Development of "Interrupt Storm Detection" Feature

October 2020

Issued by Kento Kobayashi, R&D Center, Sony Corporation

Agenda

- Background
  - What is interrupt storm?
  - Cases of interrupt storms
  - Existing ways to debug interrupt storms for each cases
- Our solution
  - Interrupt storm detection feature
  - Example of using interrupt storm detection feature for actual problem

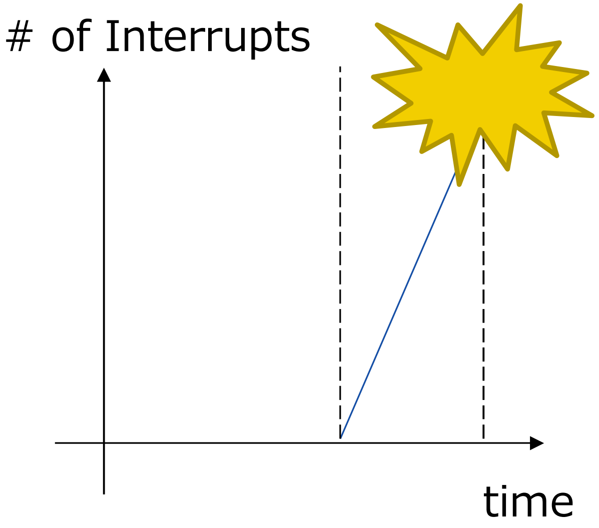
Self introduction

- Name
  - Kento Kobayashi
- Company
  - Sony Corporation
- Responsible for
  - Linux kernel and device drivers for Sony products.

Background

What is “Interrupt Storm”?

- “Interrupt Storm” is a continuous hardware interrupt to CPU.
  - CPU needs to execute interrupt handlers continuously.
- “Interrupt Storm” causes:
  - System hang-up due to high CPU utilization by the interrupt handler
  - Difficult to debug because console is not responding
- To debug interrupt storm:
  - Need to identify IRQ number which causes interrupt storm.
- Cases of “Interrupt Storm”:
  - Case1 : Unhandled(Spurious) interrupt
  - Case2 : High-frequency handled interrupt



Case1 : Unhandled(Spurious) interrupt

- **What is “Unhandled(Spurious) interrupt”?**
  - Interrupt handler doesn’t handle hardware interrupt
- **Why “Unhandled(Spurious) interrupt” occur?**
  - Problem of device driver.
  - Interrupt handler do nothing if that interrupt is not own interrupt.
  - Then interrupt status is not clear, so interrupt is raised continuously.
- **Example of “Unhandled(Spurious) interrupt” case**
  - Shared IRQ by multiple device driver.
    - Interrupt handler is executed whether not own interrupt.
    - Then if interrupt handler not recognize as own interrupt wrongly, nobody handled raised interrupt.
  - Not registered interrupt handler
    - Then nobody handled raised interrupt.

How to debug Case1: Unhandled(Spurious) interrupt

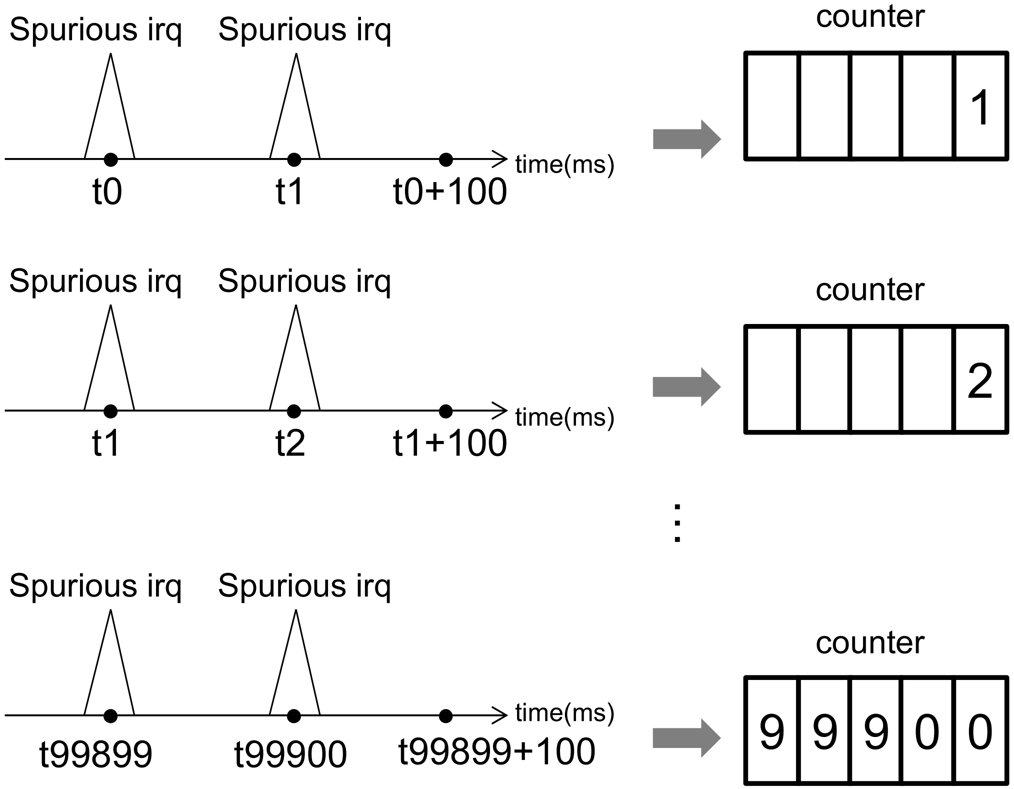
- **Using “spurious interrupt handling” kernel feature.(after v2.6.10)**
  - Disable interrupt and print IRQ number after detect 99900[times] spurious interrupt.
- **How to debug with “spurious interrupt handling”**
  - This feature shows the following message.

```
irq 15: nobody cared (try booting with the "irqpoll" option)
Disabling IRQ #15
```

