# Linux I2C Sysfs

## Overview

I2C topology can be complex because of the existence of I2C MUX (I2C Multiplexer). The Linux kernel abstracts the MUX channels into logical I2C bus numbers. However, there is a gap of knowledge to map from the I2C bus physical number and MUX topology to logical I2C bus number. This doc is aimed to fill in this gap, so the audience (hardware engineers and new software developers for example) can learn the concept of logical I2C buses in the kernel, by knowing the physical I2C topology and navigating through the I2C sysfs in Linux shell. This knowledge is useful and essential to use `i2c-tools` for the purpose of development and debugging.

### Target audience

People who need to use Linux shell to interact with I2C subsystem on a system which the Linux is running on.

### Prerequisites

1. Knowledge of general Linux shell file system commands and operations.
2. General knowledge of I2C, I2C MUX and I2C topology.

## Location of I2C Sysfs

Typically, the Linux Sysfs filesystem is mounted at the `/sys` directory, so you can find the I2C Sysfs under `/sys/bus/i2c/devices` where you can directly `cd` to it. There is a list of symbolic links under that directory. The links that start with `i2c-` are I2C buses, which may be either physical or logical. The other links that begin with numbers and end with numbers are I2C devices, where the first number is I2C bus number, and the second number is I2C address.

Google Pixel 3 phone for example:

```
blueline:/sys/bus/i2c/devices $ ls
0-0008  0-0061  1-0028  3-0043  4-0036  4-0041  i2c-1  i2c-3
0-000c  0-0066  2-0049  4-000b  4-0040  i2c-0   i2c-2  i2c-4
```

`i2c-2` is an I2C bus whose number is 2, and `2-0049` is an I2C device on bus 2 address 0x49 bound with a kernel driver.

## Terminologies

First, let us define a couple of terminologies to avoid confusions in the later sections.

### (Physical) I2C Bus Controller

The hardware system that the Linux kernel is running on may have multiple physical I2C bus controllers. The controllers are hardware and physical, and the system may define multiple registers in the memory space to manipulate the controllers. Linux kernel has I2C bus drivers under source directory `drivers/i2c/busses` to translate kernel I2C API into register operations for different systems. This terminology is not limited to Linux kernel only.

### I2C Bus Physical Number

For each physical I2C bus controller, the system vendor may assign a physical number to each controller. For example, the first I2C bus controller which has the lowest register addresses may be called `I2C-0`.

### Logical I2C Bus

Every I2C bus number you see in Linux I2C Sysfs is a logical I2C bus with a number assigned. This is similar to the fact that software code is usually written upon virtual memory space, instead of physical memory space.

Each logical I2C bus may be an abstraction of a physical I2C bus controller, or an abstraction of a channel behind an I2C MUX. In case it is an abstraction of a MUX channel, whenever we access an I2C device via a such logical bus, the kernel will switch the I2C MUX for you to the proper channel as part of the abstraction.

### Physical I2C Bus

If the logical I2C bus is a direct abstraction of a physical I2C bus controller, let us call it a physical I2C bus.

### Caveat

This may be a confusing part for people who only know about the physical I2C design of a board. It is actually possible to rename the I2C bus physical number to a different number in logical I2C bus level in Device Tree Source (DTS) under section `aliases`. See arch/arm/boot/dts/nuvoton-npcm730-gsj.dts for an example of DTS file.

Best Practice: **(To kernel software developers)** It is better to keep the I2C bus physical number the same as their corresponding logical I2C bus number, instead of renaming or mapping them, so that it may be less confusing to other users. These physical I2C buses can be served as good starting points for I2C MUX fanouts. For the following examples, we will assume that the physical I2C bus has a number same as their I2C bus physical number.

