }

       return 0;
}

static struct notifier_block platform_i2c_bus_nb = {
       .notifier_call = platform_i2c_bus_notify
};

static int __init platform_quirk(void)
{
       bus_register_notifier(&i2c_bus_type, &platform_i2c_bus_nb);
       return 0;
}

arch_initcall(platform_quirk);
```

Figure 2: IRQ Masking Quirk for the Linux Kernel

A.2 Installing the Quirk into the Linux Build

To include the quirk, a modification to the Makefile will be necessary:

`arch/arm/mach-imx/Makefile`

```
obj-$(CONFIG_SOC_IMX6Q) += clk-imx6q.o mach-imx6q.o platform-quirk.o
```

Appendix B Software Fault Log Clearance for the DA9063

B.1 Kernel Clearance

Clearing the FAULT_LOG register can be implemented during the device driver probe of the DA9063. See the file `drivers/mfd/da9063-core.c` and function `da9063_clear_fault_log` implemented in the linux-mainline/v4.2 kernels onwards.

B.2 U-Boot Bootloader Clearance

The same code example as described in Section B.1 but targeted at U-Boot is given in Figure 3.

```
#define DA9063_I2C_SLAVE 0x58
#define DA9063_FAULT_LOG_REG 0x05

unsigned char value = 0;

if (!(i2c_read(DA9063_I2C_SLAVE, DA9063_FAULT_LOG_REG, 1, &value, 1))) {
       if (value & 0x20)
              printf("Power down from a long press of nONKEY or GPIO14/15\n");
       /* Clear the DA9063 FAULT_LOG on start-up */
       i2c_write(DA9063_I2C_SLAVE, DA9063_FAULT_LOG_REG, 1, &value, 1);
}
```

Figure 3: DA9063 FAULT_LOG Reset Source Code for U-Boot

Revision History

Revision	Date	Description
1.0	18-Sep-2015	Initial version.
1.1	12-Oct-2015	Positive rewording of the abstract to reflect multiple solutions offered by this document.
1.2	15-Jun-2016	Update formatting style to follow document conventions. Section 5.2 combine two sections into one, clarifying that normal IRQs and wake-up interrupts would cause the same problems.
1.3	10-Mar-2022	File was rebranded with new logo, copyright and disclaimer

Shared IRQ Line Considerations

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Dialog Semiconductor's suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.