  - We can know interrupt storm is occurred in which IRQ number.
  - Then we can know which device driver we should investigate from /proc/interrupts.

```
# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
…snip…
  15:      34673      33826      34696      33641            level  64 Edge      foo
…snip…
```

“spurious interrupt handling” kernel feature mechanism



- Increment counter if spurious interrupt occur within 100ms of the previous spurious interrupt.

- Disable IRQ if counter reaches 99900.
- Display “Disabling IRQ#XX” in kernel log.
- Clear counter if spurious interrupt is not occurred within 100ms of the previous spurious interrupt.

Case2 : High-frequency handled interrupt

- **What is “High-frequency handled interrupt”?**
  - Interrupt handler handled interrupt, but interrupt is raised continuously.
- **Why “High-frequency handled interrupt” occur?**
  - Problem of hardware or device driver.
  - Interrupt is raised continuously whether clear interrupt cause.
- **Example of “High-frequency handled interrupt” case**
  - Hardware design mistake or design change
    - Usually occurs at start phase of development
  - Wrong interrupt trigger setting.
    - Then interrupt status is remains “interrupt occur”, interrupt will be raised continuously.
  - Forget clear interrupt cause
    - Then interrupt cause remains, interrupt will be raised continuously.

How to debug Case2: High-frequency handled interrupt

- **Using NMI (Non-maskable Interrupt) functionality**
  - What is NMI?
    - Interrupt and dump CPU registers and backtrace even if under “Interrupt Storm”.
  - Problems
    - Need to secure about how to use and invoke NMI for your board
      - NMI cannot be used on some systems or boards.
      - Can’t detect as “Interrupt Storm”.
    - Need to invoke NMI multiply to find interrupt number which causes “Interrupt Storm”
- **Using JTAG equipment**
  - What is JTAG?
    - Snoop CPU registers, memory contents.
    - Specify which interrupt handler works hard.
  - Problems
    - Need to secure about how to enable JTAG for your board
    - JTAG equipment is expensive :(

How to debug Case2: High-frequency handled interrupt

- **Using PSTORE_FRACE**
  - What is PSTORE_FTRACE
    - PSTORE_FTRACE records function call history into your persistent memory.
  - How to use?
    - Enable PSTORE_FTRACE by following command before “Interrupt Storm” occur.

```
echo 1 > /sys/kernel/debug/pstore/record_ftrace
```

