**SHD–LPC to I2C Adapter Driver**

© 2022 Inspur. All rights reserved.

This program is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation; either version 2 of the License, or

(at your option) any later version.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev.** | **Date** | **Author** | **Description** |
| 0.1 | 2022/11/10 | liwenlong02@inspur.com | Initial version; |
| 0.2 | 2022/11/18 | liwenlong02@inspur.com | 修改状态机流程图状态变化说明;修改irq流程图错误及其他等; |
| 0.3 | 2023/01/30 | liwenlong02@inspur.com | 1.修改驱动源文件和驱动ko模块文件名字与实际相符;  2.补充状态机变迁图详细说明; |
| 1.0 | 2023/05/18 | liwenlong02@inspur.com | 1.状态机增加从STATE\_DONE到STATE\_STOP状态变迁机制;  2.状态机变迁说明改为A/B模式,其中A代表变迁的条件描述;B代表当前状态下做的动作;  3.状态机中STATE\_ERROR从变迁图中移除,以凸显其他重要变迁;  4.增加I2C协议内容; |
| 1.0 | 2023/7/19 | liwenlong02@ieisystem.com | 1.对文中示图增加编号, 如:图3-1 硬件原理图;  2.增加polling模式原理说明;  3.增加部分说明使文档看着更连贯；修改文档中部分错别字; |
| 1.0 | 2024/01/15 | 张连聘 | 修正了图片 《图3-2 用户空间/内核空间与I2C Adapter关系》 并补充了概念解释 |

Table of Content

[Revision History 2](#_Toc141686672)

[Section 1: About this document 4](#_Toc141686673)

[Section 2: Functional Spec 4](#_Toc141686674)

[Section 2.1: Brief Description 4](#_Toc141686675)

[Section 3: Architecture 5](#_Toc141686676)

[Section 4: Flow 7](#_Toc141686677)

[Section 5: Develop of Debug 14](#_Toc141686678)

[Section 6: Database 15](#_Toc141686679)

[Section 7: CLI 15](#_Toc141686680)

[Section 7.1: SONiC-CLI 15](#_Toc141686681)

[Section 7.2: inCLI 15](#_Toc141686682)

[Section 8: Security 15](#_Toc141686683)

[Section 8.1: Securing data 15](#_Toc141686684)

[Section 8.2: Security protocol 16](#_Toc141686685)

[Section 8.3: Security Algorithm 16](#_Toc141686686)

[Section 8.4: Third Party Open Source Component 16](#_Toc141686687)

[Section 9: Public APIs 16](#_Toc141686688)

[Section 10: Function Test 17](#_Toc141686689)

[Section 11: Appendix 17](#_Toc141686690)

# About this document

文档详细介绍了LPC to 4路I2C Adapter 的驱动设计方案。使用LPC 转多路I2C 的原因是扩展CPU 的I2C 控制器数量，属于高速总线转低速总线的常规方案。

LPC (Low Pin Count Bus)

I2C-OCores: https://opencores.org/projects/i2c/

# Functional Spec

|  |  |
| --- | --- |
| Function | Function points |
| 支持Polling mode 和interrupt mode 切换 |  |
| 支持I2C i2c transactions 和 smbustransactions |  |
| 支持多设备（多个I2C Adapter controller） |  |
| 四个I2C Adapter共享一个GPIO中断 |  |
| 支持Linux-4.19 |  |

## Brief Description

1. Polling 和Interrupt：
   1. 支持的polling 和中断模式通过内核模块的参数控制（module\_param）
   2. I2C-OCores 提供了I2C START、I2C STOP、I2C ACK这种low level的操作，需要驱动处理I2C 通讯协议细节。因此Polling mode下需要高实时（10us）读取I2C-OCores提供的状态寄存器。带来的问题是CPU 利用率高。所以要求实现interrupt mode。
2. 支持I2C i2c transactions 和 smbustransactions
   1. 实现kernel i2c submodule system i2c\_algorithm要求的master\_xfer和smbus\_xfer

