May I interrupt? Simplest things are hardest?

Matti Vaittinen
ROHM Semiconductors
27.09.2022

Me

- Matti Vaittinen
  - Oulu, Finland
  - ROHM Semiconductors
  - Device drivers (mostly)
  - Started working with HW related SW at 2005.

Interrupt

- Device has something to tell to the Processor
- Different types (SW, MSI, GPIO...)
  - Focusing on GPIO
- Device changes line state
  - Level / Edge
- May have many reasons to do so
  - CPU reads why
- CPU “acks” device when event handled

Examples

- Device has new data (sensors, network, ...)
- Certain time has elapsed (GPS/PPS, RTC/alarm, …)
- Something went wrong (Serdes lost link, regulator abnormal voltage...)

Handling

- “HW IRQ context”
  - Sleepless working…
  - Beware locks
    - … which sleep (mutexes)
    - … which may deadlock
    - … spin_lock_irqsave() - but still beware.
  - Short!
    - Bottom halves in different context (threaded IRQs mostly)

Level active IRQs

- Device pulls line and keeps until CPU acks
  - CPU enters IRQ handling and stays there as long as line is asserted
  - CPU can disable IRQ also from CPU end
- Device pulls line and keeps until CPU acks
  - What about buses where access can sleep?

    … thread + IRQF_ONESHOT

Typical IRQ device

- Status register
  - For different reasons
- Mask register
  - To enable/disable individual reasons
- Ack register
  - To inform device which IRQs are handled and allow device to restore line

- Kuva rekistereistä tänne

Typical mistake

1. Level IRQ configured to edge.
2. Device pulls line => CPU detects edge – Ok
3. CPU reads reasons - Ok
4. CPU acks - Ok
5. Device restores line – Ok

Except… What if new IRQ is generated by device between 3 and 4?

=> Device never restores line => no more edges seen.

(Additional timers...)

Design issues I’ve seen

- Device has two IRQ pins. Status/Mask bits for both in same register.
- All IRQs acked by writing single bit / reading a reg
- Large amount of IRQ registers in device connected to slow bus
- Expect IRQs to be handled immediately

    (story about timestamp latching / SWBCN - no full story)

Tackling issue w/ large amount of regs

- Often handled using IRQ controller
  - Regmap IRQ is often your help :) (not if shotgun design)
- Hierarchical registers
  - “Main register(s)” which flags status registers with active IRQs.
  - Main register does not necessarily need ack
  - Main register bit is cleared when all active IRQs in sub register are acked
  - Supported by regmap-IRQ*

    *when main status is not acked

Debugging

- /proc/interrupts
- ftrace, irqsoff – tracer
- Oscilloscope
- Nobody cared…

- A word about shared IRQs?

- Often SW implements an irq-chip + device driver using it
- Irqchip handles status/mask/ack and provides “chained IRQs” for devices
- Use IRQ_NONE when applicable! It will save your day :)
  - Sometimes device driver handles acks/masks w/o IRQ-chip

Show Me The Code

- request_threaded_irq() / devm_request_threaded_irq()
  - Beware freeing resources prior releasing IRQ
- free_irq()

Rarely needed by drivers:

- disable_irq() / disable_irq_nosync()
  - Do not deadlock
- enable_irq()

- Irqchip code / regmap_irq