Mask (鍾文昌)
mask@mask.org.tw
http://www.mask.org.tw
http://www.mask.org.tw/blog

I2C 驅動實作案例討論

深圳 2011.3.25

版權聲明

學習驅動的重點

- 驅動
  - 硬件
  - 軟件
- 精髓
  - 驅動模型 (Driver Model)
- 困難點
  - 軟硬件相關的背景知識

驅動模型的設計概念

- 框架 (Framework)
- 函數指針 (Function pointer)
- 回調函數 (Callback)

擅寫驅動的流程

- 看規格書 (Datasheet) 及原理圖 (Schematic)
- 了解驅動模型
- 擅寫驅動
- 編譯測試
  - 動態載入卸載

實際案例討論

- TI TCA6408A
- 準備資料
- TCA6408A 規格書 (datasheet)
- TCA6408A 相關的原理圖 (schematic)
- ...

規格書

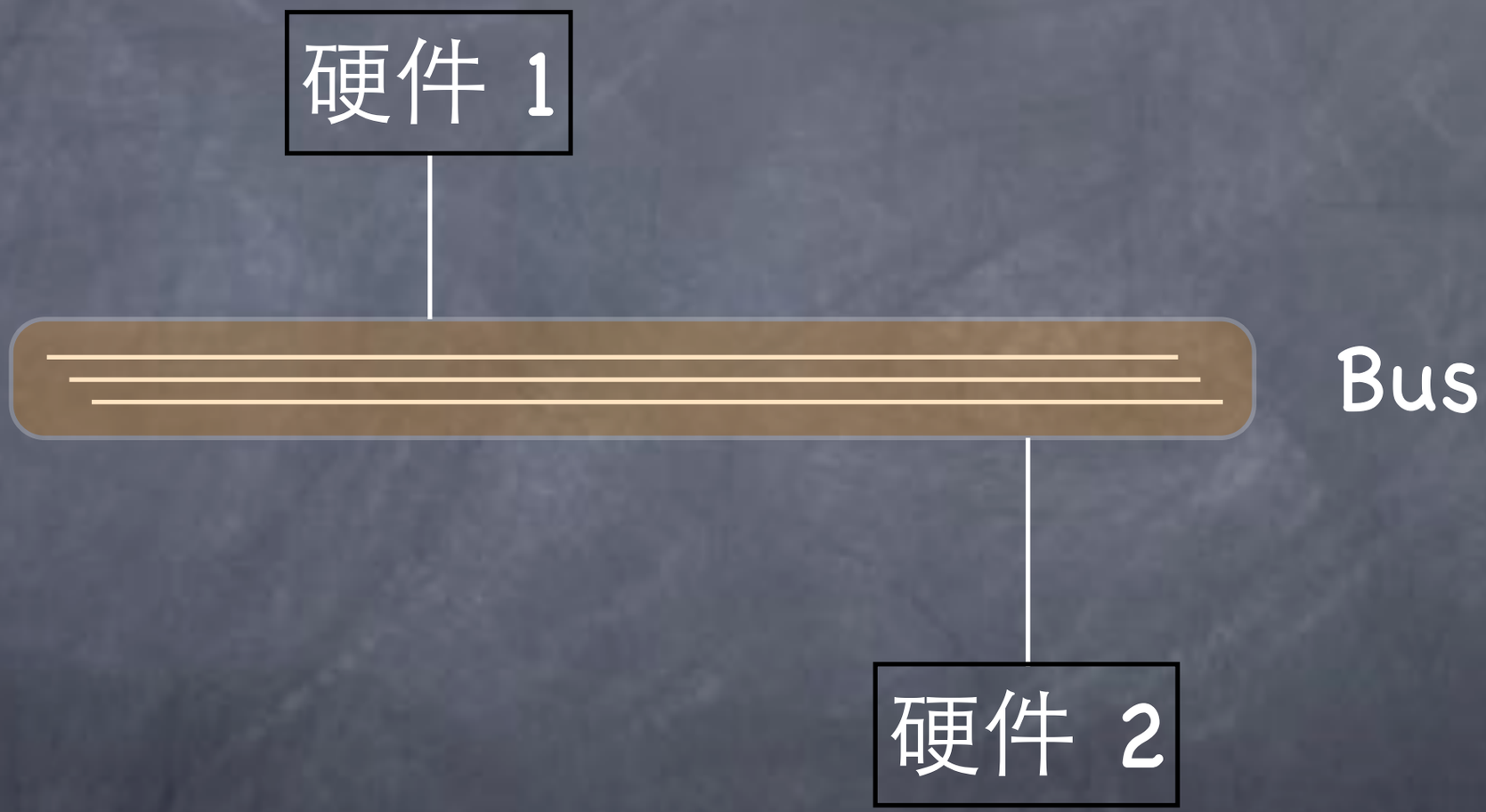
- http://focus.ti.com/lit/ds/symlink/tca6408.pdf