## Walk through Logical I2C Bus

For the following content, we will use a more complex I2C topology as an example. Here is a brief graph for the I2C topology. If you do not understand this graph at the first glance, do not be afraid to continue reading this doc and review it when you finish reading.

```
i2c-7 (physical I2C bus controller 7)
`-- 7-0071 (4-channel I2C MUX at 0x71)
    |-- i2c-60 (channel-0)
    |-- i2c-73 (channel-1)
    |   |-- 73-0040 (I2C sensor device with hwmon directory)
    |   |-- 73-0070 (I2C MUX at 0x70, exists in DTS, but failed to probe)
    |   `-- 73-0072 (8-channel I2C MUX at 0x72)
    |       |-- i2c-78 (channel-0)
    |       |-- ... (channel-1...6, i2c-79...i2c-84)
    |       `-- i2c-85 (channel-7)
    |-- i2c-86 (channel-2)
    `-- i2c-203 (channel-3)
```

### Distinguish Physical and Logical I2C Bus

One simple way to distinguish between a physical I2C bus and a logical I2C bus, is to read the symbolic link `device` under the I2C bus directory by using command `ls -l` or `readlink`.

An alternative symbolic link to check is `mux_device`. This link only exists in logical I2C bus directory which is fanned out from another I2C bus. Reading this link will also tell you which I2C MUX device created this logical I2C bus.

If the symbolic link points to a directory ending with `.i2c`, it should be a physical I2C bus, directly abstracting a physical I2C bus controller. For example:

```
$ readlink /sys/bus/i2c/devices/i2c-7/device
../../f0087000.i2c
$ ls /sys/bus/i2c/devices/i2c-7/mux_device
ls: /sys/bus/i2c/devices/i2c-7/mux_device: No such file or directory
```

In this case, `i2c-7` is a physical I2C bus, so it does not have the symbolic link `mux_device` under its directory. And if the kernel software developer follows the common practice by not renaming physical I2C buses, this should also mean the physical I2C bus controller 7 of the system.

On the other hand, if the symbolic link points to another I2C bus, the I2C bus presented by the current directory has to be a logical bus. The I2C bus pointed by the link is the parent bus which may be either a physical I2C bus or a logical one. In this case, the I2C bus presented by the current directory abstracts an I2C MUX channel under the parent bus.

For example:

```
$ readlink /sys/bus/i2c/devices/i2c-73/device
../../i2c-7
$ readlink /sys/bus/i2c/devices/i2c-73/mux_device
../7-0071
```

`i2c-73` is a logical bus fanout by an I2C MUX under `i2c-7` whose I2C address is 0x71. Whenever we access an I2C device with bus 73, the kernel will always switch the I2C MUX addressed 0x71 to the proper channel for you as part of the abstraction.

### Finding out Logical I2C Bus Number

In this section, we will describe how to find out the logical I2C bus number representing certain I2C MUX channels based on the knowledge of physical hardware I2C topology.

In this example, we have a system which has a physical I2C bus 7 and not renamed in DTS. There is a 4-channel MUX at address 0x71 on that bus. There is another 8-channel MUX at address 0x72 behind the channel 1 of the 0x71 MUX. Let us navigate through Sysfs and find out the logical I2C bus number of the channel 3 of the 0x72 MUX.

First of all, let us go to the directory of `i2c-7`:

```
~$ cd /sys/bus/i2c/devices/i2c-7
/sys/bus/i2c/devices/i2c-7$ ls
7-0071         i2c-60         name           subsystem
delete_device  i2c-73         new_device     uevent
device         i2c-86         of_node
i2c-203        i2c-dev        power
```

There, we see the 0x71 MUX as `7-0071`. Go inside it:

```
/sys/bus/i2c/devices/i2c-7$ cd 7-0071/
/sys/bus/i2c/devices/i2c-7/7-0071$ ls -l
channel-0   channel-3   modalias    power
channel-1   driver      name        subsystem
channel-2   idle_state  of_node     uevent
```

Read the link `channel-1` using `readlink` or `ls -l`:

```
/sys/bus/i2c/devices/i2c-7/7-0071$ readlink channel-1
../i2c-73
```

