



### Introduction

This document aims to resolve some of the common misconceptions regarding the support of I<sup>2</sup>C on Linux. It provides information on the Linux kernel I<sup>2</sup>C subsystem and how to use it effectively on the 2.6.x series kernels.

- [Section 1](#) provides a brief description of the I<sup>2</sup>C protocol.
- [Section 2](#) describes the software API in detail.
- [Section 3](#) describes the efficiency of each type of message.

STLinux only supports the host running as a single I<sup>2</sup>C master, so multi-master and slave operation (although supported by the hardware) is only described where it causes complications to running as a single master.

Other hardware (for example, Marvell PXA) under Linux does support slave mode. If there is a strong customer requirement then it may be added to the ST support in future.

# Contents

- 1      The I<sup>2</sup>C protocol ..... 3**
  - 1.1    I<sup>2</sup>C bus ..... 3
  - 1.2    I<sup>2</sup>C transactions ..... 3
  - 1.3    Repeated start transactions ..... 4
  
- 2      STLinux I<sup>2</sup>C support ..... 5**
  - 2.1    Example 1: An I<sup>2</sup>C transaction writing 2 bytes of data ..... 6
  - 2.2    Example 2: An I<sup>2</sup>C transaction reading 2 bytes of data ..... 6
  - 2.3    Example 3: An I<sup>2</sup>C combined message read 1 byte then write 1 byte of data ..... 7
  
- 3      Efficient I<sup>2</sup>C transfers ..... 8**
  
- 4      Revision history ..... 9**

# 1 The I<sup>2</sup>C protocol

This section describes I<sup>2</sup>C transactions at a communications protocol level. The physical level is not described, although there is a description of a simple mapping between the protocol components and the bus-level signaling.

For more details see *The I<sup>2</sup>C Bus Specification Version 2.1*, NXP document (9398 393 40011), available for download from [http://www.nxp.com/acrobat\\_download/literature/9398/39340011.pdf](http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf).

## 1.1 I<sup>2</sup>C bus

An I<sup>2</sup>C bus consists of a number of interconnected devices. There can be one or more controlling master devices and a number of slave devices. All communication is controlled by a master device. If more than one master device is present on the bus, only one can be active at a time. Master devices may also be slaves if the hardware supports it.

Each slave device on the bus is identified by a unique device address.

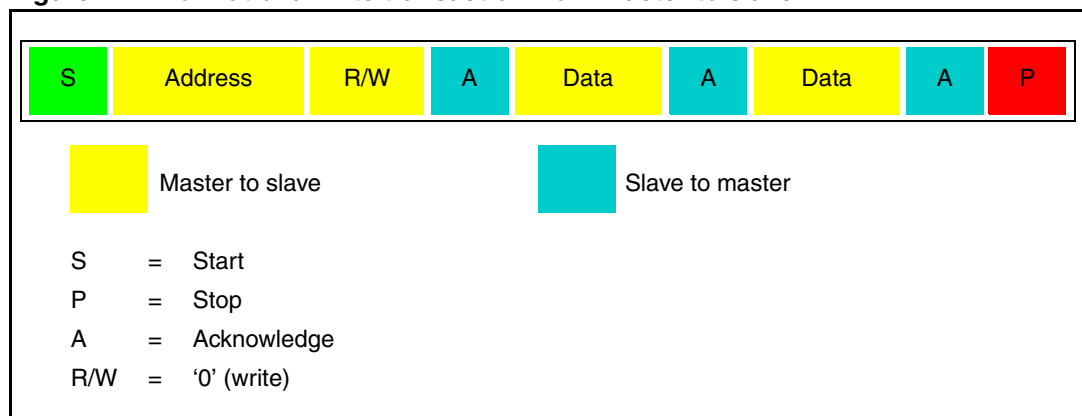
## 1.2 I<sup>2</sup>C transactions

I<sup>2</sup>C messages consist of one or more transactions. I<sup>2</sup>C transactions consist of a start, a slave address selection, a read/write selection, a data transfer and then a stop. All data transfer is acknowledged by the receiver back to the transmitter.

A data transfer can consist of one or more byte orientated read or write operations, but reads and writes cannot be mixed in a single transaction. Transactions can be combined to provide combined read and write messages (this is called a “data transfer combined format” in the I<sup>2</sup>C specification). This is described in more detail in the [Section 1.3: Repeated start transactions](#).

