



Basics of I²C on Linux

Luca Ceresoli

luca.ceresoli@bootlin.com

© Copyright 2004-2022, Bootlin.

Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!





- ▶ Embedded Linux engineer at **Bootlin**
 - Embedded Linux experts
 - Engineering services: Linux BSP development, kernel porting and drivers, Yocto/Buildroot integration, real-time, boot-time, security, multimedia
 - Training services: Embedded Linux, Linux kernel drivers, Yocto, Buildroot, graphics stack, boot-time, real-time
- ▶ Linux kernel and bootloader development, Buildroot and Yocto integration
- ▶ Open-source contributor
- ▶ Living in **Bergamo**, Italy
- ▶ `luca.ceresoli@bootlin.com`

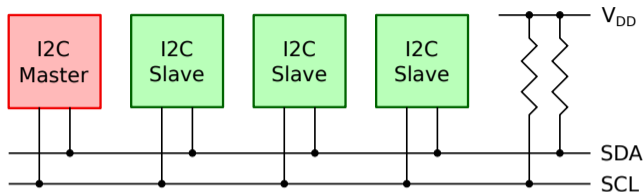
<https://bootlin.com/company/staff/luca-ceresoli/>



What is I²C



What is I²C

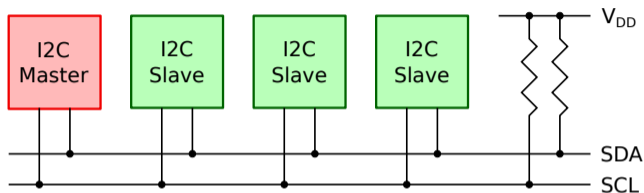


<https://docs.kernel.org/i2c/summary.html>

- ▶ A bus for Inter-Integrated-Circuit communication
- ▶ Design for hardware simplicity: 2 wires, many chips per bus, flexible
- ▶ Not discoverable, not plug-and-play
- ▶ Low speed: 100-400 kHz (with 1 MHz and 3.4 MHz extensions)
- ▶ Also known as: I2C, IIC, TWI, TWSI, ...
- ▶ <https://en.wikipedia.org/wiki/I²C>
- ▶ <https://docs.kernel.org/i2c/>



Roles



<https://docs.kernel.org/i2c/summary.html>

Adapter

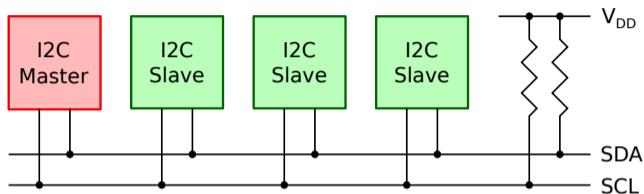
- ▶ Other names: Master, Controller, bus
- ▶ Initiates all transactions
- ▶ Usually one (multimaster possible)
- ▶ Has no address

Client

- ▶ Other names: Slave, Device
- ▶ Responds to transactions
- ▶ Many per bus
- ▶ 7-bit address set in hardware (10-bit extension)



Two wires



<https://docs.kernel.org/i2c/summary.html>

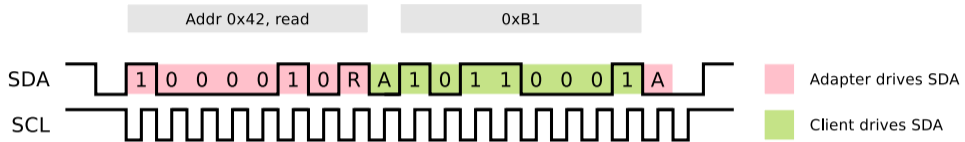
▶ Two wires

- SDA: data, bidirectional
- SCL: “clock”
 - Not really a clock
 - SDA moved at SCL falling edge, SDA read at SCL rising edge
 - Mostly driven by adapter, sometimes also by clients (clock stretching)

▶ Open collector



Communication protocol



1. Start condition
2. Adapter sends: client address (7 bit) + direction bit (R/W)
3. Client sends ACK
4. Client sends one byte
5. Adapter sends ACK
6. Stop condition