I2C\_FUNC\_I2C | I2C\_FUNC\_SMBUS\_EMUL

1. 支持多设备（多个I2C Adapter controller）
   1. 多个I2C-Ocores I2C Adapter controller可以复用此驱动；
   2. I2C Adapter controller可以分布到不同的硬件实体上，比如CPLD、FPGA；
   3. 设备的管理和驱动分离，设备的创建、删除由独立的模块负责。
2. 四个I2C Adapter共享一个GPIO中断
   1. 四个I2C Adapter 使用相同的总中断控制寄存器来控制总中断的使能和禁用；
   2. 共享中断模式下，具体中断源经过二次查找I2C Adapter 对应的寄存器确定来确定。一个中断内可能有1---N个I2C Adapter的事务需要处理。

# Architecture

1. HW block diagram（硬件框图，CPU—LPC---CPLD---I2C）



图3-1 硬件原理图

说明：CPU和PORT\_CPLDx是LPC通信，PORT\_CPLD flash内实现I2C Adapter 实现LPC to I2C通信，驱动功能就是控制I2C通信时序。中断模式下，I2C Adapter REG状态发生变化，通过interrupt 上拉GPIO CPU收到中断请求，CPU进入handle拉低interrupt GPIO电平处理对应I2C Adapter请求。

1. SW diagram

UserSpace ---Kernel Space---I2C Adapter的关系

* + 1. I2C subsystem
    2. Platform subsystem
    3. Interrupt subsystem



图3-2 用户空间/内核空间与I2C Adapter关系

引用社区文档：

**I2C Core**

The I2C core is a code base consisting of routines and data structures available to host adapter drivers and client drivers. Common code in the core makes the driver developer's job easier. The core also provides a level of indirection that renders client drivers independent of the host adapter, allowing them to work unchanged even if the client device is used on a board that has a different I2C host adapter. This philosophy of a core layer and its attendant benefits is also relevant for many other device driver classes in the kernel, such as PCMCIA, PCI, and USB.

In addition to the core, the kernel I2C infrastructure consists of the following:

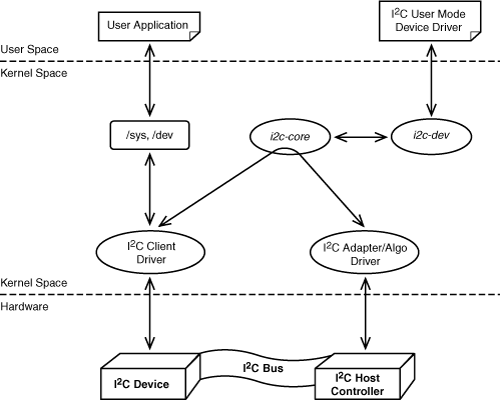
* Device drivers for I2C host adapters. They fall in the realm of bus drivers and usually consist of an *adapter* driver and an *algorithm* driver. The former uses the latter to talk to the I2C bus.
* Device drivers for I2C client devices.
* i2c-dev, which allows the implementation of user mode I2C client drivers.

You are more likely to implement client drivers than adapter or algorithm drivers because there are a lot more I2C devices than there are I2C host adapters. So, we will confine ourselves to client drivers in this chapter.

[Figure 8.2](http://www.embeddedlinux.org.cn/EssentialLinuxDeviceDrivers/final.../ch08lev1sec2.html#ch08fig02) illustrates the Linux I2C subsystem. It shows I2C kernel modules talking to a host adapter and a client device on an I2C bus.