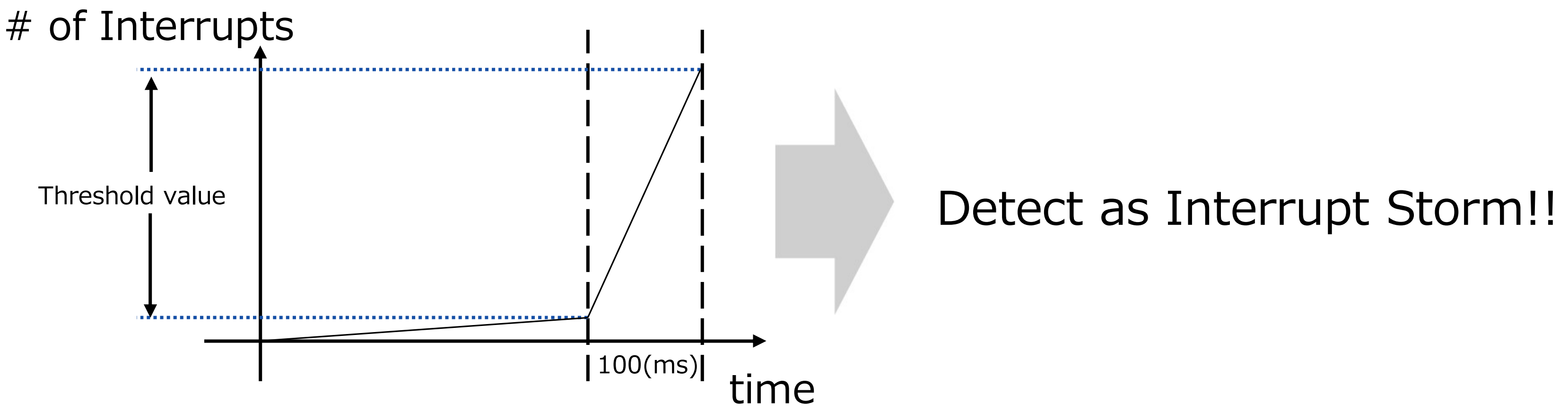
    - Reboot your board by pressing reset button once storm occur.
    - Confirm function call history by just before reboot from files under /sys/fs/pstore/* .
  - Problems
    - Persistent memory (including System RAM) is unavailable in some systems.
    - Enabling PSTORE_FTRACE changes system’s behavior.
      - Affect performance impact due to records function call history.

Those ways has some problems to debug interrupt storm!!

Our solution

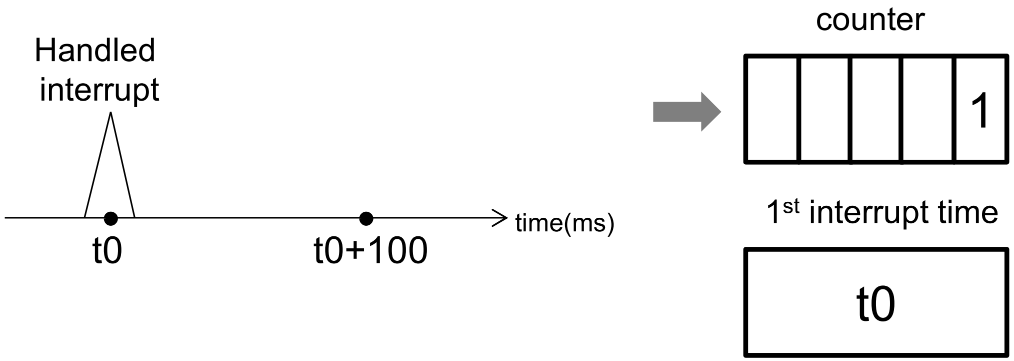
"Interrupt Storm Detection" feature – Summary

- Summary of features
  - Detect as interrupt storm if number of interrupt **exceeds a threshold per 100ms**.
  - **Print the IRQ number** to kernel log if interrupt storm is detected.
  - **Threshold can be set** by the user.
  - Can disable corresponding interrupts after detection.
  - Can invoke **kernel panic** after detection for debug.



"Interrupt Storm Detection" feature – Detail of mechanism

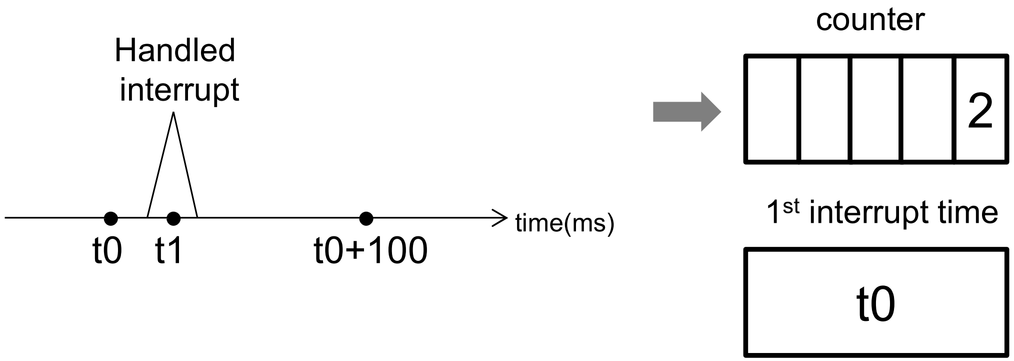
- Case where the threshold is set to 1000[times/100ms].



