



StarFive
赛昉科技

Software SDK Developer Guide for I2C

VisionFive 2

Version: 1.0

Date: 2022/11/10

Doc ID: JH7110-DGEN-009

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Copyright © Shanghai StarFive Technology Co., Ltd., 2022. All rights reserved.

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

All material appearing in this document is protected by copyright and is the property of StarFive. You may not reproduce the information contained herein, in whole or in part, without the written permission of StarFive.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: <http://www.starfivetech.com>

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com



StarFive Technology
星五科技

Contents

List of Tables.....	4
List of Figures.....	5
Legal Statements.....	ii
Preface.....	vi
1. Introduction.....	7
1.1. Function Introduction.....	7
1.2. Block Diagram.....	7
1.3. Device Tree Overview.....	8
1.4. Source Code Structure.....	9
2. Configuration.....	10
2.1. Kernel Menu Configuration.....	10
2.2. Device Tree Source Code.....	12
2.3. Device Tree Configuration.....	13
2.4. Board Level Configuration.....	14
3. Interface Description.....	15
3.1. i2c_add_adapter.....	15
3.2. i2c_new_client_device.....	15
3.3. i2c_register_driver.....	15
3.4. i2c_transfer_buffer_flags.....	16
3.5. i2c_transfer.....	16
4. General Use Example.....	17
4.1. I2C Driver Example.....	17
4.2. Commands in User Space.....	19

List of Tables

Table 0-1 Revision History.....vi



List of Figures

Figure 1-1 Block Diagram.....	8
Figure 1-2 Device Tree Workflow.....	9
Figure 2-1 Device Drivers.....	10
Figure 2-2 I2C Support.....	11
Figure 2-3 I2C Hardware Bus Support.....	11
Figure 2-4 Synopsys DesignWare Platform.....	12



Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programming basics and debugging know-how for the I2C of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the I2C relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.






Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0		The first official release.

Notes and notices

The following notes and notices might appear in this guide:

-  **Tip:**
Suggests how to apply the information in a topic or step.
-  **Note:**
Explains a special case or expands on an important point.
-  **Important:**
Points out critical information concerning a topic or step.
-  **CAUTION:**
Indicates that an action or step can cause loss of data, security problems, or performance issues.
-  **Warning:**
Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

Inter-Integrated Circuit (I2C) bus is a two-wire serial bus developed by the company of Philips. The module is used to connect micro controller and peripheral devices.

I2C contains the following 2 lines.

- *Serial Data (SDA)*: This line works for the data transmission.
- *Serial Clock (SCL)*: This line works for the clock synchronization.

StarFive JH7110 SoC has integrated a I2C bus adapter from third party IP vendor. The device driver of I2C bus adapter can be found in the mainline code of Linux.