**Figure 8.2. The Linux I2C subsystem.**

[[View full size image]](http://www.embeddedlinux.org.cn/EssentialLinuxDeviceDrivers/final.../images/YTNyaWQ3ODBzOS9jL2VnbXRwNjQ5NWEzMi9yZzE1ZmZpMHBhdGxfMC8yaGlnOGNzZ2ku.jpg)



Because SMBus is a subset of I2C, using only SMBus commands to talk to your device yields a driver that works with both SMBus and I2C adapters. [Table 8.1](http://www.embeddedlinux.org.cn/EssentialLinuxDeviceDrivers/final.../ch08lev1sec2.html#ch08tab01) lists the SMBus-compatible data transfer routines provided by the I2C core.

| **Table 8.1. SMBus-Compatible Data Access Functions Provided by the I2C Core** | |
| --- | --- |
| **Function** | **Purpose** |
| i2c\_smbus\_read\_byte() | Reads a single byte from the device without specifying a location offset. Uses the same offset as the previously issued command. |
| i2c\_smbus\_write\_byte() | Sends a single byte to the device at the same memory offset as the previously issued command. |
| i2c\_smbus\_write\_quick() | Sends a single bit to the device (in place of the Rd/Wr bit shown in [Listing 8.1](http://www.embeddedlinux.org.cn/EssentialLinuxDeviceDrivers/final.../ch08lev1sec3.html#ch08ex01)). |
| i2c\_smbus\_read\_byte\_data() | Reads a single byte from the device at a specified offset. |
| i2c\_smbus\_write\_byte\_data() | Sends a single byte to the device at a specified offset. |
| i2c\_smbus\_read\_word\_data() | Reads 2 bytes from the specified offset. |
| i2c\_smbus\_write\_word\_data() | Sends 2 bytes to the specified offset. |
| i2c\_smbus\_read\_block\_data() | Reads a block of data from the specified offset. |
| i2c\_smbus\_write\_block\_data() | Sends a block of data (<= 32 bytes) to the specified offset. |

1. 模块名称

|  |  |
| --- | --- |
| No. | Module Name |
| 1 | inspur\_i2c\_ocores\_tiger.ko |
| 2 | inspur\_lpc\_platform.ko |
| … |  |

驱动模块包含inspur\_i2c\_ocores\_tiger.ko inspur\_lpc\_platform.ko两部分.

inspur\_lpc\_platform.ko功能是注册设备，使用的是

platform\_device\_register()

inspur\_i2c\_ocores\_tiger.ko 是I2C master adapter设备的驱动,注册驱动函数i2c\_adapter\_drv\_probe ();

驱动模块加载时会根据id\_table赋值的参数列表中设备的name自动执行这个驱动文件.

i2c\_adapter\_drv\_probe (struct platform\_device \*pdev)函数是驱动加载的核心函数,函数首先根据驱动传入参数判断i2c master adapterX是polling还是interrupt工作模式。

参数定义如下:

#define PARAM\_HELP \

"enable select driver features:\n"\

" \t\tnothing parameter used is the default that it use polling mode.\n"\

" \t\tenable\_features=0x01 use polling mode.\n"\

" \t\tenable\_features=0x02 use interrupts mode.\n"

static unsigned int enable\_features;

module\_param(enable\_features, uint, 0644);

MODULE\_PARM\_DESC(enable\_features, PARAM\_HELP);

此模块使用中断模式时，使用request\_irq()函数注册中断处理程序。

驱动文件包含2个\*.c文件和一个\*.h文件:

|  |  |
| --- | --- |
| No. | Module Name |
| 1 | i2c\_ocores\_tiger.c.c |
| 2 | lpc\_platform.c |
| 3 | i2c\_ocores.h |
| … |  |

I2c master adapterX驱动依赖i2c-dev,内核代码已经包含这个模块,安装指令如下:

modprobe i2c-dev

安装lpc-i2c驱动设备文件指令: insmod ./inspur\_lpc\_platform.ko

lpc-i2c驱动支持polling模式安装和interrupt模式安装,安装指令:

interrupt mode: insmod ./inspur\_i2c\_ocores\_tiger.ko enable\_features=0x02

polling mode: insmod ./inspur\_i2c\_ocores\_tiger.ko enable\_features=0x01

注: polling模式工作时，CPU 需要高频率实时查询状态寄存器判断硬件是否完成了信号发送以及是否接收到Slave 设备发来的信号，这种模式CPU使用率比较高; 中断模式是硬件完成对应的操作或者接收到Slave发送的信号后发送中断信号给CPU , CPU响应中断handler进入中断处理中断请求,完成I2C 通信.