The SMBus protocol

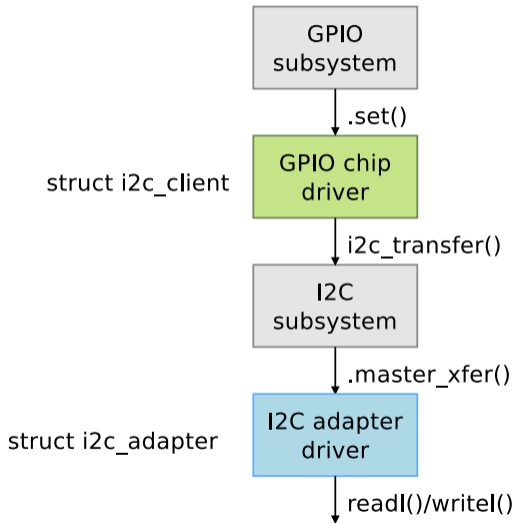
- ▶ Designed for chip communication on PC motherboards
- ▶ Mostly a subset of I²C
- ▶ Defines several commands
 - Register write: S addr+W A reg A data P
 - Register read: S addr+W A reg A RS addr+R A data NA P
- ▶ Often I²C and SMBus clients can be mixed on the same bus
 - Linux recommends using SMBus APIs for I²C chips when possible
- ▶ https://en.wikipedia.org/wiki/System_Management_Bus



I²C in the Linux Driver Model



The I2C subsystem in the linux kernel





I2C “Devices” includes both adapters and clients:

```
# ls -l /sys/bus/i2c/devices/  
lrwxrwxrwx ... 0-0039 -> ../../../../devices/platform/soc/40012000.i2c/i2c-0/0-0039  
lrwxrwxrwx ... 0-004a -> ../../../../devices/platform/soc/40012000.i2c/i2c-0/0-004a  
lrwxrwxrwx ... 1-0052 -> ../../../../devices/platform/soc/40015000.i2c/i2c-1/1-0052  
lrwxrwxrwx ... 2-0028 -> ../../../../devices/platform/soc/5c002000.i2c/i2c-2/2-0028  
lrwxrwxrwx ... 2-0033 -> ../../../../devices/platform/soc/5c002000.i2c/i2c-2/2-0033  
lrwxrwxrwx ... i2c-0 -> ../../../../devices/platform/soc/40012000.i2c/i2c-0  
lrwxrwxrwx ... i2c-1 -> ../../../../devices/platform/soc/40015000.i2c/i2c-1  
lrwxrwxrwx ... i2c-2 -> ../../../../devices/platform/soc/5c002000.i2c/i2c-2  
lrwxrwxrwx ... i2c-3 -> ../../../../devices/platform/soc/40012000.i2c/i2c-0/i2c-3
```



Device Tree



Device tree example

arch/arm/boot/dts/stm32mp15xx-dkx.dtsi

```
&i2c4 {
    i2c-scl-rising-time-ns = <185>;
    i2c-scl-falling-time-ns = <20>;
    clock-frequency = <400000>;
    status = "okay";
    // ...

    stusb1600@28 {
        compatible = "st,stusb1600";
        reg = <0x28>;
        interrupts = <11 IRQ_TYPE_LEVEL_LOW>;
        interrupt-parent = <&gpioi>;
        pinctrl-names = "default";
        pinctrl-0 = <&stusb1600_pins_a>;
        status = "okay";
        // ...
    };

    pmic: stpmic@33 {
        compatible = "st,stpmic1";
        reg = <0x33>;
        // ...
    }
}
```



More properties

▶ Adapter node

- compatible
- #address-cells = <1> (1 address number per client chip)
- #size-cells = <0> (no size numbers per client chip)
- Optional: clock-frequency (frequency of bus clock in Hz)
- Optional: i2c-scl-falling-time-ns, i2c-sda-falling-time-ns, ...
- Optional: scl-gpios, sda-gpios: for GPIO bus recovery
- Optional: single-master or multi-master
- Adapter-specific properties
- ...
- One subnode per client chip
 - reg = <client address> (Look for “Slave address” on the datasheet)
 - compatible
 - Client-specific properties

▶ See [Documentation/devicetree/bindings/i2c/i2c.txt](https://bootlin.com/docs/devicetree/bindings/i2c/i2c.txt)



Writing *client* device drivers



Client device driver: declare the driver

drivers/gpio/gpio-pca9570.c

```
static struct i2c_driver pca9570_driver = {
    .driver = {
        .name = "pca9570",
        .of_match_table = pca9570_of_match_table, // --> see later
    },
    .id_table = pca9570_id_table, // --> see later
    .probe_new = pca9570_probe, // --> see later
};
module_i2c_driver(pca9570_driver);
```




