Hardware GPIO-IRQ

ODROID-C2 supports the GPIO interrupt with Edge triggers.
The maximum number of interrupt is limited to 8 when you use the falling(Low) or rising(High) edge trigger.
If you use the both (rising and falling together) edge triggers, the maximum number is only 4.

Please note that SD card driver is using 2 interrupts to implement the card detection (insertion/remove) by default.
So **we can use only 6 GPIO IRQs**.
If you use an eMMC without SD card, you can access full 8 GPIO IRQs once you disable the SD functionality.

You must use the **Linux Kernel 3.14.29-56 Apr** or higher to use the GPIO interrupt in the user-land.

Usage

To use the GPIO Interrupt in the User space, you must export the GPIO and add the Trigger option in Edge parameter.

GPIO SYSFS node is accessible via **/sys/class/gpio/**.

GPIO Export

echo [GPIO Number(Refer **Expansion Connectors**)] > /sys/class/gpio/export

e.g) GPIOX.BIT21 Export (Pin Number 7)

```
echo 249 > /sys/class/gpio/export
```

User Space Example code

This C example code shows the status of GPIO interrupts in the User Space.
This example uses Both(rising + falling) edges on the GPIO Pin 7.

Build

```
gcc -o aml_gpio_irq_test aml_gpio_irq_test.c
```

Run

“Usage: aml_gpio_irq [GPIO Number]”

```
sudo aml_gpio_irq_test 249
```

Code

aml_gpio_irq.c

```
//[*]--------------------------------------------------------------------
-------[*]
//
//  GPIO IRQ Test Application for AMLogic.
//
//[*]--------------------------------------------------------------------
-------[*]
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>
#include <sched.h>

//[*]--------------------------------------------------------------------
-------[*]
#define SYSFS_GPIO_DIR  "/sys/class/gpio"
#define POLL_TIMEOUT    (3 * 1000) /* 3 seconds */
#define MAX_BUF         64

//[*]--------------------------------------------------------------------
-------[*]
int gpio_export(unsigned int gpio)
{
    int fd, len;
    char buf[MAX_BUF];

    fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
    if (fd < ) {
        perror("gpio/export");
        return fd;
    }

    len = snprintf(buf, sizeof(buf), "%d", gpio);
    write(fd, buf, len);
    close(fd);

    return ;
}
```

```
//[*]---------------------------------------------------------------------
-------[*]
int gpio_unexport(unsigned int gpio)
{
    int fd, len;
    char buf[MAX_BUF];

    fd = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
    if (fd < ) {
        perror("gpio/export");
        return fd;
    }

    len = snprintf(buf, sizeof(buf), "%d", gpio);
    write(fd, buf, len);
    close(fd);
    return ;
}

//[*]---------------------------------------------------------------------
-------[*]
int gpio_set_dir(unsigned int gpio, unsigned int out_flag)
{
    int fd, len;
    char buf[MAX_BUF];

    len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR
"/gpio%d/direction", gpio);

    fd = open(buf, O_WRONLY);
    if (fd < ) {
        perror("gpio/direction");
        return fd;
    }

    if (out_flag)
        write(fd, "out", 4);
    else
        write(fd, "in", 3);

    close(fd);
    return ;
}

//[*]---------------------------------------------------------------------
-------[*]
int gpio_set_value(unsigned int gpio, unsigned int value)
{
    int fd, len;
    char buf[MAX_BUF];
```

```
    len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value",
gpio);

    fd = open(buf, O_WRONLY);
    if (fd < ) {
        perror("gpio/set-value");
        return fd;
    }

    if (value)
        write(fd, "1", 2);
    else
        write(fd, "0", 2);

    close(fd);
    return ;
}

//[*]--------------------------------------------------------------------
-------[*]
int gpio_get_value(unsigned int gpio, unsigned int *value)
{
    int fd, len;
    char buf[MAX_BUF];
    char ch;

    len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value",
gpio);

    fd = open(buf, O_RDONLY);
    if (fd < ) {
        perror("gpio/get-value");
        return fd;
    }

    read(fd, &ch, 1);

    if (ch != '0') {
        *value = 1;
    } else {
        *value = ;
    }

    close(fd);
    return ;
}

//[*]--------------------------------------------------------------------
-------[*]
int gpio_set_edge(unsigned int gpio, char *edge)
```

```
{
    int fd, len;
    char buf[MAX_BUF];

    len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/edge",
gpio);

    fd = open(buf, O_WRONLY);
    if (fd < ) {
        perror("gpio/set-edge");
        return fd;
    }

    write(fd, edge, strlen(edge) + 1);
    close(fd);
    return ;
}

//[*]---------------------------------------------------------------------
-------[*]
int gpio_fd_open(unsigned int gpio)
{
    int fd, len;
    char buf[MAX_BUF];

    len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value",
gpio);

    fd = open(buf, O_RDONLY | O_NONBLOCK );
    if (fd < ) {
        perror("gpio/fd_open");
    }
    return fd;
}

//[*]---------------------------------------------------------------------
-------[*]
int gpio_fd_close(int fd)
{
    return close(fd);
}

//[*]---------------------------------------------------------------------
-------[*]
int main(int argc, char **argv, char **envp)
{
    struct pollfd fdset[2];
    int nfds = 2;
    int gpio_fd, timeout, rc;
    char *buf[MAX_BUF];
    unsigned int gpio;
```

```
    int len;

    if (argc < 2) {
        printf("Usage: aml_gpio_irq <gpio_pin>\n\n");
        printf("Waits for a change in the GPIO pin voltage level or
input on stdin\n");
        exit(-1);
    }

    gpio = atoi(argv[1]);

    gpio_export(gpio);
    gpio_set_dir(gpio, );
    gpio_set_edge(gpio, "both");    // High/Low Edge Trigger
    gpio_fd = gpio_fd_open(gpio);

    timeout = POLL_TIMEOUT;

    // set a high priority schedulling for the running program
    {
        struct sched_param sched ;

        memset (&sched, , sizeof(sched)) ;

        sched.sched_priority = 55;

        sched_setscheduler (, SCHED_RR, &sched) ;
    }

    while (1) {
        memset((void*)fdset, , sizeof(fdset));

        fdset[].fd = STDIN_FILENO;
        fdset[].events = POLLIN;

        fdset[1].fd = gpio_fd;
        fdset[1].events = POLLPRI;

        rc = poll(fdset, nfds, timeout);

        if (rc < ) {
            printf("\npoll() failed!\n");
            return -1;
        }

        if (rc == ) {
            printf("POLL Timeout!\n");
        }

        if (fdset[1].revents & POLLPRI) {
            len = read(fdset[1].fd, buf, MAX_BUF);
```

```
            printf("\npoll() GPIO %d interrupt occurred\n", gpio);
        }

        if (fdset[].revents & POLLIN) {
            (void)read(fdset[].fd, buf, 1);
            printf("\npoll() stdin read 0x%2.2X\n", (unsigned int)
buf[]);
        }

        fflush(stdout);
    }

    gpio_fd_close(gpio_fd);

    return ;
}

//[*]---------------------------------------------------------------------
-------[*]
//[*]---------------------------------------------------------------------
-------[*]
```

Kernel driver GPIO IRQ Example code by crashoverride

HC-SR04 kernel driver