We find out that the channel 1 of 0x71 MUX on `i2c-7` is assigned with a logical I2C bus number of 73. Let us continue the journey to directory `i2c-73` in either ways:

```
# cd to i2c-73 under I2C Sysfs root
/sys/bus/i2c/devices/i2c-7/7-0071$ cd /sys/bus/i2c/devices/i2c-73
/sys/bus/i2c/devices/i2c-73$

# cd the channel symbolic link
/sys/bus/i2c/devices/i2c-7/7-0071$ cd channel-1
/sys/bus/i2c/devices/i2c-7/7-0071/channel-1$

# cd the link content
/sys/bus/i2c/devices/i2c-7/7-0071$ cd ../i2c-73
/sys/bus/i2c/devices/i2c-7/i2c-73$
```

Either ways, you will end up in the directory of `i2c-73`. Similar to above, we can now find the 0x72 MUX and what logical I2C bus numbers that its channels are assigned:

```
/sys/bus/i2c/devices/i2c-73$ ls
73-0040         device          i2c-83          new_device
73-004e         i2c-78          i2c-84          of_node
73-0050         i2c-79          i2c-85          power
73-0070         i2c-80          i2c-dev         subsystem
73-0072         i2c-81          mux_device      uevent
delete_device   i2c-82          name
/sys/bus/i2c/devices/i2c-73$ cd 73-0072
/sys/bus/i2c/devices/i2c-73/73-0072$ ls
channel-0   channel-4   driver      of_node
channel-1   channel-5   idle_state  power
channel-2   channel-6   modalias    subsystem
channel-3   channel-7   name        uevent
/sys/bus/i2c/devices/i2c-73/73-0072$ readlink channel-3
../i2c-81
```

There, we find out the logical I2C bus number of the channel 3 of the 0x72 MUX is 81. We can later use this number to switch to its own I2C Sysfs directory or issue `i2c-tools` commands.

Tip: Once you understand the I2C topology with MUX, command i2cdetect -l in I2C Tools can give you an overview of the I2C topology easily, if it is available on your system. For example:

```
$ i2cdetect -l | grep -e '\-73' -e _7 | sort -V
i2c-7   i2c             npcm_i2c_7                      I2C adapter
i2c-73  i2c             i2c-7-mux (chan_id 1)           I2C adapter
i2c-78  i2c             i2c-73-mux (chan_id 0)          I2C adapter
i2c-79  i2c             i2c-73-mux (chan_id 1)          I2C adapter
i2c-80  i2c             i2c-73-mux (chan_id 2)          I2C adapter
i2c-81  i2c             i2c-73-mux (chan_id 3)          I2C adapter
i2c-82  i2c             i2c-73-mux (chan_id 4)          I2C adapter
i2c-83  i2c             i2c-73-mux (chan_id 5)          I2C adapter
i2c-84  i2c             i2c-73-mux (chan_id 6)          I2C adapter
i2c-85  i2c             i2c-73-mux (chan_id 7)          I2C adapter
```

## Pinned Logical I2C Bus Number

If not specified in DTS, when an I2C MUX driver is applied and the MUX device is successfully probed, the kernel will assign the MUX channels with a logical bus number based on the current biggest logical bus number incrementally. For example, if the system has `i2c-15` as the highest logical bus number, and a 4-channel MUX is applied successfully, we will have `i2c-16` for the MUX channel 0, and all the way to `i2c-19` for the MUX channel 3.

The kernel software developer is able to pin the fanout MUX channels to a static logical I2C bus number in the DTS. This doc will not go through the details on how to implement this in DTS, but we can see an example in: arch/arm/boot/dts/aspeed-bmc-facebook-wedge400.dts

In the above example, there is an 8-channel I2C MUX at address 0x70 on physical I2C bus 2. The channel 2 of the MUX is defined as `imux18` in DTS, and pinned to logical I2C bus number 18 with the line of `i2c18 = &imux18;` in section `aliases`.