Client device driver: i2c and device tree tables

drivers/gpio/gpio-pca9570.c

```
static const struct i2c_device_id pca9570_id_table[] = {
    { "pca9570", 4 },
    { "pca9571", 8 },
    { /* sentinel */ }
};
MODULE_DEVICE_TABLE(i2c, pca9570_id_table);

static const struct of_device_id pca9570_of_match_table[] = {
    { .compatible = "nxp,pca9570", .data = (void *)4 },
    { .compatible = "nxp,pca9571", .data = (void *)8 },
    { /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, pca9570_of_match_table);
```



Client device driver: probe function

drivers/gpio/gpio-pca9570.c

```
static int pca9570_probe(struct i2c_client *client)
{
    struct pca9570 *gpio;

    gpio = devm_kzalloc(&client->dev, sizeof(*gpio), GFP_KERNEL);
    if (!gpio)
        return -ENOMEM;

    gpio->chip.get = pca9570_get; // --> see later
    gpio->chip.set = pca9570_set; // --> see later
    // ...

    i2c_set_clientdata(client, gpio);

    return devm_gpiochip_add_data(&client->dev, &gpio->chip, gpio);
}
```



Client device driver: recap

drivers/gpio/gpio-pca9570.c

```
static int pca9570_probe(struct i2c_client *client)
{
    // 1. allocate driver-specific struct
    // 2. fill it
    // 3. device-specific initializations
    // 4. i2c_set_clientdata(client, <driver-specific struct>)
    // 5. register to appropriate subsystem (GPIO, RTC, input, IIO, ...)
}

// 6. Describe i2c device in struct i2c table and device tree table
// 7. Describe driver in a struct i2c_driver
// 8. module_i2c_driver(): declare the driver
```



Client device driver: requesting I²C transactions

drivers/gpio/gpio-pca9570.c (simplified)

```
static void pca9570_set(struct gpio_chip *chip, unsigned offset, int value)
{
    struct pca9570 *gpio = gpiochip_get_data(chip);
    struct i2c_client *client = to_i2c_client(gpio->chip.parent);
    u8 buffer;

    buffer = /* chip-specific code */;

    i2c_smbus_write_byte(client, buffer);
}
```



Requesting I²C transactions

- ▶ Simple buffer transfer
 - `i2c_smbus_write_byte()`: send one byte
 - `i2c_smbus_read_byte()`: receive one byte
 - `i2c_master_send()`: send multiple bytes
 - `i2c_master_recv()`: receive multiple bytes
- ▶ Register-like access
 - `i2c_smbus_write_byte_data()`: write a register
 - `i2c_smbus_read_byte_data()`: read a register
 - Plus variants transferring words or buffers
- ▶ And more, see:
 - <https://docs.kernel.org/i2c/i2c-protocol.html>
 - <https://docs.kernel.org/i2c/smbus-protocol.html>
- ▶ ...or use `i2c_transfer()`, the “swiss army knife of Linux I2C”
 - Makes any number of transfers
 - Does repeated start by default
 - Various flags to tweak its behaviour



i2c_transfer()

sound/soc/codecs/adau1701.c (simplified)

```
static int adau1701_reg_read(void *context, unsigned int reg, unsigned int *value)
{
    uint8_t send_buf[2], recv_buf[3];
    struct i2c_msg msgs[2];

    msgs[0].addr = client->addr;
    msgs[0].len = sizeof(send_buf);
    msgs[0].buf = send_buf; // pre-filled
    msgs[0].flags = 0; // Write transaction by default

    msgs[1].addr = client->addr;
    msgs[1].len = size;
    msgs[1].buf = recv_buf;
    msgs[1].flags = I2C_M_RD; // Read transaction

    ret = i2c_transfer(client->adapter, msgs, ARRAY_SIZE(msgs));
    if (ret < 0) return ret;
    else if (ret != ARRAY_SIZE(msgs)) return -EIO;
}
```



Userspace tools



I2C from userspace

- ▶ The first rule about I2C from userspace:
- ▶ Do not use I2C from userspace
- ▶ Use the RTC/ALSA/IIO device instead, I2C is just to get you there