原理圖

原理圖 (接上頁)

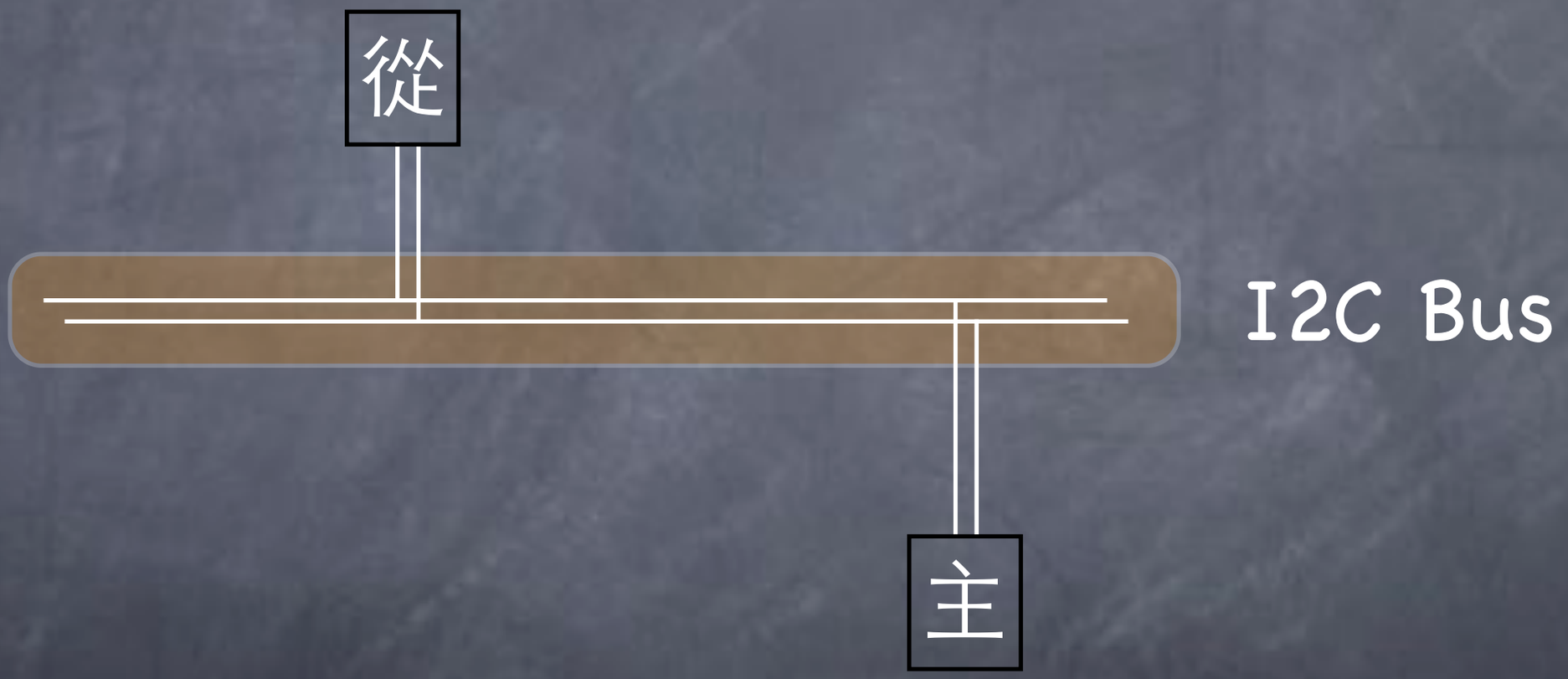
- 電壓
- 電阻, 電容
- 輸入, 輸出的方向

硬件連接圖



I2C 硬件連接圖

I2C 硬件連接圖 (接上頁)



