Gpio-int-test.c

```
/* Copyright (c) 2011, RidgeRun
 * All rights reserved.
 *
 * Contributors include:
 *   Todd Fischer
 *   Brad Lu
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    This product includes software developed by the RidgeRun.
 * 4. Neither the name of the RidgeRun nor the
 *    names of its contributors may be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY RIDGERUN ''AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL RIDGERUN BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>

 /****************************************************************
 * Constants
 ****************************************************************/

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

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

        fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
        if (fd < 0) {
                perror("gpio/export");
                return fd;
        }
```

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

        return 0;
}

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

        fd = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
        if (fd < 0) {
                perror("gpio/export");
                return fd;
        }

        len = snprintf(buf, sizeof(buf), "%d", gpio);
        write(fd, buf, len);
        close(fd);
        return 0;
}

/****************************************************************
 * gpio_set_dir
 ****************************************************************/
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 < 0) {
                perror("gpio/direction");
                return fd;
        }

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

        close(fd);
        return 0;
}

/****************************************************************
 * gpio_set_value
 ****************************************************************/
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 < 0) {
                perror("gpio/set-value");
                return fd;
        }

        if (value)
```

```
                write(fd, "1", 2);
        else
                write(fd, "0", 2);

        close(fd);
        return 0;
}

/****************************************************************
 * gpio_get_value
 ****************************************************************/
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 < 0) {
                perror("gpio/get-value");
                return fd;
        }

        read(fd, &ch, 1);

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

        close(fd);
        return 0;
}


/****************************************************************
 * gpio_set_edge
 ****************************************************************/

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 < 0) {
                perror("gpio/set-edge");
                return fd;
        }

        write(fd, edge, strlen(edge) + 1);
        close(fd);
        return 0;
}

/****************************************************************
 * gpio_fd_open
 ****************************************************************/

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 < 0) {
		perror("gpio/fd_open");
	}
	return fd;
}

/****************************************************************
 * gpio_fd_close
 ****************************************************************/

int gpio_fd_close(int fd)
{
	return close(fd);
}

/****************************************************************
 * Main
 ****************************************************************/
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: gpio-int <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, 0);
	gpio_set_edge(gpio, "rising");
	gpio_fd = gpio_fd_open(gpio);

	timeout = POLL_TIMEOUT;

	while (1) {
		memset((void*)fdset, 0, sizeof(fdset));

		fdset[0].fd = STDIN_FILENO;
		fdset[0].events = POLLIN;

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

		rc = poll(fdset, nfds, timeout);

		if (rc < 0) {
			printf("\npoll() failed!\n");
			return -1;
		}

		if (rc == 0) {
			printf(".");
		}

		if (fdset[1].revents & POLLPRI) {
			lseek(fdset[1].fd, 0, SEEK_SET);
			len = read(fdset[1].fd, buf, MAX_BUF);
			printf("\npoll() GPIO %d interrupt occurred\n", gpio);
```

```
                }

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

                fflush(stdout);
        }

        gpio_fd_close(gpio_fd);
        return 0;
}
```