- Increment counter if handled interrupt occurs
- Records 1st interrupt time

"Interrupt Storm Detection" feature – Detail of mechanism

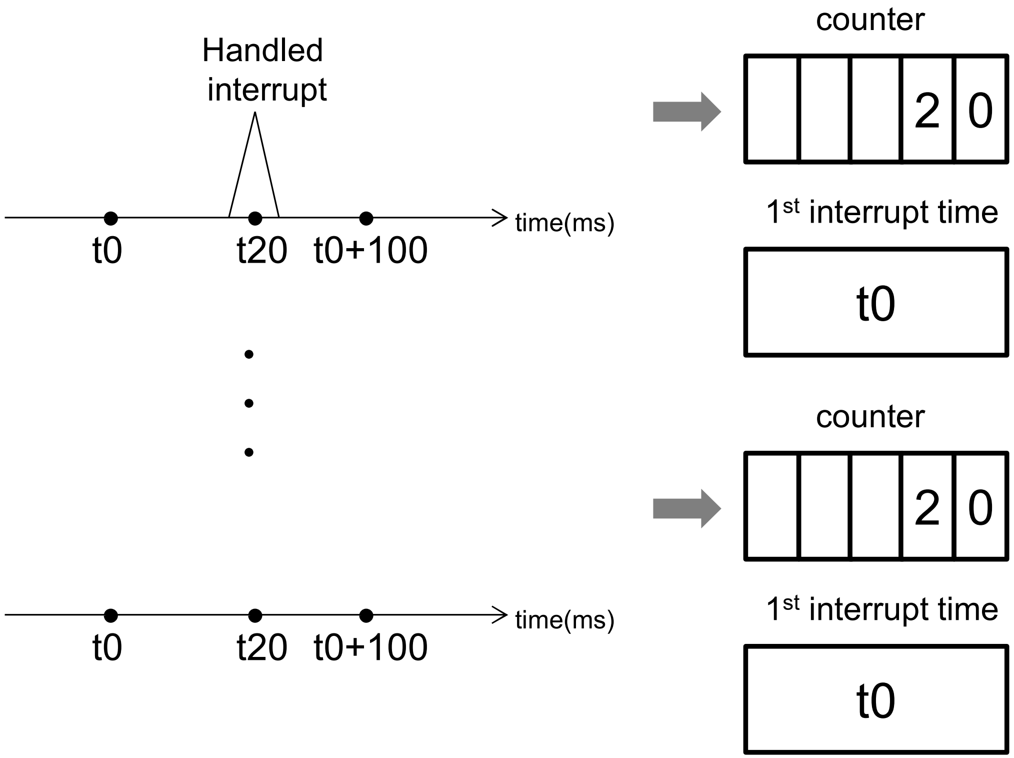
- Case where the threshold is set to 1000[times/100ms].



- Increment counter if handled interrupt occurs within 100[ms]

"Interrupt Storm Detection" feature – Detail of mechanism

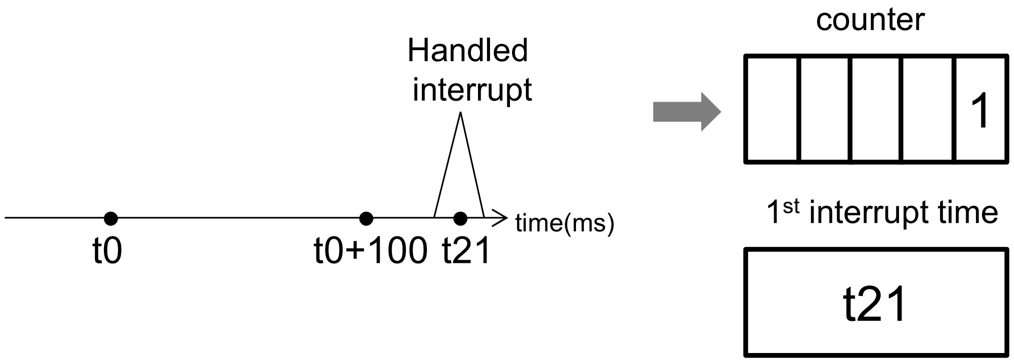
- Case where the threshold is set to 1000[times/100ms].



