Technology Consulting Company
Research, Development &
Global Standard

Using UIO in an embedded platform

Katsuya MATSUBARA
Igel Co., Ltd
Hisao MUNAKATA
Renesas Solution Corp.

2008.4.17

Background

igel

Have you thought to
control a device from user-space?

- In fact, there are user level device drivers now.
  - Multimedia devices (graphic accelerator, etc.)
  - USB: libusb

UIO (Userspace I/O)

igel

- By Han. J. Koch
  - This interface allows the ability to write the majority of a driver **in userspace** with only very shell of a driver in the kernel itself. It uses a char device and sysfs to interact with a userspace process to **process interrupts and control memory accesses**. (Quoted from Greg Kroah-Hartman's log)
- Merged into 2.6.23

Contents

igel

- What UIO provides
- How to write a UIO driver
- How overhead of UIO

What functions UIO provides

igel

- Interrupt handling
- I/O memory access
- Continuous memory allocation

Concern about UIO usage

igel

- device dependency in application
- in-kernel resource unavailable
- Harder to share a device
- Inconstant latency
- Preemptive

Benefit for embedded

igel

- Application tight-coupled to device behavior.
  - Copy-less I/O
  - Interrupt event relay to user-space
- Minor or special device use
- Exclusive use (no need to share)
- Make kernel stable and safe.
- Easier to develop, use rich user libraries.

How to write a UIO driver

igel

- Example target device: **SuperH on-chip timer unit (TMU)**
- TMU has the following features:
  - Count down periodic counter
  - **5** channels (SH-3, SH-4)
  - Selectable base frequency
  - Interrupt when underflow
  - **The kernel uses 1 or 2 channels** for tick and high resolution timer.

Use case of TMU UIO driver

igel

- As a raw time counter for measurement
  - cf) pentium counter (rdtsc)
- As a private periodic timer without dependence on tick precision

Write a UIO kernel driver for SH TMU

igel

1. Setup and register an uio_info.
2. Catch interrupts and do time-critical process in a kernel interrupt handler.

API of UIO (in kernel)

igel

- **`struct uio_info`**
  - **`name`**: device name
  - **`version`**: device driver version
  - **`irq`**: interrupt number or UIO_IRQ_CUSTOM
  - **`irq_flags`**: flags for request_irq()
  - **`handler`**: device's irq handler (optional)
    - e.g. Make sure that the interrupt has been occurred by the device.
    - e.g. Stop the interrupt
  - **`mem[]`**: memory regions that can be mapped to user-space

API of UIO (in kernel) (contd.)

igel

- **`struct uio_mem`**
  - **`addr`**: memory address
  - **`size`**: size of memory
  - **`memtype`**: type of memory region
    - **`UIO_MEM_PHYS`**
      - I/O and physical memory
    - **`UIO_MEM_LOGICAL`**
      - Logical memory (e.g. allocated by kmalloc())
    - **`UIO_MEM_VIRTUAL`**
      - Virtual memory (e.g. allocated by vmalloc())
  - **`internal_addr`**: another address for kernel driver internal use
    - e.g. ioremap()-ed address

Code of SH TMU UIO kernel driver


```
info = kzalloc(sizeof(struct uio_info), GFP_KERNEL);

info->mem[0].size = TMU_012_SIZE;
info->mem[0].memtype = UIO_MEM_PHYS;
info->mem[0].addr = TMU_012_BASE; /* address of TMU
   registers */
info->name = "SH TMU2";
info->version = "0.01";
info->irq = TMU2_IRQ;
info->irq_flags = IRQF_DISABLED;
info->handler = sh_tmu_interrupt_handler;

uio_register_device(dev, info);
```

Code of SH TMU UIO kernel driver (contd.)

igel

```
static irqreturn_t
sh_tmu_interrupt_handler(int irq, struct uio_info *dev_info)
{
    unsigned long timer_status;

    timer_status = ctrl_inw(TMU2_TCR);
    timer_status &= ~0x100;
    ctrl_outw(timer_status, TMU2_TCR);

    return IRQ_HANDLED;
}
```

Write a UIO user driver for SH TMU

igel

1. Look for an appropriate UIO device
2. Open the UIO device
3. Mmap memory regions through the UIO device
4. Initialize the device through the mmapped memory regions.
5. (Wait for interrupts by reading the UIO device.)
6. (Handle the interrupts.)
7. Input/output data through the mmapped memory regions.

