More robust I2C designs with a new fault-injection driver

Wolfram Sang, Consultant / Renesas

ELCE17

Motivation

It really got personal...

- I2C maintainer since 2012
- encountered similar type of problems handling rare error cases in I2C master drivers again and again
- myself unsure how drivers for Renesas I2C IP cores behaved

... so as a first step

- reproducible way to generate test cases was desired!

Introduction: sigrok

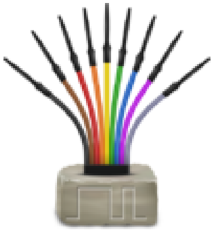


Figure 1: `https://www.sigrok.org`

The sigrok project aims at creating a portable, cross-platform, Free/Libre/Open-Source signal analysis software suite that supports various device types (e.g. logic analyzers, oscilloscopes, and many more).[1]

[1] from their website

Introduction: sigrok II

Features & Design goals[2]

- Broad hardware support

  logic analyzers, oscilloscopes, multimeters, data loggers etc.
- Cross-platform
- Scriptable protocol decoding

  stackable, Python3
- File format support

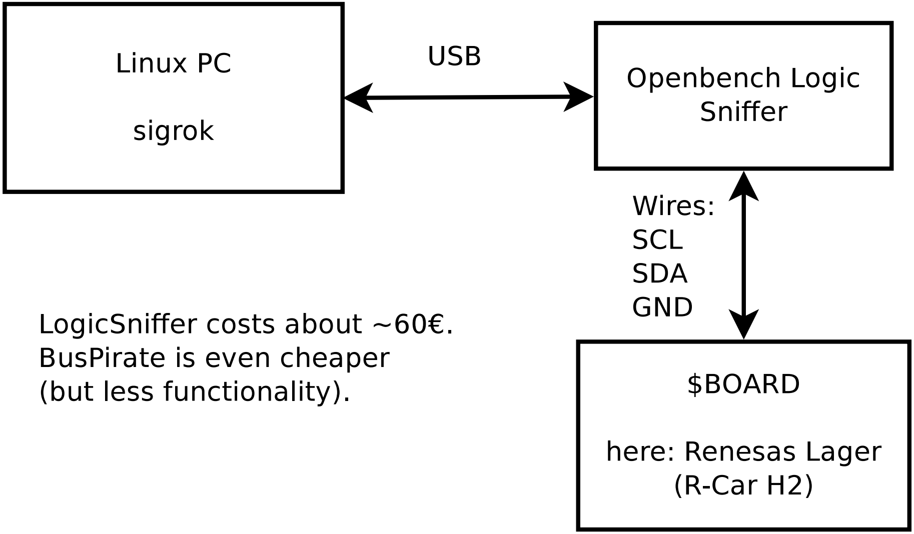
  binary, ASCII, hex, CSV, gnuplot, VCD, WAV, ...
- Reusable libraries

  libsigrok, libsigrokdecode
- Various frontends

  PulseView (LA GUI), sigrok-meter (DMM GUI), sigrok-cli

[2] from their website, slightly shortened

Setup for sigrok

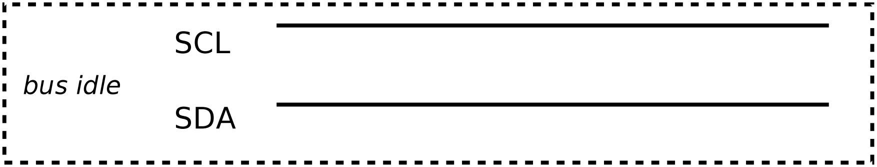


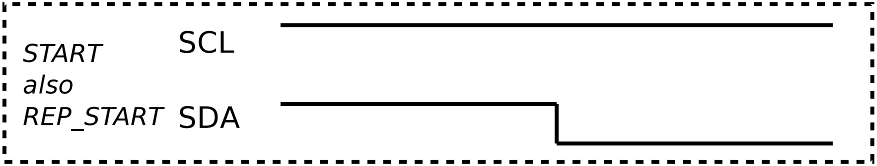
LogicSniffer costs about ~60€.
BusPirate is even cheaper
(but less functionality).

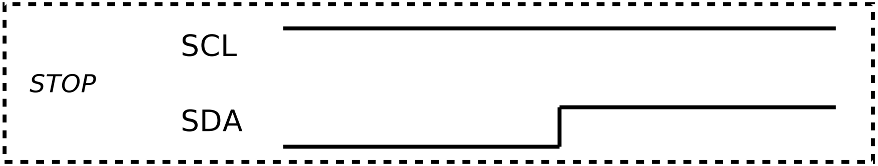
Live demo setup

Click here and there until everything works :)

Some basics: about START and STOP

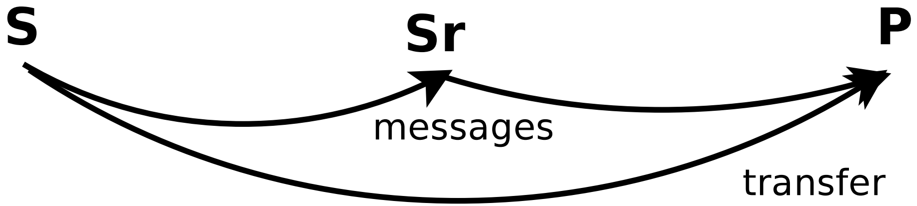






Definitions of ‘message’ and ‘transfer’

transfer	everything between START and STOP
message	everything between START or REP_START and STOP or REP_START