# Flow

<Describe code, data and control flow>

本节主要说明代码原理和控制流，驱动实现I2C-Ocores通信有两种方式：轮询模式和中断模式，将分别说明这两种方式的实现原理。在说明实现原理之前会先介绍实现这两种方式所使用的原理：I2C-Ocores IP接口操作说明和I2C通信协议。

1. **I2C-Ocores IP接口操作说明**

I2C 通讯核心模块是I2C-Ocores IP，提供的操作接口为I2C bit command，分别为I2C Start、I2C Rep Start、I2C Stop、I2C Write、I2C read.

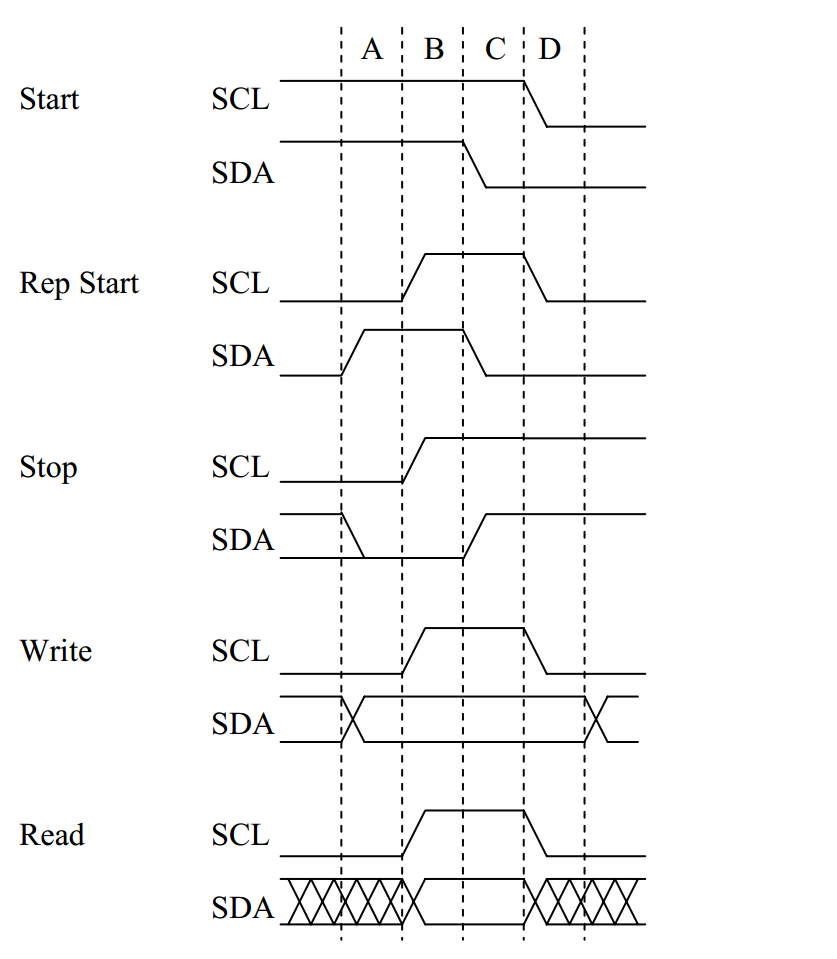


图4-1 I2C-Ocores IP的操作接口操作说明

由Driver将上述Bit command 命令组合使用，实现标准的I2C Read、I2C Write、I2C Combined format的完整通讯。I2C-Ocores IP 会接受单个Bit Command ，当其完成时会产生中断。因此，流程章节，我们重点介绍处理中断的驱动流程，以及组合Bit command 发送完成I2C 数据报文的流程，其中中断处理设计中使用了状态机的模型，针对此状态机做重点说明。