- ▶ The `i2c-tools` package provides tools to access I²C on the command line
- ▶ Useful for debugging, testing, some simple prototyping
- ▶ Accesses the I²C bus via `/dev/i2c-0`, `/dev/i2c-1`...
- ▶ Assume devices have registers, SMBus-like
- ▶ WARNING! This program can confuse your I2C bus, cause data loss and worse!
- ▶ https://i2c.wiki.kernel.org/index.php/I2C_Tools



i2cdetect

- ▶ i2cdetect: detect devices on a bus
- ▶ No guarantee it works (I²C is not discoverable by the spec)

```
# i2cdetect -l
i2c-0  i2c  STM32F7 I2C(0x40012000)  I2C adapter
i2c-1  i2c  STM32F7 I2C(0x40015000)  I2C adapter
i2c-2  i2c  STM32F7 I2C(0x5c002000)  I2C adapter
i2c-3  i2c  i2c-0-mux (chan_id 0)    I2C adapter
# i2cdetect -y 2
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- 28 -- -- -- -- --
30: -- -- -- UU -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- --
#
```

-- No response
28 Response from address 28
UU Address in use (by driver)



i2cget and i2cset

- ▶ i2cget: read a register value
- ▶ i2cset: set a register value
- ▶ Can use various types of SMBus and I²C transactions
- ▶ Limited to 8-bit register address

```
# i2cget -y 2 0x28 0x1b  
0x21  
# i2cset -y 2 0x28 0x55  
#
```



▶ i2cdump: dump value of all registers

```
# i2cdump -y 2 0x28
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 12 00 11 20 00 00 f3 00 00 00  .....?? ..?...
10: 04 00 00 40 00 00 00 00 20 00 00 21 00 e0 00 00  ?..@.... ..!?...
20: a0 32 00 00 00 ac 00 00 02 00 00 00 00 00 01 10  ?2...?..?.....??
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
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: a3 2c 91 01 08 2c d1 02 00 2c c1 03 00 f0 b0 04  ? , ??? , ?? . , ?? . ???
80: 00 af 40 06 00 00 90 01 08 2c d1 02 00 2c c1 03  .?@?..???,??. ,??
90: 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00  .....@.....
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
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  .....
#
```



- ▶ i2ctransfer: the “swiss army knife of Linux I2C”, in userspace
- ▶ Example: reimplement the `i2cget -y 2 0x28 0x1b` command:

```
# i2ctransfer -y 2 w1@0x28 0x1b r1@0x28  
0x21  
#
```

- ▶ `w1@0x28` Write transaction, 1 byte, client address 0x28
- ▶ `0x1b` Data to send in the write transaction
- ▶ `r1@0x28` Read transaction, 1 byte, client address 0x28



Hardware tools

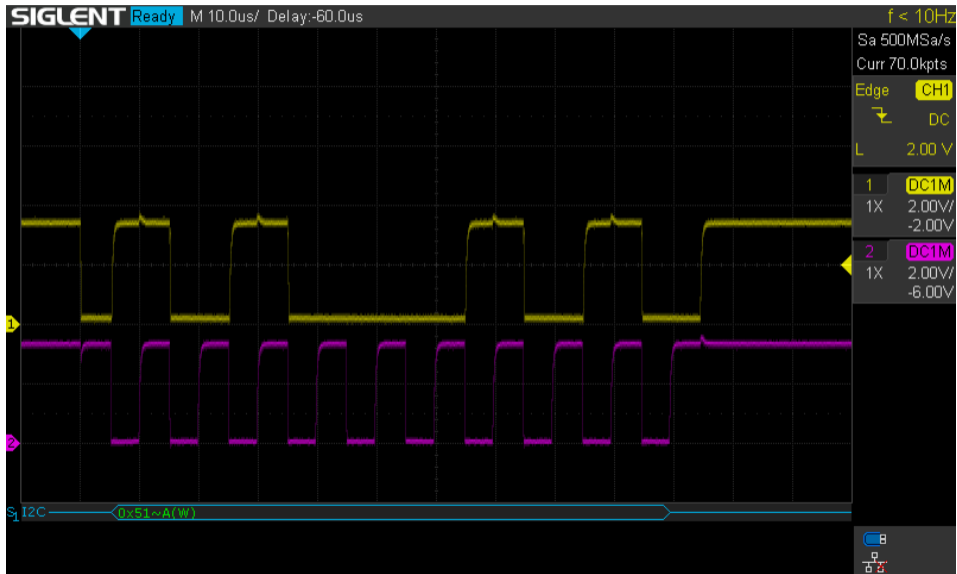


Oscilloscope

- ▶ Can show SCL and SDA with all the details
- ▶ Useful to check voltage levels, slopes, noise...
- ▶ Many models can visually decode I²C and other protocols