Live demo 1

Difference between STOP+START vs. REP_START on the wire

It really happens!

```
From: Giuseppe Cantavenera <...>
Subject: Re: [PATCH] i2c-cadence: fix repeated start in
     message sequence

...
Sadly, it would have saved our team weeks of investigation
on a major issue if we had noticed before, but that's our
problem :(
...
```

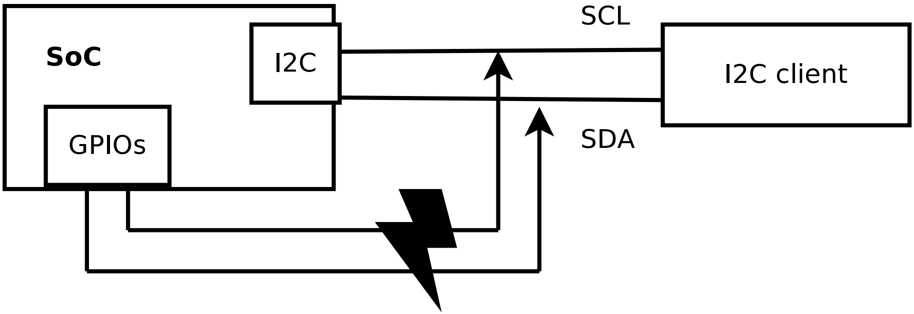
How to debug error cases?

Cases of interest

- stalled bus!
    - SDA stuck low
    - SCL stuck low
- arbitration lost
- faulty bits

Those usually happen rarely. Even if, often hard to reproduce.

Solution: fault-injector



GPIOs driven by extended *i2c-gpio* driver

GPIO based I2C fault injector

Implementation details

- currently compiled-in extension to i2c-gpio driver
- might be refactored to an additional module if it grows too large
- controlled by files in `debugfs`
    - if you don't know it already, super-convenient for such cases. Much better than sysfs!

Error case: SDA held low by a device

How it can happen

- Handover between bootloader and Kernel during a transfer
- Watchdog resets system during a transfer
- Device got stuck

What it means

- SCL high, SDA low (held by the client device) $\rightarrow$ bus not free

How it is simulated

- address phase to a known client is started
- when client acks its presence we stop clocking SCL

Live demo 2

Incomplete transfer to

- *the PMIC*
- *the audio codec*

I2C bus recovery

I2C specs have a solution for this (Revision 6, Chapter 3.1.16):

> *If the data line (SDA) is stuck LOW, the master should send nine clock pulses. The device that held the bus LOW should release it sometime within those nine clocks. If not, then use the HW reset or cycle power to clear the bus.*

The Linux Kernel has support for that

- populate a `bus_recovery_info` structure
- generic helpers if SCL/SDA are controllable
- generic helpers if you want to use GPIOs

Live demo 3

Incomplete transfer to the audio codec using another I2C IP core

When to *not* use bus recovery

Not suitable when

- SDA is not low

  you should try emitting a STOP
- the transfer timed out

  could happen because device is busy

  Problem! I2C has no timeouts defined. SMBus has.
- SCL is stuck low

  we'll talk about that very soon

SO

- only when SDA is stuck low at the beginning of a transfer

sometimes doing $RANDOM things will recover a device for you. But $RANDOM might break things for other users randomly.

Error case: SCL held low by a device

How it can happen

- Device got stuck

What it means

- SCL low (held by the client device), SDA doesn't really matter $\rightarrow$ bus not free *and* we cannot clock SCL

How it is simulated

- SCL is pinned low by the GPIO

Live demo 4

pinning SCL low

Solution is to reset

I2C specs also have a solution for this (Revision 6, Chapter 3.1.16):

> *In the unlikely event where the clock (SCL) is stuck LOW, the preferential procedure is to reset the bus using the HW reset signal if your I2C devices have HW reset inputs. If the I2C devices do not have HW reset inputs, cycle power to the devices to activate the mandatory internal Power-On Reset (POR) circuit.*

not much we can do

- return -EBUSY and let the client driver handle the necessary steps

Outlook

add some more failure cases

- arbitration lost

  hold SDA low for a while once we detect START
- SDA stuck low without external device

  hold SDA low until we counted some SCL pulses
- insert some faulty bits

  could be used to check PEC bytes

decide whether to use add-on module

- all this extra code might bloat the core driver source

Summary

What has been shown:

- I2C can be measured without much effort and cost
- really easy to detect incorrect sequences
- faults can be injected via an extended i2c-gpio driver
- I2C host drivers can then be checked against that
- when to use bus recovery and when not

Let’s do good engineering :)

Thank you!

Questions?

- Right here, right now...
- Later at the conference
- `wsa@the-dreams.de`

And thanks again to Renesas for funding this work!