1.1. Function Introduction

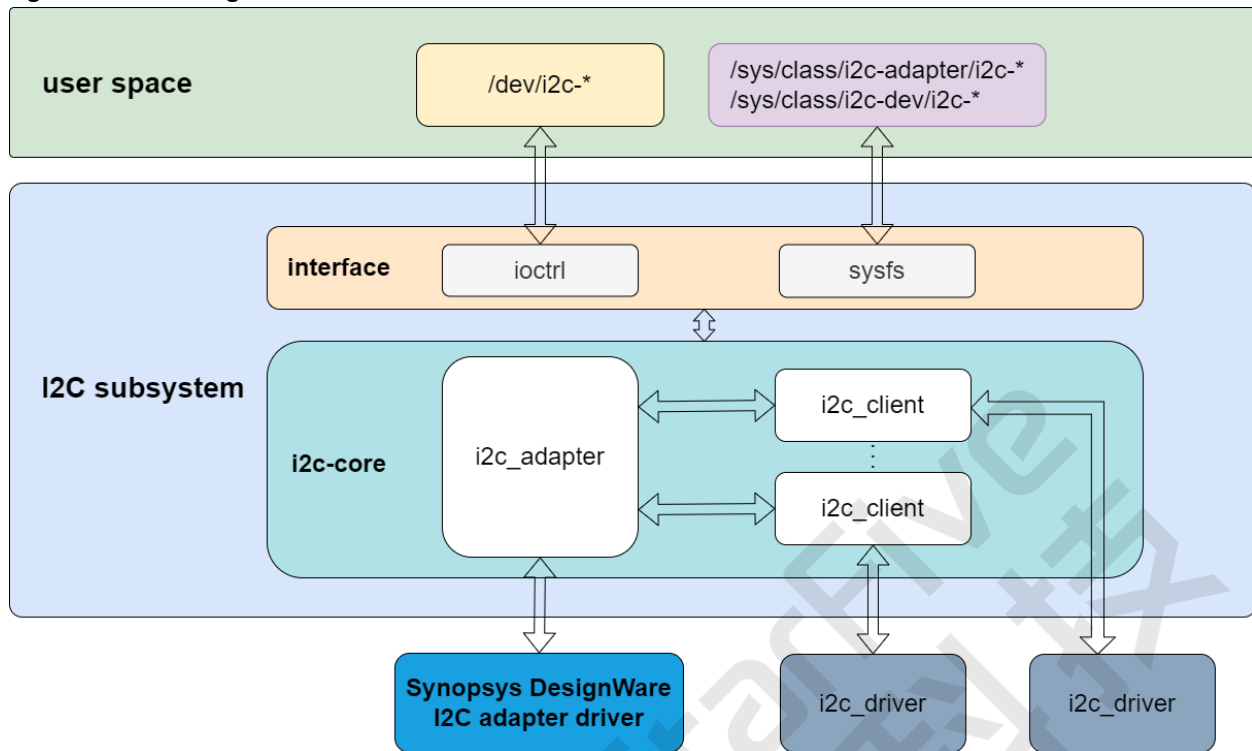
JH7110 SoC Platform supports the following features and specifications on the I2C module.

- Support the feature of Device ID
- Provide 7 Independent transmission channels
- Support master and slave I2C operation
- Support 7-bit and 10-bit slave addressing
- Support the following modes and transmission speed:
 - Standard mode - 100 KHz
 - Fast mode - 400 KHz
 - Fast mode plus - 1 MHz
 - High speed mode - 3.4 MHz
- Support programmable SDA hold time

1.2. Block Diagram

The block diagram of I2C driver is shown in the following figure.

Figure 1-1 Block Diagram



The user space layer is general for all Linux system modules.

The I2C subsystem includes the following sub-modules.

- **interface:** The sub-module provides interfaces to interact with `i2c_adapter` or `i2c_client`.
- **i2c-core:** The sub-module builds the core functions of I2C subsystem. It provides `i2c_adapter` and `i2c_client` APIs.
- **Synopsys DesignWare I2C adapter driver:** The sub-module registers the `i2c_adapter` and provide functions for receive/send operations.
- **i2c driver:** The sub-module uses application interfaces to receive/send data.

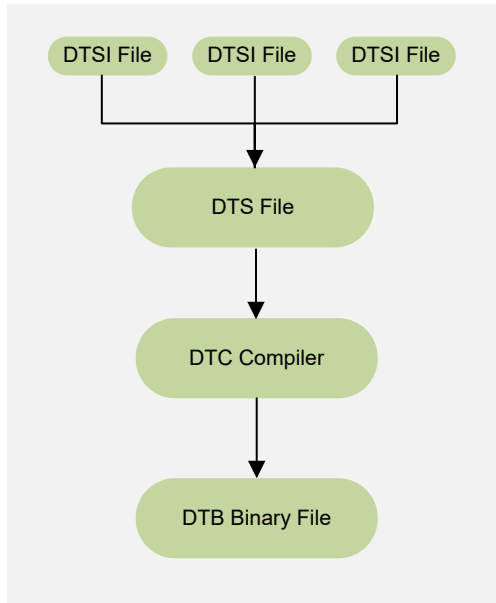
1.3. Device Tree Overview

Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

A device tree is primarily represented in the following forms.