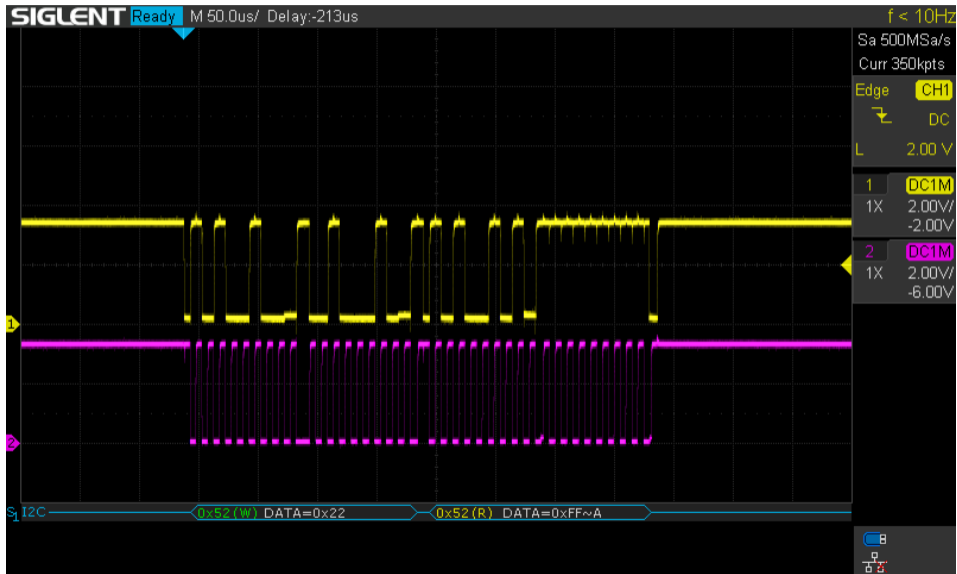


Oscilloscope — NACK





Oscilloscope — Register read





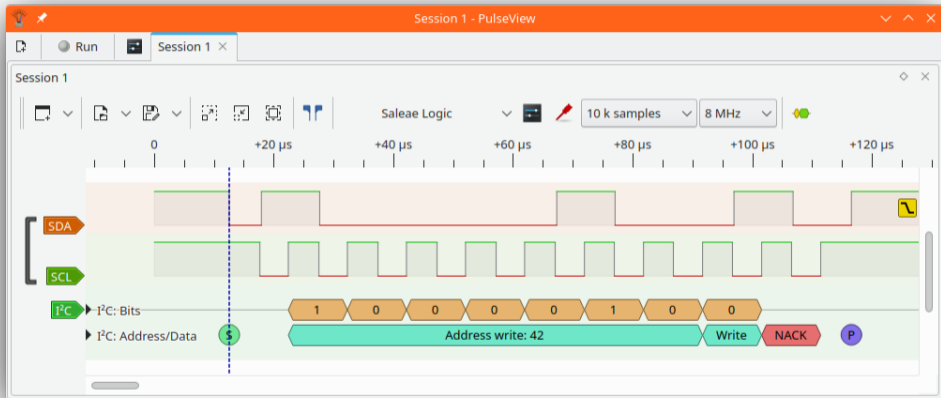
Logic analyzer

- ▶ **Sigrok** is suite of signal analysis software
 - <https://sigrok.org>
- ▶ **Pulseview**: a logic analyzer and oscilloscope, based on Sigrok
 - Visually decodes I²C and other protocols
 - <https://sigrok.org/wiki/PulseView>
- ▶ Open source, GPLv3+
- ▶ They work well with cheap acquisition devices



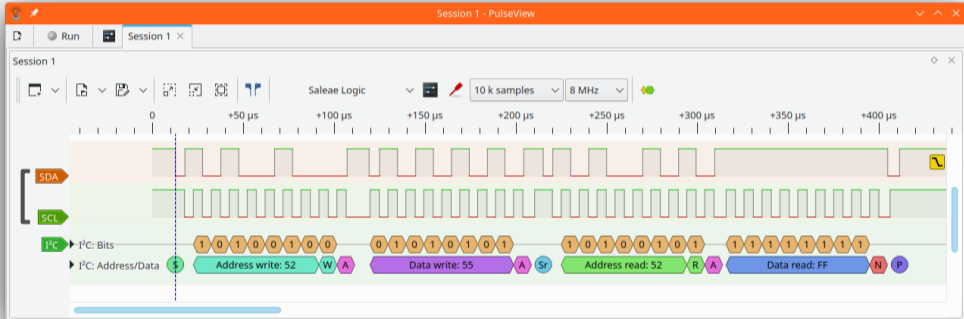


Pulseview — NACK





Pulseview — Register read





Troubleshooting



Troubleshooting tools

- ▶ Return code from `i2c_*`() functions — Never ignore errors!
- ▶ Kernel logs
- ▶ `i2c-tools`
- ▶ Oscilloscope or logic analyzer



