Application note

DA9053 2-Wire Implementation

AN-PM-003

Abstract

This application note outlines a software workaround to prevent issues arising with initial I2C communication between a processor and the DA9053 2-wire interface. The workaround can be applied to any I2C slave device, and ensures that the device is in the correct idle state, ready to accept communications.

Contents

1 Terms and definitions

DA905x	For this document this will be used to represent DA9021, DA9022, DA9052, DA9053 and DA9057

2 References

[1] DA9053-00-IDS3a_140114.pdf, Datasheet, Dialog Semiconductor

[2] UM10204, I2C-bus specification and user manual, Rev .4 – 13 February 2012, NXP Semiconductors

3 Introduction

The DA9053 I2C communication function does not receive a reset signal from an external device. During power up, the I2C function can enter an invalid state which can have one of two repercussions. The first is that the DA9053 might ignore the very first I2C message that was intended for it. The second is where the DA9053 might hold the SDA line low. Both of these can result in the I2C bus being seen as invalid or locked, preventing further communications. Both of these can be corrected in software by implementing the following workaround. This workaround is also referred to in the I2C specification.

4 NAK of the first I2C message

Figure 1 below shows an example of the DA9053 responding with a NAK to the first message. As seen in the oscilloscope plot, the SDA line is high for the last clock cycle. (Yellow trace = SCL and blue trace = SDA.)

This can only happen as the DA9053 starts. As the IO supply rail for the I2C pull-ups rises, the 2-wire interface logic can detect an invalid condition due to the timing of the two signals. This happens because the I2C module is already out of reset as this power rail comes up. Once in the invalid condition, it can miss the next valid START condition on the 2-wire interface and treat it as the first bit of the message. If the power rail for the I2C pull-ups is present before the DA9053 comes out of RESET then the first I2C access will be correctly handled.

The workaround suggested will prevent a NAK on the first communication. It is assumed that the software will correctly deal with any other status messages from the I2C communication.

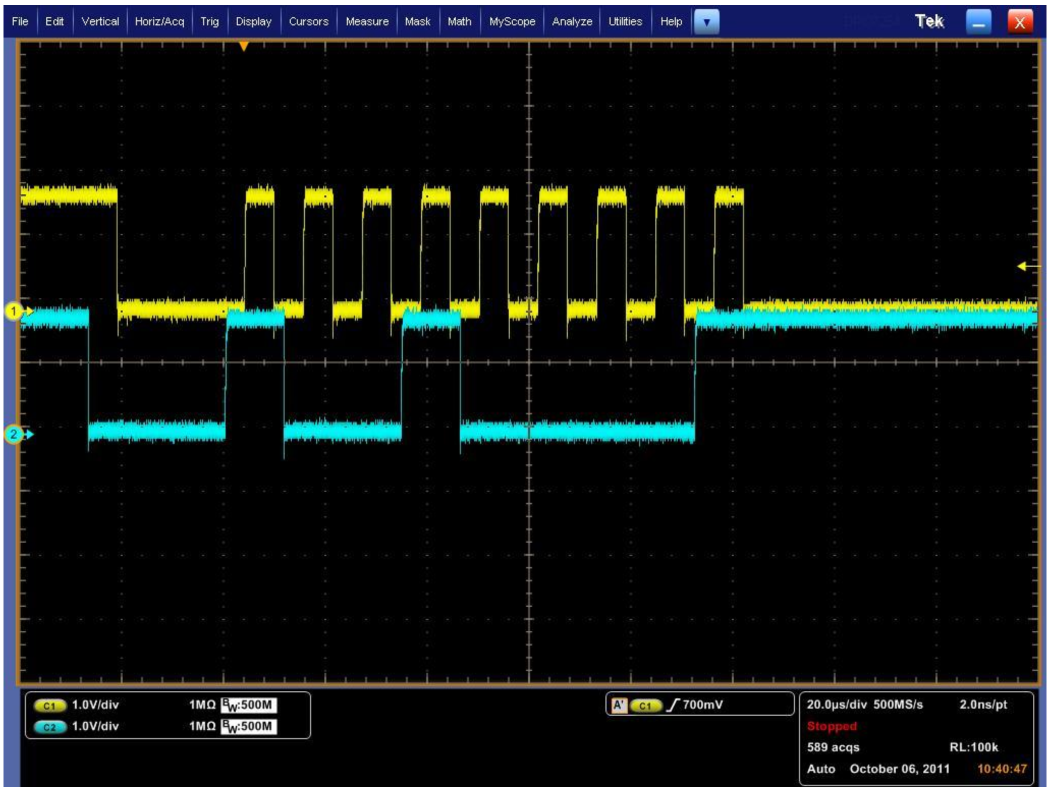

Figure 1: Diagram Example

5 SDA signal line is stuck low

This is the more serious situation whereby the invalid condition results in the DA9053 driving the SDA signal low. This will appear to the I2C host as though there is another master driving the bus and will lock the I2C bus.