什麼是 I2C

- 目的、用途
- I^2C
- Inter-IC
- http://ics.nxp.com/support/documents/interface/pdf/i2c.bus.specification.pdf
- http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf

I2C 特性

- 簡單
  - 2 條線
    - SCL (clock)
    - SDA (data)
- 省電
- 限制

I2C 通訊協議 (Protocol)

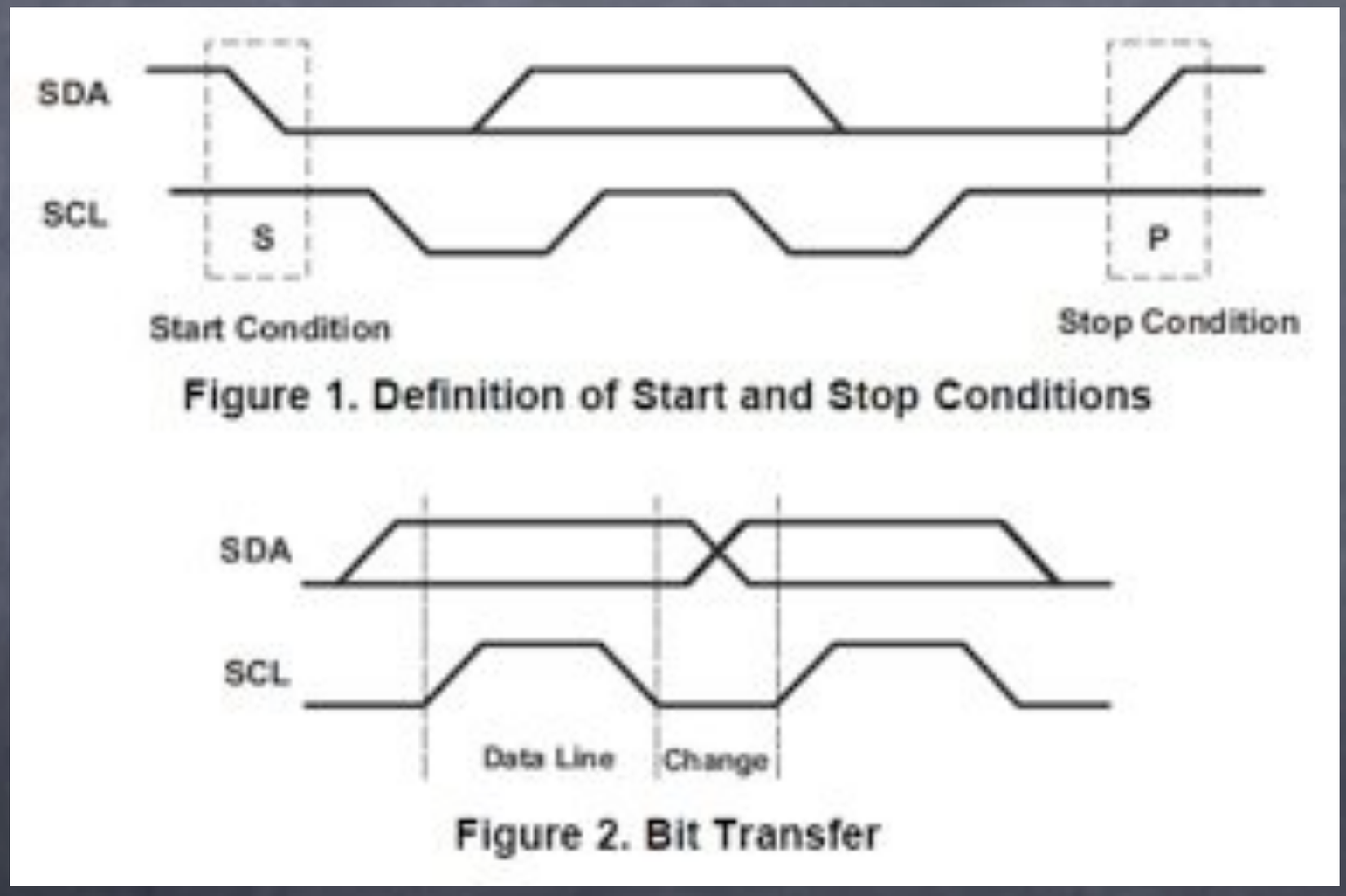


Figure 1. Definition of Start and Stop Conditions

Figure 2. Bit Transfer

I2C 位址

- 7-bit
- 10-bit

I2C 保留位址

Table 3. Reserved addresses

Slave address	R/W bit	Description
0000 000	0	general call address[1]
0000 000	1	START byte[2]
0000 001	X	CBUS address[3]
0000 010	X	reserved for different bus format[4]
0000 011	X	reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	reserved for future purposes
1111 0XX	X	10-bit slave addressing

