Interfacing with I2C Devices

From eLinux.org

This page is meant to provide some basic information about how to interface with I²C (http://en.wikipedia.org/wiki/I%C2%B2C) devices through the /dev/i2c interface. The I²C bus is commonly used to connect relatively low-speed sensors and other peripherals to equipment varying in complexity from a simple microcontroller to a full-on motherboard. I²C is extremely popular due to its ease-of-use and ability to control multiple peripherals while utilizing only two pins on the host controller. Although I²C has a variety of modes, this page will deal purely with communication between a Linux-based master and a slave peripheral for the time being.

Contents

Code Examples

Note: Examples shown on this page were developed based on a Texas Instruments BeagleBoard and some changes *will* be required depending on the system being utilized.

Beagleboard I2C2 Enable

The TI BeagleBoard has 3 I²C buses available, which control a variety of on-board peripherals, including the DVI display driver and power sequencing. As bus 2 is by far the easiest to connect to due to its presence on the 0.1" spaced expansion header, it is assumed the user will be controlling that bus for the

purposes of this example. However, by default bus 2 is disabled due to a lack of pull-up resistors on the board, so external pull-ups to 1.8V must be added and the kernel recompiled to enable i2c2. It is important to remember that this bus runs at 1.8V on the Beagleboard and that external level shifters will be required to interface with 3.3V or 5V devices.

See BeagleBoard#Linux_kernel and BeagleBoardLinuxKernel for details on how to recompile your Linux kernel. To enable i2c2 specifically during that process, enable the setting during the "make menuconfig" step.

It is important to note that the method of enabling i2c2 varies depending on your kernel and applied patches. Recent kernel versions have changed how PIN_MUX settings are set. For me, running the openembedded stable/2009 kernel (2.6.29-r46) was enough. By default i2c2 was properly configured and enabled. A fast way to check is to see if /dev/i2c-2 exists (But this does not check proper mux settings).

I2C2 is pinned out as pins 23 (SDA) and 24 (SCL) on the expansion header.

i2c-tools

i2c-tools (http://www.lm-sensors.org/wiki/I2CTools) is a set of I²C programs that make it easy to debug I²C devices without having to write any code. For example, with the BeagleBoard:

```
$i2cdetect -r 2
```

Will send out read byte commands on the /dev/i2c-2 line to probe for addresses, and return any devices found. This is useful for checking what devices are functioning properly. (Note: the -r flag may interfere with write-only devices, but the default probing method does not work on the Beagle.) i2cget (http://www.lm-sensors.org/wiki/man/i2cget) and i2cset (http://www.lm-sensors.org/wiki/man/i2cset) write and read to devices respectively.

Basic Communication with the AD7991 (http://elinux.org/Interfacing_with_I%C2%B2C_Devices#Code_Examples) ADC (http://en.wikipedia.org/wiki/Analog-to-digital_converter)

The AD7991 has four inputs as well as the ability to use one of the input pins as a reference voltage the other inputs are measured against. If that input is not used as the reference voltage, it uses the supply voltage as the reference voltage. The power on default configuration uses all 4 channels as inputs, so in this case no further configuration is necessary. Because the AD7991 is 12-bit device, its outputs ranges linearly from 0 to 4096 as the voltage ranges from 0

to the reference voltage.

There are multiple ways to communicate with I²C devices, including the writing of a full kernel driver. This adds significant extra complexity, however, as basic bus control can be accomplished with the open, ioctl, read, and write commands.

Opening the Bus

In order to communicate with an I²C peripheral with this simple structure, you must first open the bus for reading and writing like you would any file. A call to open (http://en.wikipedia.org/wiki/Open_(system_call)) must be used rather than fopen so that writes to the bus are not buffered. Open returns a new file descriptor (a non-negative integer) which can then be used to configure the bus. A typical reason for failure at this stage is a lack of permissions to access /dev/i2c-2. Adding the user to a group which has permissions to access the file will alleviate this problem, as will adjusting the file permissions to enable user access. Adding a udev rule to set the I²C device group (http://xgoat.com /wp/2008/01/29/i2c-device-udev-rule/) is the most permanent solution.

```
int file;
char *filename = "/dev/i2c-2";
if ((file = open(filename, O_RDWR)) < 0) {
    /* ERROR HANDLING: you can check errno to see what went wrong */
    perror("Failed to open the i2c bus");
    exit(1);
}
```

Initiating communication with the AD7991

After successfully acquiring bus access, you must initiate communication with whatever peripheral you are attempting to utilize. I²C does this by sending out the seven bit address of the device followed by a read/write bit. The bit is set to 0 for writes and 1 for reads. This is another common failure point, as manufacturers tend to report the I²C address of the peripheral in a variety of ways. Some report the address as a seven bit number, meaning that the address must be shifted left by a bit and then have the r/w bit tacked onto the end. Others will provide it as an eight bit number and assume you will set the last bit accordingly. Although a few manufacturers actually say which method they use to describe the address, the vast majority do not, and the user may have to resort to testing via trial and error.

The AD7991 used in this example is the AD7991-1, which has an address reported by the datasheet as 0101001. To use this properly, zero pad the address on the left and store it as 0b00101001. The calls to read and write after the ioctl will automatically set the proper read and write bit when signaling the peripheral.

```
int addr = 0b00101001;          // The I2C address of the ADC
if (ioctl(file, I2C_SLAVE, addr) < 0) {
```

```
    printf("Failed to acquire bus access and/or talk to slave.\n");
    /* ERROR HANDLING; you can check errno to see what went wrong */
    exit(1);
}
```

Reading from the ADC

The read system call is used to obtain data from the I²C peripheral. Read requires a file handle, a buffer to store the data, and a number of bytes to read. Read will attempt to read the number of bytes specified and will return the actual number of bytes read, which can be used to detect errors.