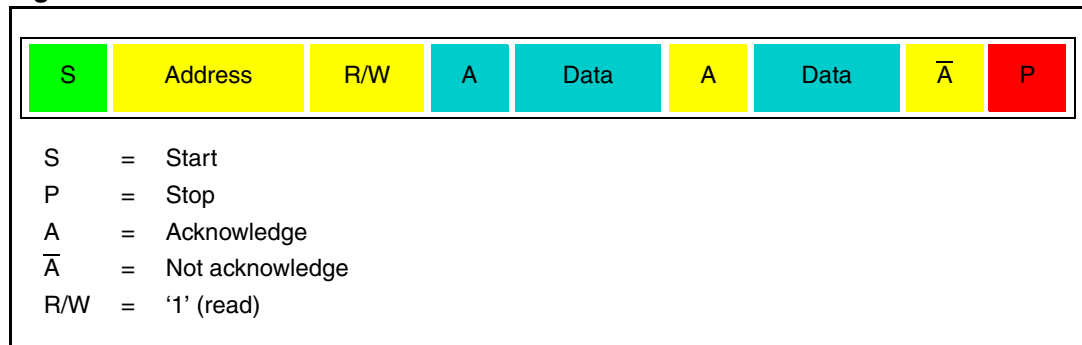
A write transaction from the master to the slave has the format displayed in [Figure 1](#).

**Figure 1. Format of a write transaction from master to slave**



A read transaction has a similar format, as shown in [Figure 2](#).

**Figure 2. Format of a read transaction from master to slave**



### 1.3 Repeated start transactions

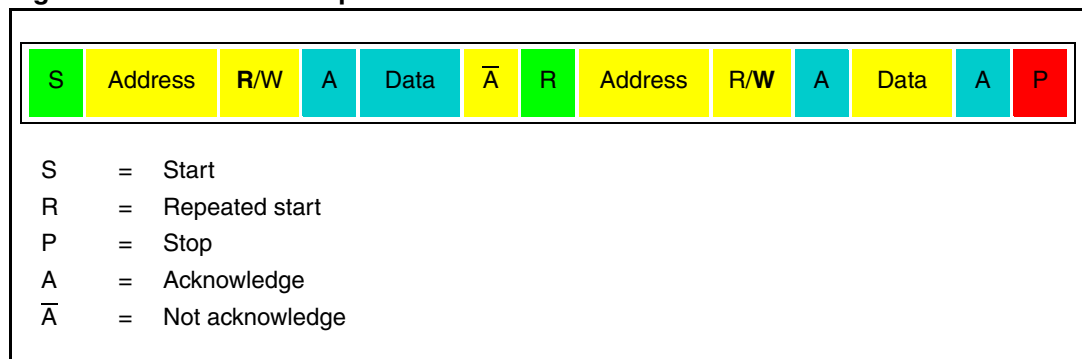
When transactions are combined, the STOP of the first transaction and the START of the second transaction can be replaced by a “repeated START” instead. This gives a small performance increase over multiple transactions as it removes the requirement for the STOP setup time.

In fast mode only, the setup time for a repeated start is half the STOP to START setup time. Therefore in standard mode (100 kHz), 4 μs per transaction is saved. In fast mode (400 kHz), 1.3 μs per transaction is saved. This represents about half a clock cycle or about 2% per single-byte transaction. For multi-byte transactions, there is no significant performance increase.

The main reason for using repeated start transactions is to lock the bus to a particular master in multi-master systems. Another less common reason is to hold an I<sup>2</sup>C repeater open.

A combined format message consisting of a single byte read and write is shown in [Figure 3](#).

**Figure 3. Format of a repeated start transaction**



*Note: The slave address is always reissued so it is possible to address different slaves (each slave resets its state on detecting a START condition). For a multiple read or write (but not both) to the **same** address it is **considerably less efficient** than using a single transaction with multiple byte transfers due to the repeated address and the setup time for the repeated start.*

## 2 STLinux I<sup>2</sup>C support

In Linux, the I<sup>2</sup>C bus can be accessed both through user-space and the kernel. This section describes the kernel interface, but the user-space interface follows similar conventions.

An I<sup>2</sup>C transaction is described by the simple structure `i2c_msg` defined in `include/linux/i2c.h`:

```
/*
 * I2C Message - used for pure I2C transaction,
 * also from /dev interface
 */

struct i2c_msg {
    __u16 addr;    /* slave address */
    __u16 flags;
    __u16 len;    /* msg length */
    __u8 *buf;    /* pointer to msg data */
};
```

The `flags` field in `i2c_msg` has the settings given in [Table 1](#).