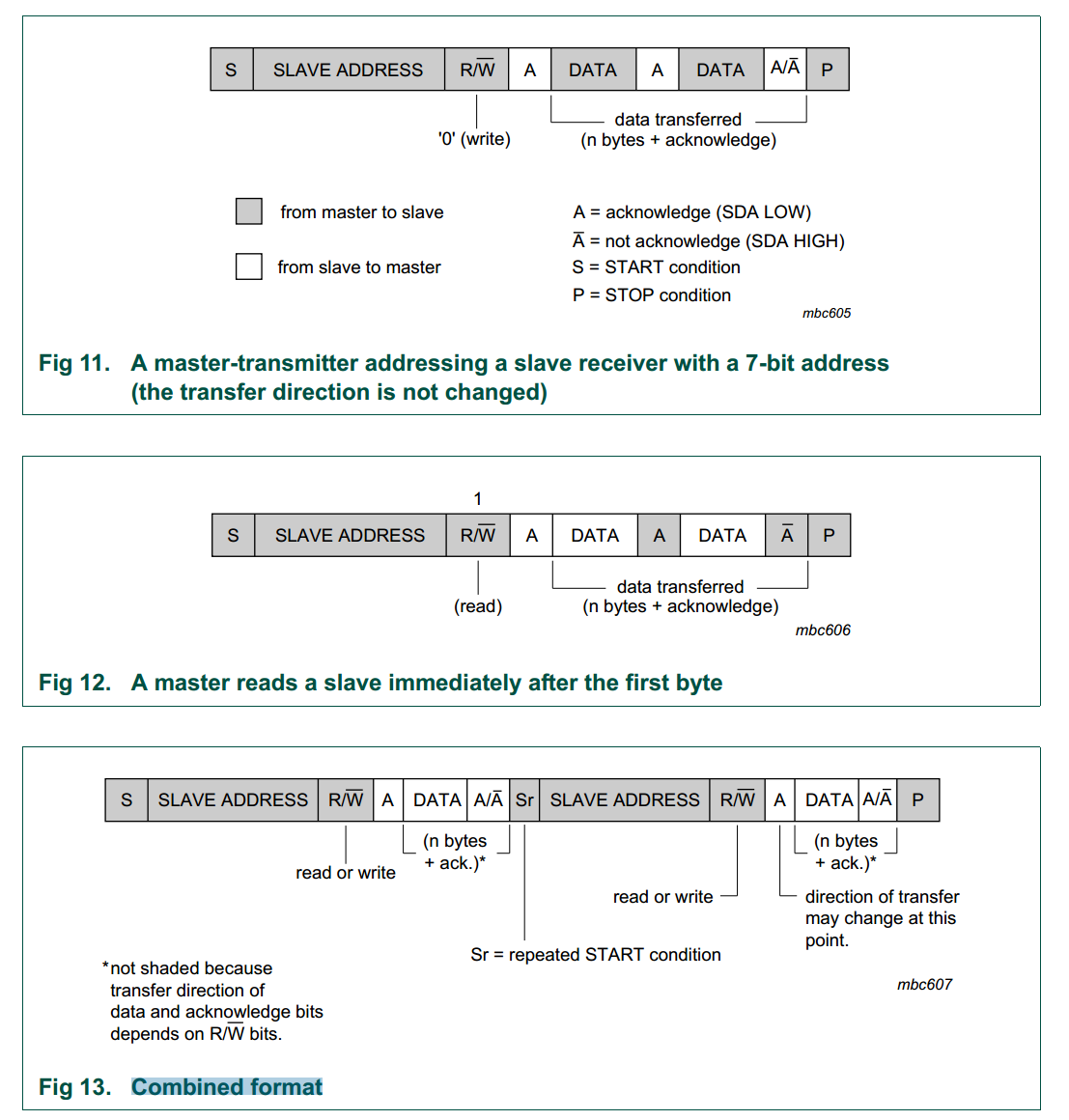


图4-2 I2C数据报文的流程

1. **I2C 通讯协议说明：**

Normally, a standard communication consists of four parts:

1) START signal generation

2) Slave address transfer

3) Data transfer

4) STOP signal generation

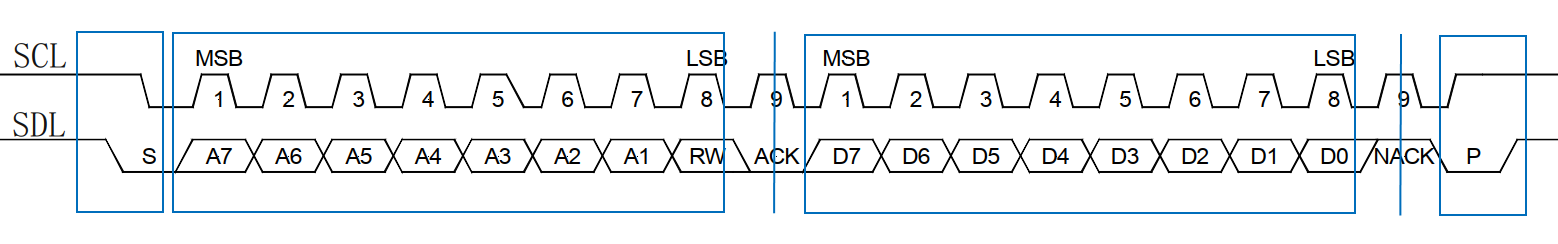


图4-3 I2C通信协议流程

**START signal**

When the bus is free/idle, meaning no master device is engaging the bus (both SCL and SDA lines are high), a master can initiate a transfer by sending a START signal. A START signal, usually referred to as the S-bit, is defined as a high-to-low transition of SDA while SCL is high. The START signal denotes the beginning of a new data transfer. A Repeated START is a START signal without first generating a STOP signal. The master uses this method to communicate with another slave or the same slave in a different transfer direction (e.g. from writing to a device to reading from a device) without releasing the bus.

(The core generates a START signal when the STA-bit in the Command Register is set and the RD or WR bits are set. Depending on the current status of the SCL line, a START or Repeated START is generated.)

**Slave Address Transfer**

The first byte of data transferred by the master immediately after the START signal is the slave address. This is a seven-bits calling address followed by a RW bit. The RW bit signals the slave the data transfer direction. No two slaves in the system can have the same address. Only the slave with an address that matches the one transmitted by the master will respond by returning an acknowledge bit by pulling the SDA low at the 9th SCL clock cycle.

The master writes date to the slave, the seven-bits address followed by a low level.

**Data Transfer**

Once successful slave addressing has been achieved, the data transfer can proceed on a byte-by-byte basis in the direction specified by the RW bit sent by the master. Each transferred byte is followed by an acknowledge bit on the 9th SCL clock cycle. If the slave signals a No Acknowledge, the master can generate a STOP signal to abort the data transfer or generate a Repeated START signal and start a new transfer cycle.

If the master, as the receiving device, does not acknowledge the slave, the slave releases the SDA line for the master to generate a STOP or Repeated START signal.

To write data to a slave, store the data to be transmitted in the Transmit Register and set the WR bit. To read data from a slave, set the RD bit. During a transfer the core set the TIP flag, indicating that a Transfer is In Progress. When the transfer is done the TIP flag is reset, the IF flag set and, when enabled, an interrupt generated. The Receive Register contains valid data after the IF flag has been set. The user may issue a new write or read command when the TIP flag is reset.

**STOP signal**

The master can terminate the communication by generating a STOP signal. A STOP signal, usually referred to as the P-bit, is defined as a low-to-high transition of SDA while SCL is at logical ‘1’.

1. **Polling模式原理**

polling模式工作原理流程图如下：



4-4 polling模式软件流程图

polling模式原理如上图, 驱动设置完I2C-Ocores命令, 每隔1us查询一次SR寄存器, 直至完成通信, 这种方法的缺点是CPU占用率高;