- **Device Tree Compiler (DTC):** The tool used to compile device tree into system-readable binaries.
- **Device Tree Source (DTS):** The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.
- **Device Tree Source Information (DTSI):** The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- **Device Tree Blob (DTB):** The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

Figure 1-2 Device Tree Workflow

1.4. Source Code Structure

The source code structure of I2C is listed in the following code blocks.

The following code block displays the source code structure for the I2C Subsystem layer.

```

├─ drivers
│  └─ i2c
│     ├── i2c-core.h
│     ├── i2c-boardinfo.c
│     ├── i2c-core-of.c
│     ├── i2c-core-slave.c
│     ├── i2c-mux.c
│     ├── i2c-core-acpi.c
│     ├── i2c-core-smbus.c
│     ├── i2c-smbus.c
│     ├── i2c-core-base.c
│     └─ i2c-dev.c
  
```

The following code block displays the source code structure for the Synopsys DesignWare I2C adapter driver.

```

linux-5.15.0
├─ drivers
│  └─ i2c
│     └─ busses
│        ├── i2c-designware-core.h
│        ├── i2c-designware-common.c
│        ├── i2c-designware-master.c
│        ├── i2c-designware-slave.c
│        ├── i2c-designware-pcidrv.c
│        ├── i2c-designware-baytrail.c
│        └─ i2c-designware-platdrv.c
  
```

2. Configuration

2.1. Kernel Menu Configuration

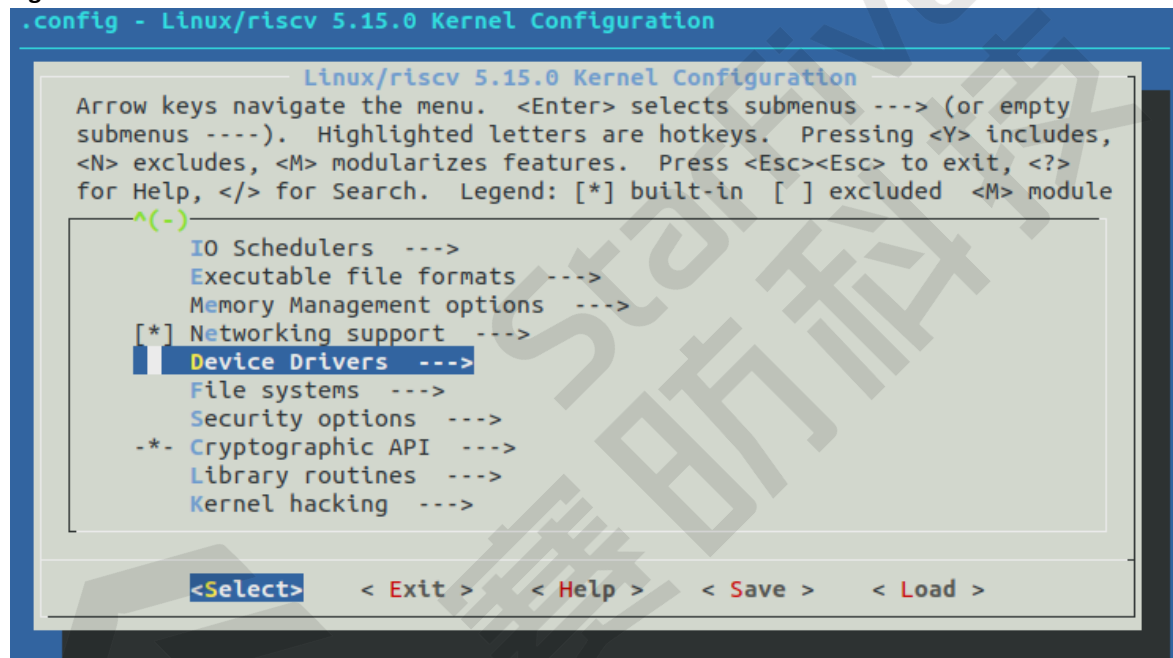
Follow the steps below to enable the kernel configuration for I2C.