**Table 1. Values for the flags field**

Flag	Value	Description
I2C_M_RD	0x0001	Read data
I2C_M_TEN	0x0010	Use a 10-bit I <sup>2</sup> C address
I2C_M_RECV_LEN	0x0400	Length is the first received byte
I2C_M_NO_RD_ACK	0x0800	Ignore read acknowledge
I2C_M_IGNORE_NAK	0x1000	Ignore not acknowledge
I2C_M_REV_DIR_ADDR	0x2000	Send a read flag on the bus for a write transaction or a write flag on the bus when a read is requested (This is a work around for broken I <sup>2</sup> C slave hardware)
I2C_M_NOSTART	0x4000	Do not issue any more START/address after the initial START/address in a combined message (This is a work around for broken I <sup>2</sup> C slave hardware)

Transactions are then transmitted across the bus using the kernel API:

```
int i2c_transfer(
    struct i2c_adapter *adap,
    struct i2c_msg *msgs,
    int num)
```

The Linux calls required to reproduce the example transactions given in [Section 1.2 on page 3](#) are described in C-like pseudo-code in [Section 2.1](#), [Section 2.2](#) and [Section 2.3](#). Also shown is the output from performing the operations on a real I<sup>2</sup>C bus (bus 0 on an MB411 fitted with a Cut 4 STi7109). The bus traffic was captured using a hardware I<sup>2</sup>C analyzer and

a hardware I<sup>2</sup>C slave emulator set to `slave_address=0x38` which transmitted 0xAB 0xCD on byte reads.

## 2.1 Example 1: An I<sup>2</sup>C transaction writing 2 bytes of data

```
struct i2c_msg msg[1];
unsigned char buffer[2];

msg[0].addr = slave_address;
msg[0].flags = 0;
msg[0].len = 2;
msg[0].buf = buffer;

if (i2c_transfer(client.adapter, msg, 1) < 0){
    pr_err("i2c_test: i2c_transfer failed\n");
}
```

### Analyzer output:

```
9.27ms      2      SP      W      38      00      00
```

This means a transaction of duration 9.27 ms, 2 bytes, start and stop detected, write, slave address 0x38, data 0x00 0x00 was detected on the bus.

## 2.2 Example 2: An I<sup>2</sup>C transaction reading 2 bytes of data

```
struct i2c_msg msg[1];
unsigned char buffer[2];

msg[0].addr = slave_address;
msg[0].flags = I2C_M_RD;
msg[0].len = 2;
msg[0].buf = buffer;

if (i2c_transfer(client.adapter, msg, 1) < 0){
    pr_err("i2c_test: i2c_transfer failed\n");
}
```

### Analyzer output:

```
35.0ms      2      SP      R      38      AB      CD*
```

This means a transaction of duration 35 ms, 2 bytes, start and stop detected, read, slave address 0x38, data 0xAB 0xCD with NACK (the \*) was detected on the bus.

## 2.3 Example 3: An I<sup>2</sup>C combined message read 1 byte then write 1 byte of data

```
struct i2c_msg msg[2];
unsigned char buffer[2];

msg[0].addr = slave_address;
msg[0].flags = I2C_M_RD;
msg[0].len = 1;
msg[0].buf = buffer;

msg[1].addr = slave_address;
msg[1].flags = 0;
msg[1].len = 1;
msg[1].buf = buffer;

if (i2c_transfer(client.adapter, msg, 2) < 0){
    pr_err("i2c_test: i2c_transfer failed\n");
}
```

### Analyzer output:

19.4ms	1	S	R	38AB*
16.0ms	1	SP	W	38 AB

The meaning of this transaction is as described in the previous examples.

*Note:* The first transaction is **NOT** followed by a STOP (P) as combined messages automatically use repeated start transactions.

### 3 Efficient I<sup>2</sup>C transfers

The I<sup>2</sup>C bus is a slow external serial bus (100 kHz in Standard Mode, 400 kHz in Fast Mode). [Table 2](#) shows the efficiency of each type of message.

**Table 2. Message efficiency**

Message	Efficiency	Reason
A combined message containing transactions with a large amount of data in each transaction.	Highest	Multiple data requires no additional START/address to be transmitted. Multiple transactions in a combined message use a repeated start.
Single transactions containing large amount of data.	High	Multiple data requires no additional START/address to be transmitted but a single transaction results in a STOP/START time gap.
A combined message with small amount of data in each transaction.	Poor	More instances of START/address need to be transmitted but at least a repeated start is used.
Single transactions containing a single byte of data.	Very Poor	STOP transmitted at each transaction so STOP/START time gap required before next transaction. Many instances of probably redundant START/address need to be transmitted.

These guidelines are true for all transactions on an I<sup>2</sup>C bus and hence have no reliance on any particular operating system, in other words, it is valid for OS21 as well as Linux.

If a device driver is using I<sup>2</sup>C accesses of the same type (read or write), a small number of bytes and the same slave address then the available bandwidth on the bus is small due to the high protocol overhead. In fact, only about 42% of bits transmitted on the bus are the actual data in the case of single byte transfers. Taking timing into consideration, the available bits per second bus bandwidth will be <30% compared to single transactions containing large amounts of data.



## 4 Revision history

**Table 3. Document revision history**

Date	Revision	Changes
12-Aug-2008	A	Initial release.

**Please Read Carefully:**

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

**UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.**

**UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.**

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2008 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

[www.st.com](http://www.st.com)