1. **interrupt模式实现原理**

本部分所涉及的名词解释如下：

1. GPE\_EN: 0x42a(GPE13)寄存器bit3标识PORT\_CPLD1是否可以给CPU上送中断; 0x42a(GPE13)寄存器bit4标识PORT\_CPLD2是否可以给CPU上送中断;
2. GPE\_STA: 0x422(GPE13)寄存器bit3标识产生中断是否为PORT\_CPLD1; 0x422(GPE13)寄存器bit3标识产生中断是否为PORT\_CPLD2;
3. GPIO: CPU pin脚, I2C-OCores通过控制这个pin脚的高低电平, 通知CPU PORT\_CPLD是否有中断产生;
4. AdapterX: LPC转I2C驱动实现的adapter, X表示8个Adapter中任意一个;

4.1 interrupt模式软件原理



4-5 interrupt模式软件原理图

应用层访问eload/AOC eeprom时, 先通过LPC总线将需要读写操作的从设备地址和读写指令写入I2C-Ocores寄存器(PORT\_CPLD), LPC转I2C驱动严格按照I2C通信时序控制I2C-Ocores寄存器实现I2C通信.

具体地:(假设PORT\_CPLD1 flash I2C-Ocores有中断发生)

1. 若I2C-Ocores与从设备通信完成, PORT\_CPLD1通过拉低CPU GPIO通知CPU I2C-Ocores已完成和从设备通信, LPC转I2C驱动进入中断处理函数.
2. 中断处理函数首先判断中断源是否PORT\_CPLD1上送, 如果不是则退出中断函数, 反之则继续执行中断处理函数: 关闭CPU中断GPE\_EN, 再写PORT\_CPLD1寄存器拉高GPIO停止PORT\_CPLD1向CPU上报中断, 再清除GPE\_STA寄存器后进行中断处理.
3. 因为每个PORT\_CPLD利用OCore IP实现了4个I2C-Ocores, 因此需要查询PORT\_CPLD1寄存器低四位, 如果有中断产生驱动逐一处理.
4. 中断处理完成后控制GPE\_EN寄存器开启中断使能, 再写PORT\_CPLD1寄存器允许I2C-Ocores上报中断, 准备执行下次中断.
   1. 中断处理流程图



4-6 interrupt模式软件流程图

* 1. 状态机的变迁图



4-7 interrupt模式状态机图

|  |
| --- |
| 特别说明:状态机中除STATE\_DONE/STATE\_STOP状态,都有概率发生错误，进入到STATE\_ERROR状态。I2C 传输的开始和结束是应用软件和驱动控制，中间通讯过程中所有写I2C-Ocores寄存器的行为，都会触发后续的中断。  关于I2C-Ocores的操作说明，请参考：I2C-openCore文档 https://opencores.org/projects/i2c/ |
| ①I2C transfer start / 设置从设备地址和设置I2C-Ocores控制寄存器,设置状态机状态为STATE\_START; |
| ②I2C写请求 / 设置I2C-Ocores控制寄存器写bit4置1,状态机设置STATE\_WRITE; |
| ③I2C读请求 / 设置I2C-Ocores控制寄存器写bit5置1,状态机设置STATE\_READ; |
| ④⑤一个I2C MSG内容传输完成,转换读写模式(Combined transactions),状态机状态由STATE\_WRITE转变为STATE\_READ或STATE\_READ转变为STATE\_WRITE; |
| ⑥⑦一个I2C Transfer传输完成 / 设置I2C-Ocores 控制寄存器bit6置1(generate stop condition),设置状态机STATE\_STOP; |
| ⑧传输完成 / 设置I2C-Ocores控制寄存器bit0置1(clears a pending interrupt),唤醒i2c\_access()函数中事件等待函数,状态机状态设置STATE\_DONE; |
| ⑨在STATE\_DONE状态中断handle被调用 / 中断handle收到一个异常中断,退出中断,设置I2C-Ocores控制寄存器bit0置1(clears a pending interrupt); |