1. Under the root directory of `freelight-u-sdk`, type the following command to enter the kernel menu configuration GUI.

```
make linux-menuconfig
```

2. Enter the **Device Drivers** menu.

Figure 2-1 Device Drivers



3. Enter the **I2C support** menu.

Figure 2-2 I2C Support

```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers
  Device Drivers
  Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
  submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes,
  <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?>
  for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module
  ^(-)
  IEEE 1394 (FireWire) support --->
  [*] Network device support --->
  Input device support --->
  Character devices --->
  I2C support --->
  < > I3C support ----
  [*] SPI support ----
  < > SPMI support ----
  < > HSI support ----
  < > PPS support ----
  ↓(+)
```

< Select > < Exit > < Help > < Save > < Load >

4. Enter the I2C Hardware Bus Support menu.

Figure 2-3 I2C Hardware Bus Support

```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers > I2C support
  I2C support
  Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
  submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes,
  <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?>
  for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module
  -* I2C support
  [*] Enable compatibility bits for old user-space
  <*> I2C device interface
  < > I2C bus multiplexing support
  [*] Autoselect pertinent helper modules
  I2C Hardware Bus support --->
  < > I2C/SMBus Test Stub
  [ ] I2C slave support
  [ ] I2C Core debugging messages
  [ ] I2C Algorithm debugging messages
  ↓(+)
```

< Select > < Exit > < Help > < Save > < Load >

5. Select the Synopsys DesignWare Platform option.

Figure 2-4 Synopsys DesignWare Platform

```

.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers > I2C support > I2C Hardware Bus support
      I2C Hardware Bus support
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes,
<N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?>
for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module
^(-)
< > VIA VT82C586B
< > VIA VT82C596/82C686/82xx and CX700/VX8xx/VX900
    *** I2C system bus drivers (mostly embedded / system-on-chip) **
< > CBUS I2C driver
[ ] Synopsys DesignWare Slave
[*] Synopsys DesignWare Platform
< > Synopsys DesignWare PCI
< > EMMA Mobile series I2C adapter
< > GPIO-based bitbanging I2C
< > ST-Ericsson Nomadik/Ux500 I2C Controller
+(-)
<Select> < Exit > < Help > < Save > < Load >

```

6. Save your change before you exit the kernel configuration dialog.

2.2. Device Tree Source Code

Overview Structure

The device tree source code of JH7110 is listed as follows:

```

linux
├── arch
│   ├── riscv
│   │   ├── boot
│   │   └── dts
│   │       ├── starfive
│   │       │   ├── codecs
│   │       │   │   ├── sf_pdm.dtsi
│   │       │   │   ├── sf_pwm dac.dtsi
│   │       │   │   ├── sf_spdif.dtsi
│   │       │   │   ├── sf_tdm.dtsi
│   │       │   │   └── sf_wm8960.dtsi
│   │       │   ├── evb-overlay
│   │       │   │   ├── jh7110-evb-overlay-can.dts
│   │       │   │   ├── jh7110-evb-overlay-rgb2hdmi.dts
│   │       │   │   ├── jh7110-evb-overlay-sdio.dts
│   │       │   │   ├── jh7110-evb-overlay-spi.dts
│   │       │   │   ├── jh7110-evb-overlay-uart4-emmc.dts
│   │       │   │   └── jh7110-evb-overlay-uart5-pwm.dts
│   │       │   └── Makefile
│   │       ├── jh7110-clk.dtsi
│   │       ├── jh7110-common.dtsi
│   │       ├── jh7110.dtsi
│   │       ├── jh7110-evb-can-pdm-pwm dac.dts
│   │       ├── jh7110-evb.dts
│   │       ├── jh7110-evb.dtsi
│   │       ├── jh7110-evb-dvp-rgb2hdmi.dts
│   │       ├── jh7110-evb-pcie-i2s-sd.dts
│   │       ├── jh7110-evb-pinctrl.dtsi
│   │       ├── jh7110-evb-spi-uart2.dts
│   │       ├── jh7110-evb-uart1-rgb2hdmi.dts
│   │       └── jh7110-evb-uart4-emmc-spdif.dts

