I2C i2c-tools

I2C i2c-tools

Stable: 10.04.2020 - 15:47 / Revision: 10.04.2020 - 15:43

Contents

1 Article purpose

This article aims to give some first information useful to start with the Linux® tool : I2C tools.

2 Introduction

i2c-tools is a complete user-space package that comes on top of I2C subsystem. It offers:

- tools: a set of I2C programs that make it easy to debug I2C peripherals without having to write any code
- libi2c: library to develop applications.

3 Tools list

- i2cdetect[1]
- i2cdump[2]
- i2cget[3]
- i2cset[4]
- i2ctransfer[5]

4 Installation on your target

i2c-tools is embedded by default in OpenSTLinux distribution.

I2C tools are already bundled within OpenEmbedded. No installation is thus required.

With OpenEmbedded Rocko (2.4.1), I2C tools revision is v3.1.2 and doesn't embed i2ctransfer .

i2ctransfer only comes starting v4.0 included into OpenEmbedded Thud (2.6.x)

5 Getting started

5.1 Devices detection

It can be very helpful to see which peripherals are connected to a specific I2C bus.
Check all instantiated I2C adapters:

```
Board $> i2cdetect -l
```

If I2C adapters are instantiated, the following return will be print :

```
i2c-0   i2c             ST I2C(0xAAAAAAA)                       I2C adapter
i2c-1   i2c             ST I2C(0xBBBBBBB)                       I2C adapter
i2c-2   i2c             ST I2C(0xCCCCCCC)                       I2C adapter
i2c-3   i2c             ST I2C(0xDDDDDDD)                       I2C adapter
i2c-4   i2c             ST I2C(0xEEEEEEE)                       I2C adapter
i2c-5   i2c             ST I2C(0xFFFFFFF)                       I2C adapter
```

Get the list of detected peripherals on the specific I2C bus:

```
Board $> i2cdetect -y <i2cbus number>
```

```
Board $> i2cdetect -y 3
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- UU -- -- -- -- --
50: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

UU -> Probing was skipped, because this address is currently in use by a driver. This strongly suggests that there is a device at this address probed with a driver.

More information about **i2cdetect**[1]

5.2 Read register

Read all the registers from a peripheral:

```
Board $> i2cdump -f -y <i2cbus number> <peripheral address>
```

```
Board $> i2cdump -f -y 0 0x5f
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bc    ...............?
10: 3f 00 87 33 96 be ec a1 9e b2 fe 00 e8 01 80 82    ?.?3????????.????
20: 00 00 00 00 00 00 00 00 51 f2 ae 00 10 f3 c6 00    ........Q??.???.
30: 41 92 a0 0e 00 c4 ee ff 32 03 bf d3 ff ff d0 02    A???.??.2???..??
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bc    ...............?
90: 3f 00 87 33 96 be ec a1 9e b2 fe 00 e8 01 80 82    ?.?3????????.????
a0: 00 00 00 00 00 00 00 00 51 f2 ae 00 10 f3 c6 00    ........Q??.???.
b0: 41 92 a0 0e 00 c4 ee ff 32 03 bf d3 ff ff d0 02    A???.??.2???..??
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

More information about **i2cdump**[2].

To read directly one register, use **i2cget**[3]:

```
Board $> i2cget -f -y <i2cbus number> <peripheral address>
```

Example: read register 0x0f of the peripheral at address 0x5f on bus 0:

```
Board $> i2cget -f -y 0 0x5f 0x0f
0xbc
```

5.3 Write register

To write directly a register, use **i2cset**[4]:

```
Board $> i2cset -f -y <i2cbus number> <peripheral address> <value>
```

Example: write 0xac in register 0x0f of the peripheral at address 0x5f on the bus 0:

```
Board $> i2cset -f -y 0 0x5f 0x0f 0xac
```

A write can fail if the register is in read only mode.

5.4 Auto-increment devices

Even if not part of the I2C standard, it is common to find an automatic incrementation feature on I2C devices, in particular those dealing with large set of registers (typically I2C RAM or EEPROM).

Such devices automatically increment an internal address pointer at each read or write operation, so when several read commands are issued **at the same address**, the value returned at each read may be different each time.

Here are some examples with WM8994 audio codec device on STM32MP157C-EV1 Evaluation board:

First start an audio playback to power-up audio codec device (refer to How to play audio)

Read WM8994 software reset register at address "0x0000":

```
Board $> i2cget -y 0 0x1b 0x00 w
0x9489
```

0x9489 read as a word, understand 0x89 0x94 which is the device id (WM8994) and is indeed the content of the software reset register.

If the same command is repeated again:

```
Board $> i2cget -y 0 0x1b 0x00 w
0x0000
Board $> i2cget -y 0 0x1b 0x00 w
0x0060
```

different values are returned, that do not correspond to the software reset register anymore "0x0000", but correspond successively to registers "0x0001" and "0x0002".

To reset the internal address counter, just write a value at the targeted register address:

```
Board $> i2cset -y 0 0x1b 0x00 0x00
```

Then subsequent read will restart at this address:

```
Board $> i2cget -y 0 0x1b 0x00 w
0x9489
```

Please note that the auto-increment mode may usually be disabled by writing into a device specific configuration register (refer to the device datasheet for details).