- If counter is not reached threshold, interrupt storm is not occurred

"Interrupt Storm Detection" feature – Detail of mechanism

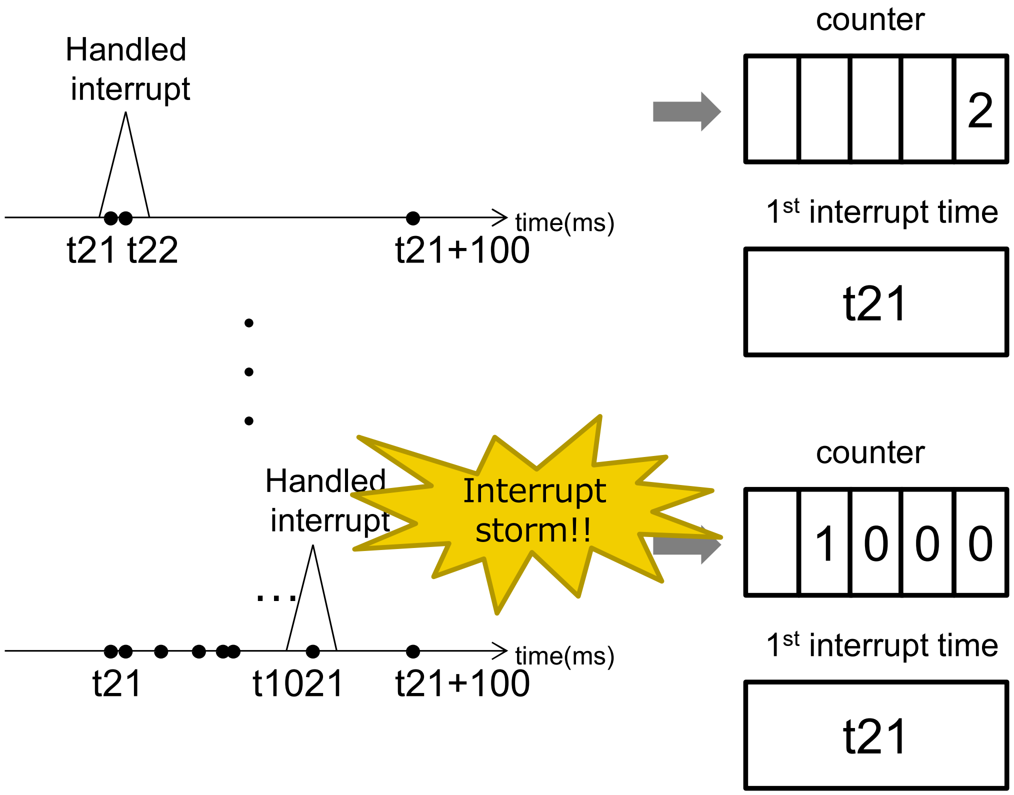
- Case where the threshold is set to 1000[times/100ms].



- If handled interrupt is occurred after 100[ms], set counter to 1.
- Records as 1st interrupt time.

"Interrupt Storm Detection" feature – Detail of mechanism

- Case where the threshold is set to 1000[times/100ms].



- If the counter reaches the threshold (1000 times), detect as interrupt storm

"Interrupt Storm Detection" feature – Main features

- Kernel configs:

```
# Setting whether to enable Interrupt Storm Detection
Config  INTR_STORM_DETECT
        bool "Support interrupt storm detection"
        default n

# Setting the number of interrupts detected as interrupt storms
config INTR_STORM_DETECT_LIMIT
        int "Count considered as an interrupt storm."
        depends on INTR_STORM_DETECT
        default 100000
```

"Interrupt Storm Detection" feature – Main features

- **Setting Thresholds**
  - Can set threshold by the following command for each IRQ number.

```
# echo 20000 > /proc/irq/<IRQ number>/storm/storm_limit
```

  - How to determine threshold value:
    - Appropriate threshold values are different depending on the system
    - Must consider about the outlier value for each system.
  - To know how many times of interrupts are raised in the last 100ms:

```
# cat /proc/irq/storm_info_all
IRQ:    current_count
…(snip)…
 15:             2501             foo
…(snip)…
```

"Interrupt Storm Detection" feature – Main features

- How to debug Interrupt Storm?
  1. If interrupt storm is detected, the following message is displayed.

```
IRQ storm detect IRQ#15!
```

  2. Clarify which device driver generates the interrupt storm by /proc/interrupts.

```
# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
…(snip)
  15:      34673      33826      34696      33641          level  64 Edge   foo
```

  3. After that you can debug device driver or HW.

"Interrupt Storm Detection" feature – Other features

- Other features
  A) Disable corresponding interrupts if interrupt storm is detected.
    - System can continue to run after interrupt storm occurs.

  B) Invoke kernel panic after interrupt storm detected.
    - Stop system after interrupt storm detected.

- Notes for these features:
  - These features have a significant impact on the system.
  - Must be disabled after you identified IRQ number.