```

```

| | | | |   └─ jh7110-evb-uart5-pwm-i2c-tdm.dts
| | | | |   └─ jh7110-fpga.dts
| | | | |   └─ jh7110-visionfive-v2.dts
| | | | |   └─ Makefile
| | | | |   └─ vf2-overlay
| | | | |       └─ Makefile
| | | | |       └─ vf2-overlay-uart3-i2c.dts

```

SoC Platform

The device tree source code of the JH7110 SoC platform is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

VisionFive 2

The device tree source code of the VisionFive 2 *Single Board Computer (SBC)* is in the following path:

```

freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-common.dtsi
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi

```

2.3. Device Tree Configuration

The I2C device tree source code is stored in the files of `jh7110.dts` and `jh7110-common.dtsi`.

The following code blocks provide more details by taking the node "i2c0" as example.

In the file `jh7110.dts`:

```

i2c0: i2c@10030000 {
    compatible = "snps,designware-i2c";
    reg = <0x0 0x10030000 0x0 0x10000>;
    clocks = <&clkgen JH7110_I2C0_CLK_CORE>,
            <&clkgen JH7110_I2C0_CLK_APB>;
    clock-names = "ref", "pclk";
    resets = <&rstgen RSTN_U0_DW_I2C_APB>;
    interrupts = <35>;
    #address-cells = <1>;
    #size-cells = <0>;
    status = "disabled";
};

```

The following list provides explanations for the parameters included in the above code block.

- **compatible**: Compatibility information, used to associate the driver and its target device.
- **reg**: Register base address "0x10030000" and range "0x10000".
- **clocks**: The clocks used by the I2C module.
- **clock-names**: The names of the above clocks.
- **resets**: The reset signals used by the I2C module.
- **reset-names**: The names of the above reset signals.
- **interrupts**: Hardware interrupt ID.
- **status**: The work status of the I2C, "enabled" or "disabled".

In the file `jh7110-common.dtsi`:

```

&i2c0 {
    clock-frequency = <100000>;
    i2c-sda-hold-time-ns = <300>;
    i2c-sda-falling-time-ns = <510>;
    i2c-scl-falling-time-ns = <510>;
    auto_calc_scl_lhcnt;
    pinctrl-names = "default";
};

```

| 2 - Configuration

```
pinctrl-0 = <&i2c0_pins>;
status = "disabled";

ac108_a: ac108@3b {
    compatible = "x-power,ac108_0";
    reg = <0x3b>;
    #sound-dai-cells = <0>;
    data-protocol = <0>;
};

wm8960: codec@1a {
    compatible = "wlf,wm8960";
    reg = <0x1a>;
    #sound-dai-cells = <0>;

    wlf,shared-lrclk;
};
```

The following list provides explanations for some parameters included in the above code block.

- **clock-frequency**: Use this bit to set the I2C sample rate.
- **i2c-sda-hold-time-ns**: Use this bit to set the hold time of the SDA line.
- **i2c-sda-falling-time-ns**: Use this bit to set the falling time of the SDA line.
- **i2c-scl-falling-time-ns**: Use this bit to set the falling time of the SCL line.

2.4. Board Level Configuration

The pinctrl.dtsi file contains the pin control configuration. The file is stored in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
```

The following code block provides an example of the pins used by i2c0, including **pinmux** (Pin MUX), **ioconfig** (pin control configuration), **dout** (data output), **doen** (data output enable) and **din** (data input) signals.