# Develop of Debug

程序使用内核API dev\_err、dev\_dbg、dev\_info打印日志

例如：dev\_info(&pdev->dev, "print log"); // 其他API遵从这个格式

动态调试方法：

1. 打开内核动态调试开关，make menuconfig选中

CONFIG\_DYNAMIC\_DEBUG以及CONFIG\_DEBUG\_FS

1. Linux启动后，使用命令行挂载上dbgfs

mkdir /mnt/dbg

mount -t debugfs none /mnt/dbg

1. 使用下面方式控制你想输出dev\_dbg()信息

控制某个文件所有dev\_dbg()，echo -n “file xxx.c +p” >

/mnt/dbg/dynamic\_debug/control

控制某个函数所有dev\_dbg()，echo -n “func xxx +p” >

/mnt/dbg/dynamic\_debug/control

运行程序，使用dmesg则可以看到相应dev\_dbg()的输出信息

当调试结束，不再想输出dev\_dbg()信息了，使用下面命令关闭即可

echo -n “file xxx.c -p” > /mnt/dbg/dynamic\_debug/control

echo -n “func xxx -p” > /mnt/dbg/dynamic\_debug/control

# Database

<Describe the configuration if some functions have. If none, then N/A.>

本文档不涉及

# CLI

<SONiC-CLI and inCLI format description>

本文档不涉及

## SONiC-CLI

本文档不涉及

## inCLI

本文档不涉及

# Security

本文档不涉及

## Securing data

<If there are encrypted storage list and encrypted transmission list, please specify in this section.>

Example:

[Encrypted](javascript:;) [Storage](javascript:;) List: N/A

[Encrypted](javascript:;) Transfer List: N/A

本文档不涉及

## Security protocol

<If security protocol is used on the module, describe it in this section. >

Example:

Protocals: sftp, scp, etc.

本文档不涉及

## Security Algorithm

<If this module uses specific protocol, encryption, transmission algorithms, please describe them in this section. >

Example:

Protocol Algorithm: N/A

[Encrypted](javascript:;) Algorithm: N/A

[Transfer](javascript:;) Algorithm: N/A

本文档不涉及

## Third Party Open Source Component

<If this module uses third party open source components, please describe them in this section. >

Example:

Component Name: rsyslogd

Component Version: 8.24.0-1+deb9u3

Download: http://security.debian.org/debian-security/pool/updates/main/r/rsyslog/rsyslog\_8.24.0-1+deb9u3\_arm64.deb

CVE Vulnerability: CVE-2018-16881

遵守内核GPL规范: MODULE\_LICENSE("GPL");

# Public APIs

<Public APIs to provide the service. If none, then N/A.>

1. Kernel I2C subsystem 的公共API（kernel space）

请参考/include/i2c.h。

1. Sysfs（userspace）

在/sys/bus/i2c/device/目录下创建i2c-$bus\_num节点，通过执行如下两条命令：

// i2c adapter下游链路直接挂eload

echo "optoe 0x50" > /sys/bus/i2c/devce/i2c-$bus\_num

// i2c adapter下游链路连接pca9548，pca96548虚拟出8路i2c通道，每一路i2c通路挂载一个eload

echo "pca9548 0x70" > /sys/bus/i2c/devce/i2c-$bus\_num

echo "optoe2 0x50" > /sys/bus/i2c/devce/i2c-$bus\_num

先创建pca9548节点在创建对应eload sysfs节点。

1. /dev（userspace）

在/dev/目录下创建/dev/i2c-$bus创建对应的sysfs节点，i2cdump、i2cget等工具会使用这中节点。

# Function Test

< Detailed description of the test scheme>

本节主要说明CPU加压测试条件，I2C Adapter可用性和可靠性测试方案。

1. 回板后增加CPU压力条件下，测试8路I2C Adapter同时工作的可靠性、稳定性；（压力读写测试、多进程读写、所有i2c master 12小时+，不能出现读写异常或者错误数据）
2. 驱动加卸载 1K 次，不能报错，无内存泄露；
3. 多路并行工作，CPU 利用率低于 3%（中断模式下）；
4. CPU 加压和内存加压情况下，读写无异常；所有外设读取加压的情况下，读写无异常。

# Appendix

< Anything not described in the above section>

1. 请参看I2C-openCore文档 https://opencores.org/projects/i2c/
2. 请参看Kernel Documention I2C <https://elixir.bootlin.com/linux/v4.19.152/source/Documentation/i2c>