API of UIO (in user-space)

igel

- **`/sys/class/uio?/`**: information about device and UIO
  - **`name`**: UIO name
  - **`version`**: UIO version
  - **`maps/map?/`**: memory regions
    - **`addr`**: address of memory region
    - **`size`**: region size
- **`/dev/uio?`**: device access
  - **`read()`**: wait for interrupts
  - **`mmap()`**: map device memory regions to user space
    - offset = region number * PAGESIZE

Code of SH TMU UIO user driver

igel

```
fd = open("/dev/uio0", O_RDWR|O_SYNC);

/* Map device's registers into user memory */
/* fitting the memory area on pages */
offset = addr & ~PAGE_MASK;
addr = 0 /* region 0 */ * PAGE_SIZE;
size = (size + PAGE_SIZE - 1) / PAGE_SIZE * PAGE_SIZE;

iomem = mmap(0, size, PROT_READ|PROT_WRITE, MAP_SHARED,
   fd, addr);
iomem += offset;
```

Code of SH TMU UIO user driver (contd.)

igel

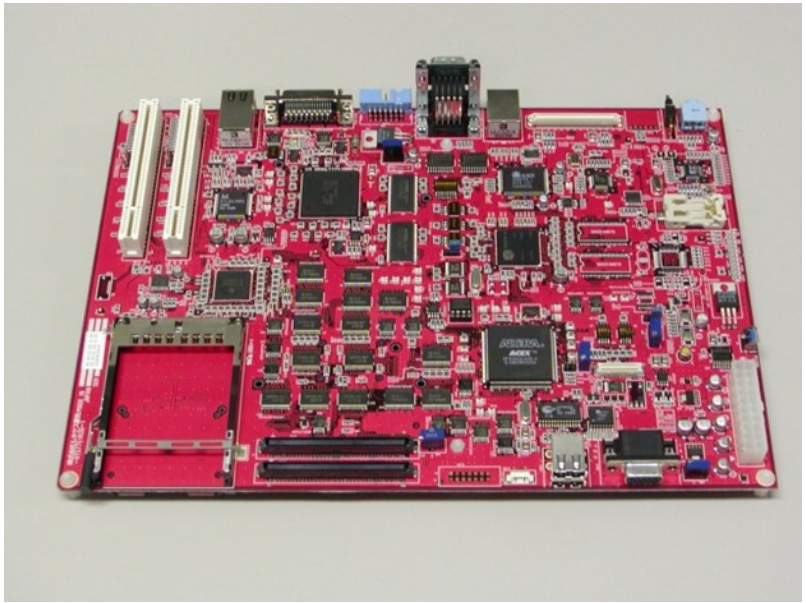
```
/* Stop the counting */
*(u_char *)SH_TMU_TSTR(iomem) |= ~(TSTR_TSTR2);
...
/* Wait for an interrupt */;
read(fd, &n_pending, sizeof(u_long));
val = *(u_int *)SH_TMU2_TCNT(iomem);
...
/* Stop the TMU */
*(u_char *)SH_TMU_TSTR(iomem) &= ~(TSTR_TSTR2);

munmap(iomem, size);
close(fd);
```

Measurement

igel

- Renesas RTS7751R2D board
  - SH7751R(SH-4 architecture) 240MHz
  - 256MB RAM
  - 5channels of TMU
  - NFS rootfs
- Base software
  - Linux-2.6.25-rc9
  - UIO driver for SH TMU2



UIO Overhead

igel

- Latency in UIO interrupt handling (preliminary results)

load	MIN(us)	MAX(us)	AVERAGE(us)
none	60	89	64
ping -f	60	199	147
make vmlinux	104	203	126

  - The latency depends on scheduler.
    - SCHED_FIFO, SCHED_RR priority
    - Realtime preemption patch (CONFIG_PREEMPT_RT)
- Read the counter value with less overhead.

FYI: Backport for older kernels

igel

- Not so hard.
- small code (820 lines in uio.c and uio_driver.h)
- use a few legacy framework (interrupt handling, device file, and sysfs)
- e.g. backport into 2.6.16
  - Change arguments of interrupt handler
  - Replace device_create() and device_destroy() invocations with appropriate code.

FYI: Recent changes

igel

- Some fixes (in 2.6.25)
  - Cache off in physical memory mapping
- Another UIO driver (in the GregKH tree)
  - SMX Cryptengine

Conclusion

igel

- UIO provides for user-space
  - Low overhead device access
  - Continuous physical memory allocation
  - Interrupt handling
- UIO is useful for embedded systems
  - Minor or special device
  - Exclusive use
  - Make applications closer to device

I hope that after my presentation,
you will want to try using UIO.