This scenario can occur if the DA9053 shuts down during an I2C message whilst it is driving the bus low. If the DA9053 is shutdown in the middle of an I2C message but it is not driving SDA low, then upon the restart of the DA9053 it is possible that the first message will be answered with a NAK. This situation can arise because the I2C module in the DA9053 does not get reset: it therefore maintains its previous state and believes that it should still be driving the bus when it is switched back on. Therefore, once a shutdown command has been sent to the DA9053, no further traffic on the I2C bus should be allowed.

6 Workaround

Both of the aforementioned issues can be prevented by applying a one-time software workaround before any I2C communication is attempted. The workaround is most usually placed in the boot loader.

As previously mentioned, the fix is to implement the same workaround that is described in the I2C specification. The procedure is:

1. Configure the I2C port pins to be GPIOs, preferably open-drain but push-pull is also acceptable.
2. Set the SCL line low then high nine times with at least 3 µs between changes, for example: udelay(3)
3. Set SDA line low. Wait at least 3 µs
4. Set SCL line low. Wait at least 3 µs
5. Set SCL line high. Wait at least 3 µs
6. Set SDA line high. Wait at least 3 µs
7. Configure the pins to be under control of I2C controller.

Note that this procedure must be made when the signal lines are being operated as GPIOs since the state of the lines may preclude use of I2C primitive controls: for example, sending START would not work if SDA were already low.

The above procedure can be seen in Figure 2 below.

6.1 Code Example for the iMX53

```
static void reset_i2c_clients(unsigned int module_base)
{
     unsigned int i, reg;

     /* GPIO5[26] for i2c1 SDA */
     mxc_request_iomux(MX53_PIN_CSI0_D8, IOMUX_CONFIG_ALT1);
     mxc_iomux_set_pad(MX53_PIN_CSI0_D8, PAD_CTL_DRV_HIGH |
                PAD_CTL_PKE_ENABLE | PAD_CTL_PUE_PULL |
                PAD_CTL_100K_PU);
     /* GPIO5[27] for i2c1 SCL */
     mxc_request_iomux(MX53_PIN_CSI0_D9, IOMUX_CONFIG_ALT1);
     mxc_iomux_set_pad(MX53_PIN_CSI0_D9, PAD_CTL_DRV_HIGH |
                PAD_CTL_PKE_ENABLE | PAD_CTL_PUE_PULL |
                PAD_CTL_100K_PU);
```

```
        reg = readl(GPIO5_BASE_ADDR + 0x0);
        reg &= ~(1<<26 | 1<<27);
        writel(reg, GPIO5_BASE_ADDR + 0x0);
        reg = readl(GPIO5_BASE_ADDR + 0x4);
        reg &= ~(1<<26 | 1<<27);
        writel(i, GPIO5_BASE_ADDR + 0x4);

        udelay(200);

        for (i = 10; i; i--) {
                reg |= 1<<27;
                writel(reg, GPIO5_BASE_ADDR + 0x4);
                udelay(11);

                reg &= ~(1<<27);
                writel(reg, GPIO5_BASE_ADDR + 0x4);
                udelay(14);
        }

        udelay(80);

        reg |= 1<<26;
        writel(reg, GPIO5_BASE_ADDR + 0x4);
        udelay(11);
        reg |= 1<<27;
        writel(reg, GPIO5_BASE_ADDR + 0x4);
        udelay(11);
        reg &= ~(1<<27);
        writel(reg, GPIO5_BASE_ADDR + 0x4);
        udelay(14);
        reg &= ~(1<<26);
        writel(reg, GPIO5_BASE_ADDR + 0x4);
}
```