I2C 保留位址 (接上頁)

- drivers/i2c/i2c-core.c
  - 舊版
    - (addr < 0x03 || addr > 0x77)
  - 新版
    - /*
    - * Reserved addresses per I2C specification:
    - * 0x00 General call address / START byte
    - * 0x01 CBUS address
    - * 0x02 Reserved for different bus format
    - * 0x03 Reserved for future purposes
    - * 0x04-0x07 Hs-mode master code
    - * 0x78-0x7b 10-bit slave addressing
    - * 0x7c-0x7f Reserved for future purposes
    - */

TCA6408A 位址

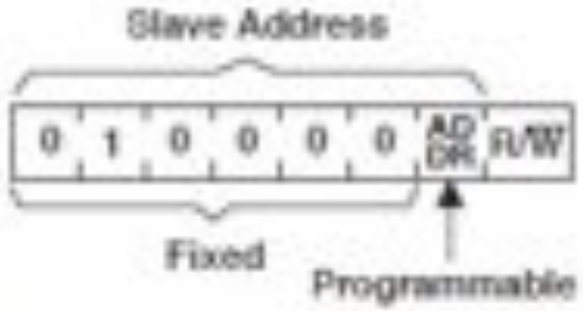
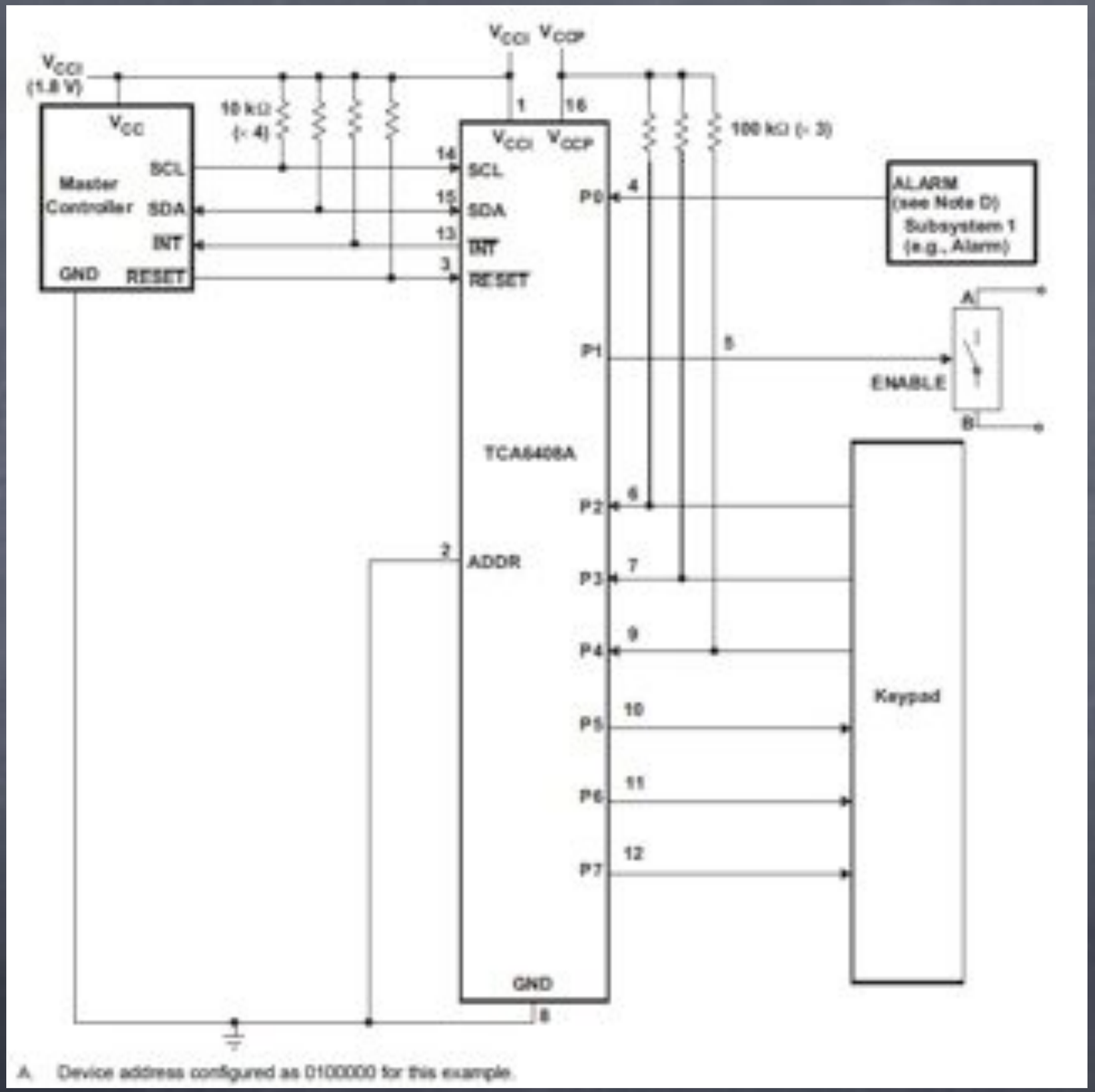