```
i2c0_pins: i2c0-pins {
    i2c0-pins-scl {
        sf,pins = <PAD_GPIO57>;
        sf,pinmux = <PAD_GPIO57_FUNC_SEL 0>;
        sf,pin-ioconfig = <IO(GPIO_IE(1) | (GPIO_PU(1)))>;
        sf,pin-gpio-dout = <GPO_LOW>;
        sf,pin-gpio-doen = <OEN_I2C0_IC_CLK_OE>;
        sf,pin-gpio-din = <GPI_I2C0_IC_CLK_IN_A>;
    };

    i2c0-pins-sda {
        sf,pins = <PAD_GPIO58>;
        sf,pinmux = <PAD_GPIO58_FUNC_SEL 0>;
        sf,pin-ioconfig = <IO(GPIO_IE(1) | (GPIO_PU(1)))>;
        sf,pin-gpio-dout = <GPO_LOW>;
        sf,pin-gpio-doen = <OEN_I2C0_IC_DATA_OE>;
        sf,pin-gpio-din = <GPI_I2C0_IC_DATA_IN_A>;
    };
};
```

3. Interface Description

The I2C module of JH7110 has the following interfaces.

3.1. i2c_add_adapter

The interface has the following parameters.

- **Synopsis:**

```
int i2c_add_adapter(struct i2c_adapter *adapter)
```

- **Description:** The interface is used to declare the I2C adapter by using a dynamic bus ID.

- **Parameter:**

- **adapter:** The I2C adapter to add.

- **Return:**

- **Success:** 0.

- **Failure:** Error code.

3.2. i2c_new_client_device

The interface has the following parameters.

- **Synopsis:**

```
struct i2c_client *i2c_new_client_device(struct i2c_adapter *adap, struct i2c_board_info const *info)
```

- **Description:** The interface is used to instantiate an I2C client device.

- **Parameter:**

- **adapter:** The adapter which manages the I2C client device.

- **info:** The description information of the I2C client device.

- **Return:** The pointer to the new I2C client device.

3.3. i2c_register_driver

The interface has the following parameters.

- **Synopsis:**

```
int i2c_register_driver(struct module *owner, struct i2c_driver *driver)
```

- **Description:** The interface is used to register the I2C driver.

- **Parameter:**

- **owner:** The sub-module to which the I2C driver belongs.

- **driver:** The I2C driver to add.

- **Return:**

- **Success:** 0.

- **Failure:** Error code.

3.4. i2c_transfer_buffer_flags

The interface has the following parameters.

- **Synopsis:**

```
int i2c_transfer_buffer_flags(const struct i2c_client *client, char *buf, int count, u16 flags)
```

- **Description:** The interface is used to issue a single I2C message which transfers data to/from a buffer.

- **Parameter:**

- **client:** The handle to the client device.
- **buf:** The buffer in which data is stored.
- **count:** The count of bytes to transfer.



- **Note:**

The parameter should be less than 64 K since **msg.len** is u16.

- **flags:** The flags to be used for the message, e.g. I2C_M_RD for reads.

- **Return:**

- **Success:** The count of bytes which are transferred successfully.
- **Failure:** Error code.

3.5. i2c_transfer

The interface has the following parameters.

- **Synopsis:**

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

- **Description:** The interface is used to execute a single or combined I2C message.

- **Parameter:**

- **adap:** The handle to the I2C bus.
- **msgs:** The parameter includes one or more than one messages to execute before the flag of "STOP" is issued to terminate the operation; Each message begins with a flag of "START".
- **num:** The total number of messages to be executed.

- **Return:**

- **Success:** The total number of messages which are executed successfully.
- **Failure:** Error code.

4. General Use Example

In most cases, the I2C adapter works in master mode. Before you start transferring data in your I2C driver, you need to finish some configuration.

1. Set a suitable sample rate of I2C bus through the device tree according to your client device requirement.

The following code blocks provide some commonly used configurations.

To set the sample rate as 100 KHz.