Take it further, it is possible to design a logical I2C bus number schema that can be easily remembered by humans or calculated arithmetically. For example, we can pin the fanout channels of a MUX on bus 3 to start at 30. So 30 will be the logical bus number of the channel 0 of the MUX on bus 3, and 37 will be the logical bus number of the channel 7 of the MUX on bus 3.

## I2C Devices

In previous sections, we mostly covered the I2C bus. In this section, let us see what we can learn from the I2C device directory whose link name is in the format of `${bus}-${addr}`. The `${bus}` part in the name is a logical I2C bus decimal number, while the `${addr}` part is a hex number of the I2C address of each device.

### I2C Device Directory Content

Inside each I2C device directory, there is a file named `name`. This file tells what device name it was used for the kernel driver to probe this device. Use command `cat` to read its content. For example:

```
/sys/bus/i2c/devices/i2c-73$ cat 73-0040/name
ina230
/sys/bus/i2c/devices/i2c-73$ cat 73-0070/name
pca9546
/sys/bus/i2c/devices/i2c-73$ cat 73-0072/name
pca9547
```

There is a symbolic link named `driver` to tell what Linux kernel driver was used to probe this device:

```
/sys/bus/i2c/devices/i2c-73$ readlink -f 73-0040/driver
/sys/bus/i2c/drivers/ina2xx
/sys/bus/i2c/devices/i2c-73$ readlink -f 73-0072/driver
/sys/bus/i2c/drivers/pca954x
```

But if the link `driver` does not exist at the first place, it may mean that the kernel driver failed to probe this device due to some errors. The error may be found in `dmesg`:

```
/sys/bus/i2c/devices/i2c-73$ ls 73-0070/driver
ls: 73-0070/driver: No such file or directory
/sys/bus/i2c/devices/i2c-73$ dmesg | grep 73-0070
pca954x 73-0070: probe failed
pca954x 73-0070: probe failed
```

Depending on what the I2C device is and what kernel driver was used to probe the device, we may have different content in the device directory.

### I2C MUX Device

While you may be already aware of this in previous sections, an I2C MUX device will have symbolic link `channel-*` inside its device directory. These symbolic links point to their logical I2C bus directories:

```
/sys/bus/i2c/devices/i2c-73$ ls -l 73-0072/channel-*
lrwxrwxrwx ... 73-0072/channel-0 -> ../i2c-78
lrwxrwxrwx ... 73-0072/channel-1 -> ../i2c-79
lrwxrwxrwx ... 73-0072/channel-2 -> ../i2c-80
lrwxrwxrwx ... 73-0072/channel-3 -> ../i2c-81
lrwxrwxrwx ... 73-0072/channel-4 -> ../i2c-82
lrwxrwxrwx ... 73-0072/channel-5 -> ../i2c-83
lrwxrwxrwx ... 73-0072/channel-6 -> ../i2c-84
lrwxrwxrwx ... 73-0072/channel-7 -> ../i2c-85
```

### I2C Sensor Device / Hwmon

I2C sensor device is also common to see. If they are bound by a kernel hwmon (Hardware Monitoring) driver successfully, you will see a `hwmon` directory inside the I2C device directory. Keep digging into it, you will find the Hwmon Sysfs for the I2C sensor device:

```
/sys/bus/i2c/devices/i2c-73/73-0040/hwmon/hwmon17$ ls
curr1_input        in0_lcrit_alarm    name               subsystem
device             in1_crit           power              uevent
in0_crit           in1_crit_alarm     power1_crit        update_interval
in0_crit_alarm     in1_input          power1_crit_alarm
in0_input          in1_lcrit          power1_input
in0_lcrit          in1_lcrit_alarm    shunt_resistor
```

For more info on the Hwmon Sysfs, refer to the doc:

Naming and data format standards for sysfs files

### Instantiate I2C Devices in I2C Sysfs

Refer to the doc:

How to instantiate I2C devices, Method 4: Instantiate from user-space