Figure 4. TCA6408A Address

Address Reference

ADDR	I²C BUS SLAVE ADDRESS
L	32 (decimal), 20 (hexadecimal)
H	33 (decimal), 21 (hexadecimal)



A. Device address configured as 0100000 for this example.

原理圖

Why TCA6408A

TCA6408A

- 輸入 (input)
- 輸出 (output)
- 中斷 (interrupt)

設置 TCA6408A

- Input / Output
- High / Low

設置 TCA6408A (接上頁)

Command Byte

CONTROL REGISTER BITS								COMMAND BYTE (HEX)	REGISTER	PROTOCOL	POWER-UP DEFAULT
B7	B6	B5	B4	B3	B2	B1	B0				
0	0	0	0	0	0	0	0	00	Input Port	Read byte	xxxx xxxx
0	0	0	0	0	0	0	1	01	Output Port	Read/write byte	1111 1111
0	0	0	0	0	0	1	0	02	Polarity Inversion	Read/write byte	0000 0000
0	0	0	0	0	0	1	1	03	Configuration	Read/write byte	1111 1111

設置 TCA6408A (接上頁)

Register 3 (Configuration Register)

BIT	C-7	C-6	C-5	C-4	C-3	C-2	C-1	C-0
DEFAULT	1	1	1	1	1	1	1	1

- 1 : input
- 0 : output

I2C 資料傳輸步驟

- Start
- Slave address with R(1)/W(0)
- ACK
- Send/Receive 8 bits of data
- ACK
- Stop

設置 TCA6408A (接上頁)