"Interrupt Storm Detection" feature – Other features

A) Disable corresponding interrupts if interrupt storm is detected

- Kernel config:

```
config INTR_STORM_DETECT_DISABLE_IRQ
        bool "Disable IRQ after interrupt storm detected"
        depends on INTR_STORM_DETECT
        default n
```

- proc interface:

```
# echo 0 or 1 > /proc/irq/<IRQ number>/storm/disable_after_detect
```

B) Invoke kernel panic after interrupt storm detected

- Kernel config:

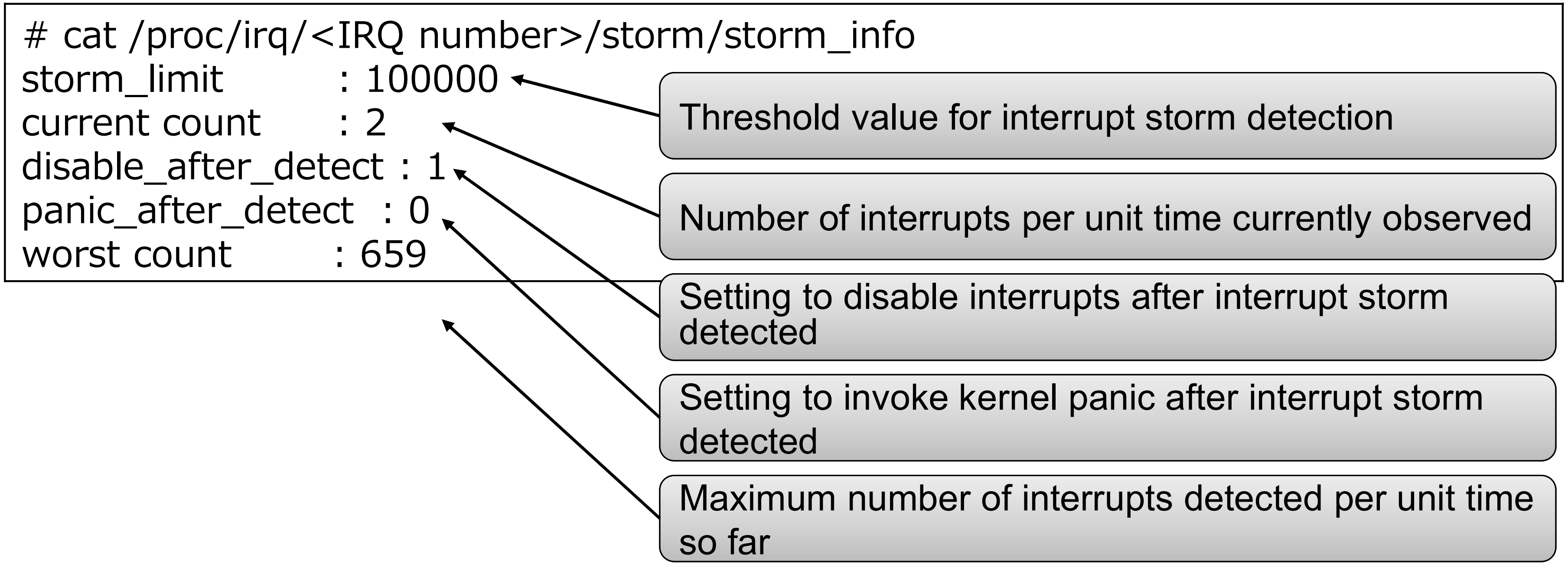
```
config INTR_STORM_DETECT_PANIC
        bool "Do panic after  interrupt storm detected"
        depends on INTR_STORM_DETECT
        default n
```

- proc interface:

```
# echo 0 or 1 > /proc/irq/<IRQ number>/storm/panic_after_detect
```

"Interrupt Storm Detection" feature – Debug info

- Debug information
  - Some useful information for each interrupt number can be shown.



```
# cat /proc/irq/<IRQ number>/storm/storm_info
storm_limit          : 100000
current count        : 2
disable_after_detect : 1
panic_after_detect   : 0
worst count          : 659
```