No ACK from client — systematic

- ▶ Problem: a client **never** responds to transactions
 - i2c-tools symptom: Error: Read failed
 - Kernel internal APIs symptom: `-ENXIO`
- ▶ `i2cdetect`: a client (possibly yours) at any unexpected address?
 - Check address pins on client chip: datasheet, schematics
- ▶ `i2cdetect`: no client at any unexpected address?
 - Client not powered, held in reset, broken, unsoldered pin
- ▶ Oscilloscope: no activity on bus, SCL/SDA always high
 - Pinmux (I²C adapter not reaching the pads)
 - Device tree: device under wrong bus
- ▶ Oscilloscope: no activity on bus, SCL/SDA always low
 - Missing pull-up resistors (external or internal)



No ACK from client — sporadic

- ▶ Problem: a client **sporadically** does not respond to transactions
 - i2c-tools symptom: `Error: Read failed`
 - Kernel internal APIs symptom: `-ENXIO`
- ▶ Oscilloscope: SCL/SDA lines return to high level too slowly
 - Weak pull-up
 - Workaround: reduce `clock-frequency` in device tree
- ▶ Oscilloscope: noise on SCL/SDA lines
 - Hardware must be fixed
- ▶ Oscilloscope: SCL/SDA delays incorrect
 - Propagation delay in lines at high speed? Review PCB
 - Tune `i2c-scl-internal-delay-ns...`
 - Workaround: reduce `clock-frequency` in device tree



No ACK from client after reset

- ▶ Problem: a client **sporadically** does not respond after unclean reset
 - Symptom: driver fails to respond, fails to probe
- ▶ No clean shutdown → driver could not set client to idle state
 - E.g. client left in the middle of a transaction, kernel starts a new one
- ▶ Reset all clients during boot
 - In hardware, if possible
 - In the bootloader otherwise



Bus busy

- ▶ Problem: SCL line held low
 - Symptom: `bus busy` in kernel logs

```
stm32f7-i2c 40015000.i2c: bus busy
stm32f7-i2c 40015000.i2c: Trying to recover bus
```

- ▶ Systematic
 - Short circuit / mounting problem
- ▶ Sporadic
 - Chip gone crazy
 - Bus recovery could fix it
 - Multimaster problem

Questions? Suggestions? Comments?

Luca Ceresoli

luca.ceresoli@bootlin.com

Slides under CC-BY-SA 3.0

<https://bootlin.com/pub/conferences/>



Extra slides

Luca Ceresoli

luca.ceresoli@bootlin.com

© Copyright 2004-2022, Bootlin.

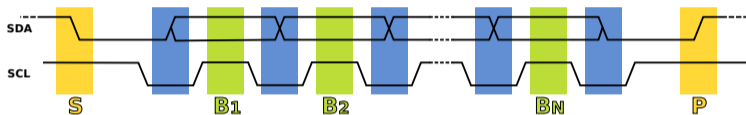
Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!





Bit-level communication

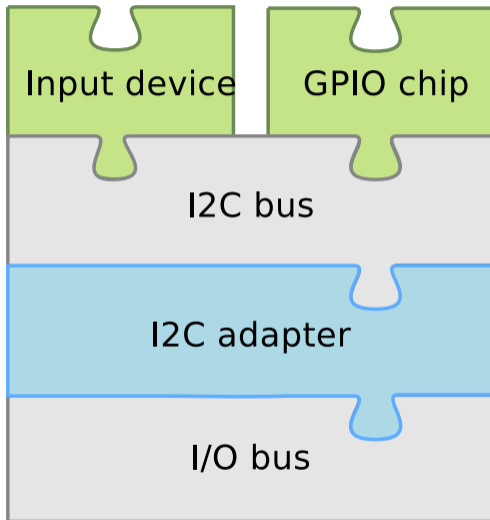


https://upload.wikimedia.org/wikipedia/commons/6/64/I2C_data_transfer.svg

- ▶ SCL low = move SDA
- ▶ SCL high = sample SDA
- ▶ Exception: Start / Stop condition



The driver model





I²C muxes and switches