5.5 I2C transfer

With OpenEmbedded Rocko (2.4.1), I2C tools revision is v3.1.2 and doesn't embed i2ctransfer .

i2ctransfer becomes available starting with I2C tools revision v4.0 included into OpenEmbedded Thud (2.6.x)

This is a user-space program used to send concatenated I2C messages.

Most devices require a write access to a register before being able to read. **i2ctransfer**[5] offers a way to combine write and read procedures. It also handles multiple bytes write/read in a single command with an additional suffix.

To read a set of bytes:

```
Board $> i2ctransfer -f -y <i2cbus number> r<number of bytes>@<peripheral address>
```

To write a set of bytes:

```
Board $> i2ctransfer -f -y <i2cbus number> w<number of bytes>@<peripheral address>
<byte value 1> <byte value 2> ... <byte value n>
```

To write a set of bytes then read a set of bytes:

```
Board $> i2ctransfer -f -y <i2cbus number> w<number of bytes to write>@<peripheral
address> <byte value 1> <byte value 2> ... <byte value n> r<number of bytes to read>
```

Example (bus 0, read 8 bytes at offset 0x64 from EEPROM at 0x50)

```
Board $> i2ctransfer 0 w1@0x50 0x64 r8
```

"w1" for "write 1 byte" (the 0x64 offset), "r8" for "read 8 bytes"

Example (same EEPROM, at offset 0x42 write 0xff 0xfe ... 0xf0)

```
Board $> i2ctransfer 0 w17@0x50 0x42 0xff-
```

"w17" for "write 17 bytes", first 0x42 byte for the offset, and 0xff- for the 16 subsequent bytes ("-" for auto value decrease starting from 0xff).

See following chapter for 16 bits addressing devices handling.

5.6 16 bits devices handling

The I2C standard protocol supports natively 7 bits of address (or 10 bits of address in extended mode) followed by 8 bits of data.

However some I2C devices embed 16-bit data registers with internal 16-bit address space. Here is how the i2c-tool allows to drive such devices.

- To read a 16 bits value, add "**w**" for "word" at the end of command:

```
Board $> i2cget -f -y <i2cbus number> <peripheral address> <address> w
```

Please note that <address> is 8-bit wide, while the returned data is 16-bit wide. The **interpretation of <address> is device dependent** (One possible interpretation is that <address> drives the 8 MSB bits of the 16-bit address while the 8 LSB bits are set to 0).

- To write a 16 bits value specifying the 16 bits address, send both the address and the data as a set of bytes in a single "I2C block write" by adding "**i**" at the end of **i2cset**[4] command:

```
Board $> i2cset -f -y <i2cbus number> <peripheral address> <MSB address> <LSB address>
<MSB value> <LSB value> i
```

Here are some examples with WM8994 audio codec device on STM32MP157C-EV1 Evaluation board:

First start an audio playback to power-up audio codec device (refer to How to play audio)

- Read the device id from register "Software Reset" at address 0x0000:

```
Board $> i2cget -y 0 0x1b 0x0 w
0x9489
```

"w" stands for "word" access. Since the word is read in **little endian** and the device is big endian, we have to reverse the endianness.

The returned word 0x9489 should be interpreted as 0x89 0x94 which is the indeed the (WM8994) device ID.

- Update value of register "AIF1 Control" at address 0x0300:

```
Board $> i2cget -y 0 0x1b 0x3 w
0x5040
```

Current value is 0x4050.

Let's assume the AIF1ADC_TDM pin needs to be put in tristate, this is done by settings bit 13, hence by writing 0x6050:

```
Board $> i2cset -y 0 0x1b 0x03 0x00 0x60 0x50 i
Board $> i2cget -y 0 0x1b 0x3 w
0x5060
```

The "AIF1 Control" register value has been updated to 0x6050 as expected.

Doing the same with I2C transfer is far more simple:

```
Board $> i2ctransfer -f -y 0 w4@0x1b 0x03 0x00 0x60 0x50 -r2
Board $> 0x60 0x50
```

"w4" for "write 4 bytes": the first 2 bytes for address (0x0300), the next 2 bytes for register address (0x6050)

"r2" for "read 2 bytes": the word register value

6 References

All these tools (i2cset, i2cget, i2cdump, i2cdectect and i2ctransfer) are available in this GIT:

```
git clone git://git.kernel.org/pub/scm/utils/i2c-tools/i2c-tools.git or
git clone https://git.kernel.org/pub/scm/utils/i2c-tools/i2c-tools.git
```

Source code browsing I2C Tools Code

- 1.01.1 https://www.mankier.com/8/i2cdetect
- 2.02.1 https://www.mankier.com/8/i2cdump
- 3.03.1 https://www.mankier.com/8/i2cget
- 4.04.14.2 https://www.mankier.com/8/i2cset
- 5.05.1 https://www.mankier.com/8/i2ctransfer

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today forvolatile random-accesssemiconductor memories.)

Electrically-erasable programmable read-only memory