Threshold value for interrupt storm detection

Number of interrupts per unit time currently observed

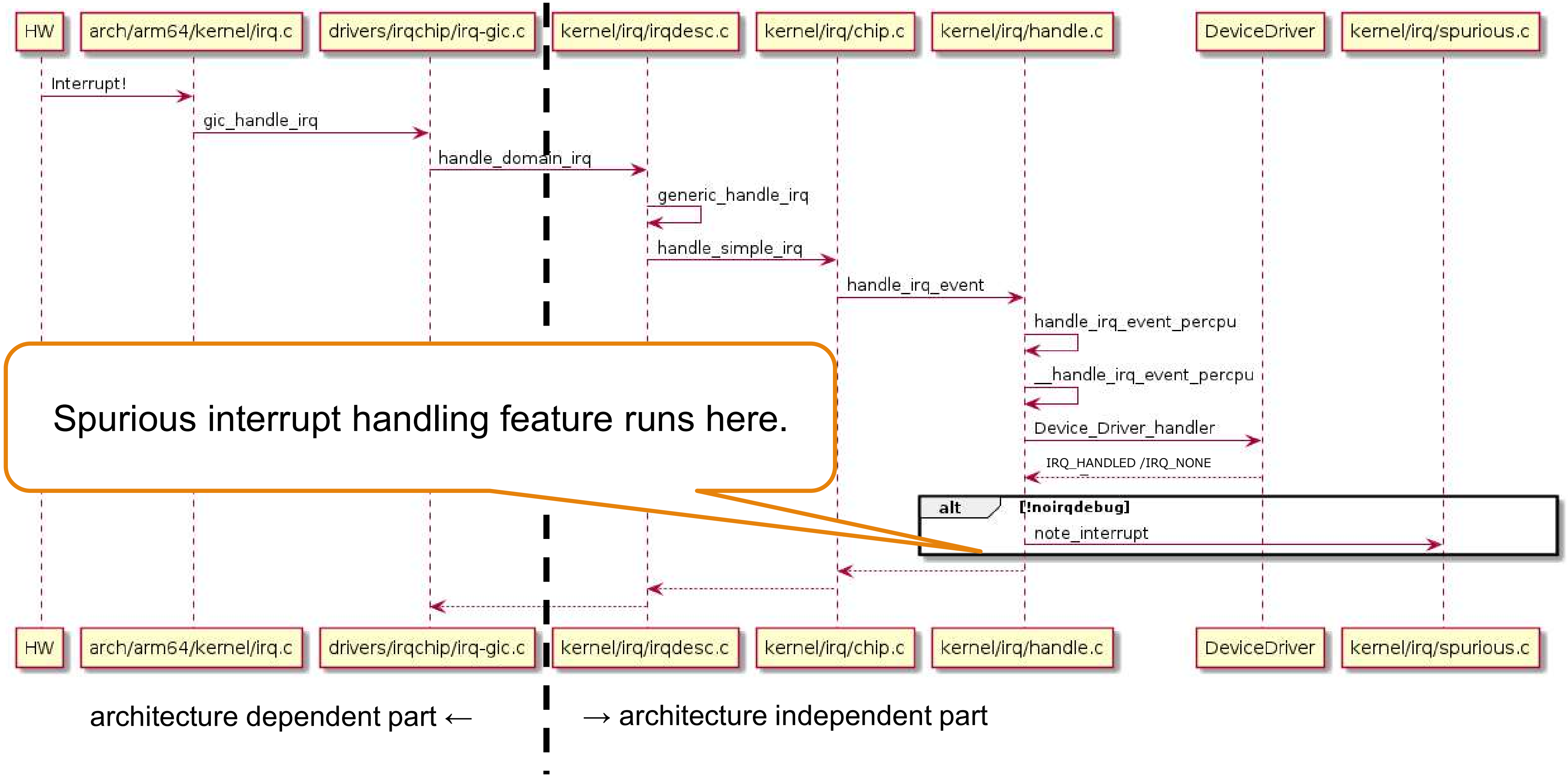
Setting to disable interrupts after interrupt storm detected

Setting to invoke kernel panic after interrupt storm detected

Maximum number of interrupts detected per unit time so far

"Interrupt Storm Detection" feature – Implementation

- Sequence of interrupts before adding functions (ARM64)