The code in the else block below calculates the voltage present at the ADC pin assuming a 5 volt reference/supply voltage. The AD7991 samples a 12 bit value, which is read back as two eight bit values. Rather than waste the empty bits, two of them are also used to signal the channel data being sent. See page 21 of the datasheet for more details.

```
char buf[10] = {0};
float data;
char channel;

for (int i = 0; i<4; i++) {
    // Using I2C Read
    if (read(file,buf,2) != 2) {
        /* ERROR HANDLING: i2c transaction failed */
        printf("Failed to read from the i2c bus.\n");
        buffer = g_strerror(errno);
        printf(buffer);
        printf("\n\n");
    } else {
        data = (float)((buf[0] & 0b00001111)<<8)+buf[1];
        data = data/4096*5;
        channel = ((buf[0] & 0b00110000)>>4);
        printf("Channel %02d Data:  %04f\n",channel,data);
    }
}
```

Writing to the ADC

The write system call is used to obtain data from the I²C peripheral. Write requires a file handle, a buffer in which the data is stored, and a number of bytes to write. Write will attempt to write the number of bytes specified and will return the actual number of bytes written, which can be used to detect errors.

Some devices require an internal address to be sent prior to the data to specify the register on the external device to access. The AD7991 contains only one configuration register, and as such, does not require a internal register selection address to be sent to the device. For devices with more than one configuration register, the address of the register should be written first, followed by the data to be placed there. See the datasheet specific to the part for more details.

```
//unsigned char reg = 0x10; // Device register to access
//buf[0] = reg;

buf[0] = 0b11110000;
if (write(file,buf,1) != 1) {
    /* ERROR HANDLING: i2c transaction failed */
    printf("Failed to write to the i2c bus.\n");
    buffer = g_strerror(errno);
    printf(buffer);
    printf("\n\n");
}
```

Detecting Errors

Errors on the I²C bus range from incorrect permissions and addressing to hardware errors that may simply not allow peripherals to respond. The errno interface is used to identify these errors using the glib function g_strerror. For simple usage, calling the function as above will work. See the glib api for more details.

Completed Code

```
#include <glib.h>
#include <glib/gprintf.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void
sensors_ADC_init(void) {
    int file;
    char filename[40];
    const gchar *buffer;
    int addr = 0b00101001;        // The I2C address of the ADC

    sprintf(filename,"/dev/i2c-2");
    if ((file = open(filename,O_RDWR)) < 0) {
        printf("Failed to open the bus.");
        /* ERROR HANDLING; you can check errno to see what went wrong */
        exit(1);
    }

    if (ioctl(file,I2C_SLAVE,addr) < 0) {
        printf("Failed to acquire bus access and/or talk to slave.\n");
        /* ERROR HANDLING; you can check errno to see what went wrong */
        exit(1);
    }

    char buf[10] = {0};
    float data;
    char channel;

    for(int i = 0; i<4; i++) {
        // Using I2C Read
        if (read(file,buf,2) != 2) {
```

```
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus.\n");
            buffer = g_strerror(errno);
            printf(buffer);
            printf("\n\n");
        } else {
            data = (float)((buf[0] & 0b00001111)<<8)+buf[1];
            data = data/4096*5;
            channel = ((buf[0] & 0b00110000)>>4);
            printf("Channel %02d Data:  %04f\n",channel,data);
        }
    }

    //unsigned char reg = 0x10; // Device register to access
    //buf[0] = reg;
    buf[0] = 0b11110000;

    if (write(file,buf,1) != 1) {
        /* ERROR HANDLING: i2c transaction failed */
        printf("Failed to write to the i2c bus.\n");
        buffer = g_strerror(errno);
        printf(buffer);
        printf("\n\n");
    }
}
```

Cross Compilation

In order to compile the code and create a binary that can be run on the embedded system, you will need a cross compiler (e.g. ARMCompilers) on the host computer and added to the PATH.

For example, using the CodeSourcery ARM toolchain:

```
$ gcc CROSS-COMPILE=arm-none-linux-gnueabi ARCH=arm i2c_interface.c -o i2c_binary
```

The resulting binary can then be moved to the embedded device and executed.

- If you get a warning about I2C_SLAVE not being defined, you may need to include both <linux/i2c.h> and <linux/i2c-dev.h> (The location has changed in newer kernels vs. older kernels and the above example is for newer)

Tested Devices

- AD7991 Quad Input ADC (http://www.analog.com/en/analog-to-digital-converters/ad-converters/ad7991/products/product.html)
  - Utilized with Sharp GP2D12 IR Range Sensors (http://www.acroname.com/robotics/parts/SharpGP2D12-15.pdf). (That link is broken. Here (http://www.acroname.com/robotics/parts/c_Sensors.html) is a list of sensors from that site.
- SRF08 Ultrasonic Range Sensors (http://www.robot-electronics.co.uk/htm/srf08tech.shtml)
- ADXL345 3-axis Accelerometer (http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf)

External Links

- Wikipedia I²C Page (http://en.wikipedia.org/wiki/I%C2%B2C)
- Linux Kernel I2C Documentation (http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/i2c/dev-interface)
- Standard Linux Syscalls (http://linux.die.net/man/2/syscalls)
- Presentation on I2C, PWM and Hardware interfacing with the BeagleBoard

Categories: Robotics | BeagleBoard