```
&i2c0 {
    clock-frequency = <100000>;
    i2c-sda-hold-time-ns = <300>;
    i2c-sda-falling-time-ns = <510>;
    i2c-scl-falling-time-ns = <510>;
}
```

To set the sample rate as 400 KHz.

```
&i2c0 {
    clock-frequency = <400000>;
    i2c-sda-hold-time-ns = <300>;
    i2c-sda-falling-time-ns = <150>;
    i2c-scl-falling-time-ns = <150>;
}
```

2. Mount your I2C client device on a specified adapter through the device tree. The property **compatible** has to match your I2C driver and **reg** is set as the I2C slave address.

The following code block shows a simple case of **ac108** mounted on adapter **i2c0**.

```
&i2c0 {
    ac108_a: ac108@3b {
        compatible = "x-power,ac108_0";
        reg = <0x3b>;
        #sound-dai-cells = <0>;
        data-protocol = <0>;
    };
}
```

4.1. I2C Driver Example

The following demo shows how to implement a I2C driver and transfer data using I2C APIs of Linux. In this demo, "xxx" represents the name of the I2C client device.

```
static int write_reg(struct i2c_client *client, u16 reg, u8 val)
{
    struct i2c_msg msg;
    u8 buf[3];
    int ret;

    buf[0] = reg >> 8;
    buf[1] = reg & 0xff;
    buf[2] = val;

    msg.addr = client->addr;
    msg.flags = client->flags;
    msg.buf = buf;
    msg.len = sizeof(buf);

    ret = i2c_transfer(client->adapter, &msg, 1);
    if (ret < 0) {
        dev_err(&client->dev, "%s: error: reg=%x, val=%x\n",
            __func__, reg, val);
        return ret;
    }
}
```

```

    return 0;
}

static int read_reg(struct i2c_client *client, u16 reg, u8 *val)
{
    struct i2c_msg msg[2];
    u8 buf[2];
    int ret;

    buf[0] = reg >> 8;
    buf[1] = reg & 0xff;

    msg[0].addr = client->addr;
    msg[0].flags = client->flags;
    msg[0].buf = buf;
    msg[0].len = sizeof(buf);

    msg[1].addr = client->addr;
    msg[1].flags = client->flags | I2C_M_RD;
    msg[1].buf = buf;
    msg[1].len = 1;

    ret = i2c_transfer(client->adapter, msg, 2);
    if (ret < 0) {
        dev_err(&client->dev, "%s: error: reg=%x\n",
            __func__, reg);
        return ret;
    }

    *val = buf[0];
    return 0;
}

//...

//use function write_reg(), read_reg() to operate registers

//...

//Achieve function probe_new() and remove() of i2c_driver
static int xxx_probe(struct i2c_client *client);
static int xxx_remove(struct i2c_client *client);

static const struct i2c_device_id xxx_id[] = {
    {"xxx", 0},
    {},
};
MODULE_DEVICE_TABLE(i2c, xxx_id);

static const struct of_device_id xxx_dt_ids[] = {
    { .compatible = "xxx" },
    { /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, xxx_dt_ids);

static struct i2c_driver xxx_i2c_driver = {
    .driver = {
        .name = "xxx",
        .of_match_table = xxx_dt_ids,
    },
    .id_table = xxx_id,
    .probe_new = xxx_probe,
    .remove = xxx_remove,
};

module_i2c_driver(xxx_i2c_driver);

```

4.2. Commands in User Space

Linux provides some useful I2C related commands in the file system helping us control I2C bus.

The following code blocks provide some simple examples.

- To detect all the registered I2C adapters:

```
i2cdetect -l
```

- To detect all the client devices mounted on the specified I2C adapter:

```
i2cdetect -y -r [i2c_adapter_index]
```

- To read the address from the specified I2C client:

```
i2cget -y [i2c_adapter_index] [i2c_client address] [address]
```

- To write the address to the specified I2C client:

```
i2cset -y [i2c_adapter_index] [i2c_client address] [address] [value]
```