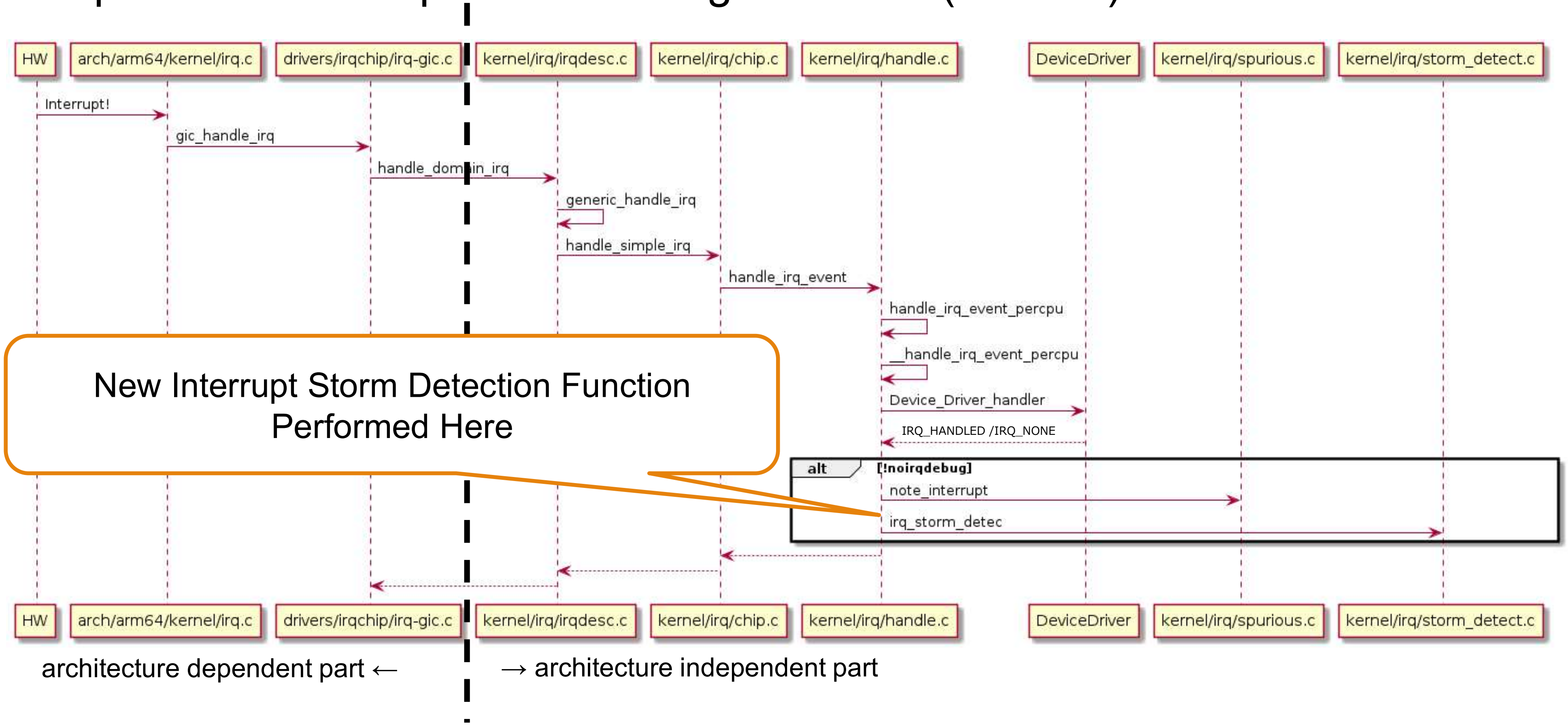
Spurious interrupt handling feature runs here.

architecture dependent part ←

→ architecture independent part

"Interrupt Storm Detection" feature – Implementation

- Sequence of interrupts after adding functions (ARM64)



New Interrupt Storm Detection Function Performed Here

architecture dependent part ←

→ architecture independent part

Actual problem caused by interrupt storm

- **Problem**
  - Exception occurred by softlockup at __do_softirq() in our development board for our products.
  - **This problem is caused by interrupt storm.**
- **How to debug this problem**
  - Debug about softlockup like follows
    1. Enable CONFIG_LOCKUP_DETECTOR and CONFIG_BOOTPARAM_SOFTLOCKUP_PANIC.
    2. Enable softlockup_panic by follow.

       ```
       # echo 1 > /proc/sys/kernel/softlockup_panic
       ```
    3. Reproduce problems.
    4. Confirm softlockup call trace.
    5. Call trace of softlockup at __do_softirq() displayed many times.

To break down problem, try to use interrupt storm detection feature.

Actual problem caused by interrupt storm

- **How to debug interrupt storm**
  1. “Interrupt Storm Detection” feature shows the following message.(Threshold setting is 10000[times/100msec])

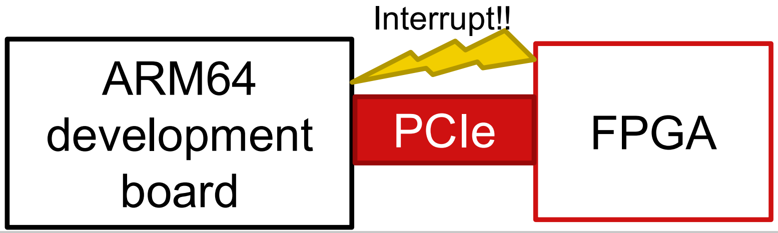
```
[ 1963.635312] IRQ storm detect IRQ#387!
```

  2. Confirm /proc/interrupts.

```
#cat /proc/interrupts
..(snip)..
387:  1  0  0  0 GICv2 104 Level PCIE, PCIe PME, aerdrv, PCIe PME, aerdrv
```

  3. Investigate PCIe device driver and hardware.

- **Cause**
  - FPGA which connected through PCIe had a problem in its firmware.



"Interrupt Storm Detection" feature – Limitations

- Can’t identify device driver which registers shared interrupt handler.
  - Only we can know IRQ number when interrupt storm is detected.

- Can’t detect interrupt handler which occupies CPU for a long time.
  - This feature only detect high-frequency interrupts.

SONY