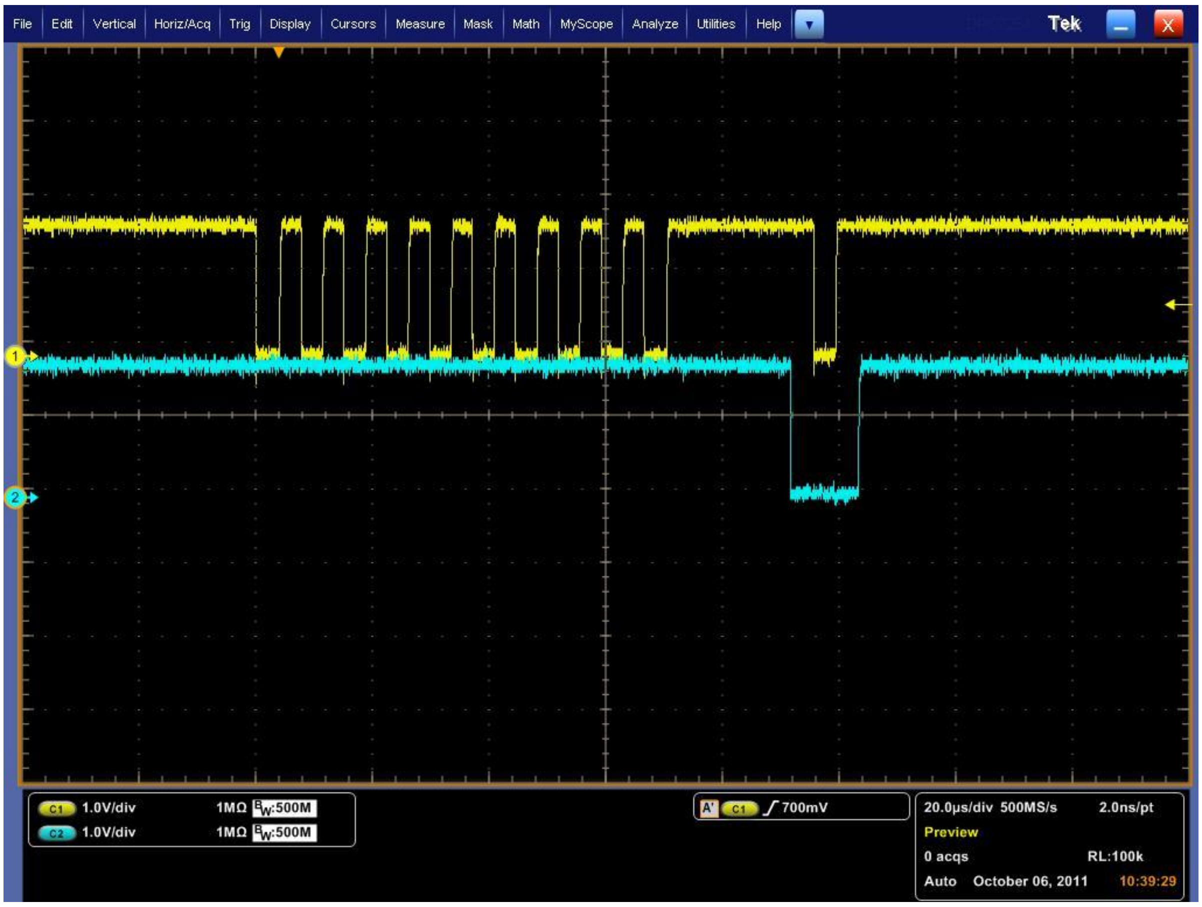

Figure 2: Fix for clearing the 2-wire SDA interface

The above implementation must be done before any attempt is made to communicate on the I2C bus.

Note that this procedure is very similar to that defined in the UM10204 section 3.1.16 of the I2C-bus specification and user manual from NXP Semiconductors.

Also note that this procedure can be safely applied to any I2C slave device and will result in that device being placed in the idle state.

7 Further system recommendations

The DA9053 contains two I2C interfaces and a 4-wire interface. In applications where the communication with the PMIC is critical to the system performance, it is recommended to use the 4-wire interface where possible. An example is a system that will use frequent DVC to continually optimise the power/performance of the processor. The 4-wire interface is faster, is more robust due to its use of an enable signal to indicate when the bus is active, and has a guaranteed lower latency as it is point-to-point and so can never suffer from bus contention.

In cases where it is not possible to use the 4-wire interface, Dialog recommends the use of the HS-2-Wire interface rather than the standard Power Manager Interface since the HS interface can be slightly more tolerant of noise on the I2C lines.

To improve the robustness of the I2C during power up/down operations, it is recommended to utilise the PD_DIS (Power Down Disable) functionality in the DA9053 sequencer. By setting the PD_DIS step to be early in the power down sequence, and by setting the HS-2-WIRE_PD and PM-IF_PD bits in the PD_DIS register, the I2C interfaces can be disabled before any of the IO supplies are powered down, thus helping to protect against spurius edges being detected during the power down and subsequent power up procedures.

8 Conclusion

A procedure has been proposed which has proven to remove the two most common causes of DA9053 2-wire interface communication issues. Both of these issues can be worked around with the software procedure outlined above. The procedure is similar to that recommended in the I2C specification.

Revision history

Revision	Date	Description
1.0	01-Mar-2012	Initial version.
1.1	03-Jun-2015	Update to latest template

DA9053 2-Wire Implementation

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

United Kingdom (Headquarters) *Dialog Semiconductor (UK) LTD* Phone: +44 1793 757700	North America *Dialog Semiconductor Inc.* Phone: +1 408 845 8500	Singapore *Dialog Semiconductor Singapore* Phone: +65 64 8499 29	China (Shenzhen) *Dialog Semiconductor China* Phone: +86 755 2981 3669
Germany *Dialog Semiconductor GmbH* Phone: +49 7021 805-0	Japan *Dialog Semiconductor K. K.* Phone: +81 3 5425 4567	Hong Kong *Dialog Semiconductor Hong Kong* Phone: +852 3769 5200	China (Shanghai) *Dialog Semiconductor China* Phone: +86 21 5424 9058
The Netherlands *Dialog Semiconductor B.V.* Phone: +31 73 640 8822	Taiwan *Dialog Semiconductor Taiwan* Phone: +886 281 786 222	Korea *Dialog Semiconductor Korea* Phone: +82 2 3469 8200	
Email: enquiry@diasemi.com	Web site: www.dialog-semiconductor.com		