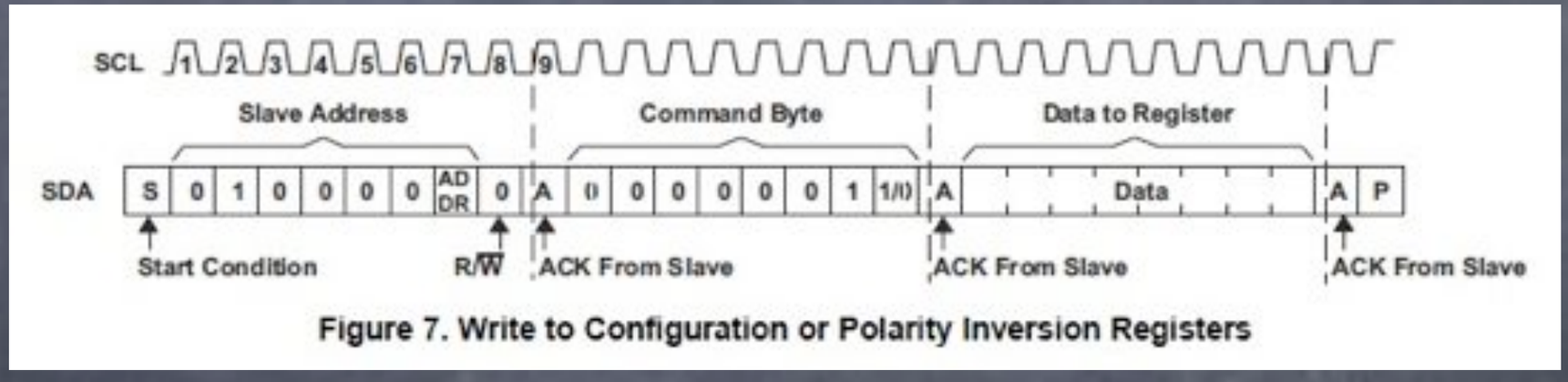


Figure 7. Write to Configuration or Polarity Inversion Registers

設置 TCA6408A (接上頁)

- u8 conf = 0x??;
- tca6408_i2c_write(&client,0x3,conf);
- int tca6408_i2c_write(struct i2c_client *client, u8 cmd, u8 data)
  - u8 buf[2] = {cmd, data};
  - struct i2c_msg msg;
  - msg.addr = client->addr;
  - msg.flags = 0;
  - msg.len = 2;
  - msg.buf = buf;
  - i2c_transfer(client->adapter, &msg, 1);

設置輸出接腳

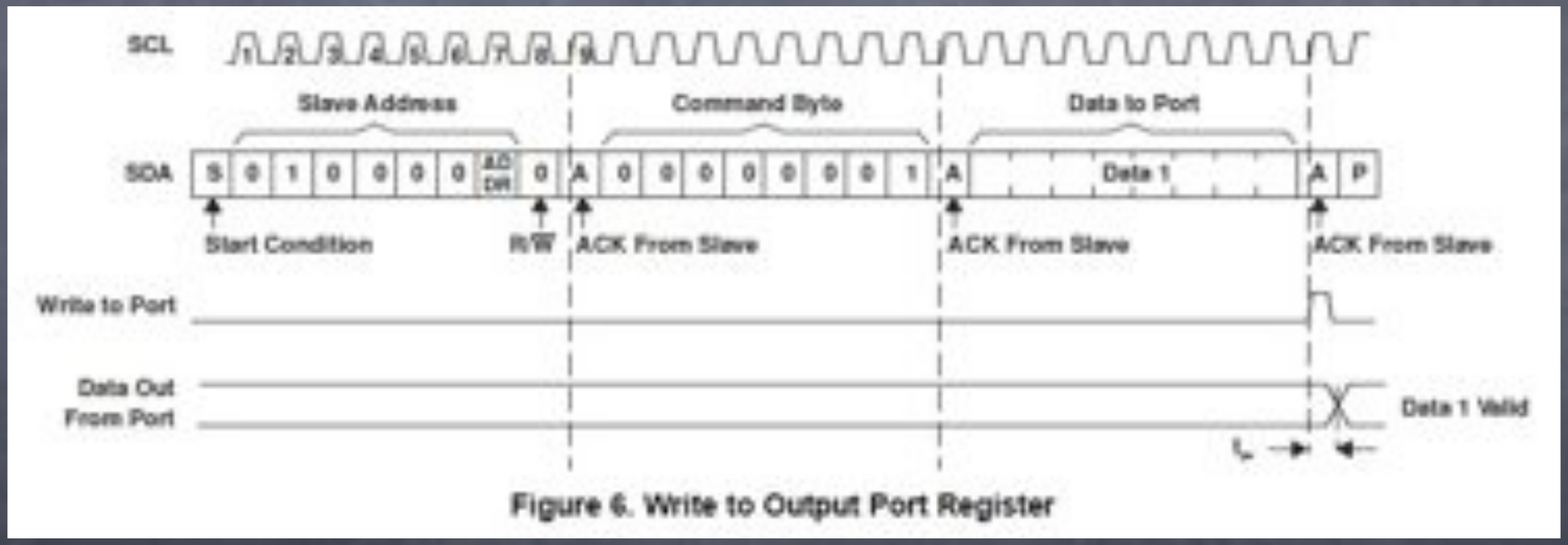


Figure 6. Write to Output Port Register

設置輸出接腳 (接上頁)

- u8 out = 0x??;
- tca6408_i2c_write(&client,0x1,out);

讀取輸入資料

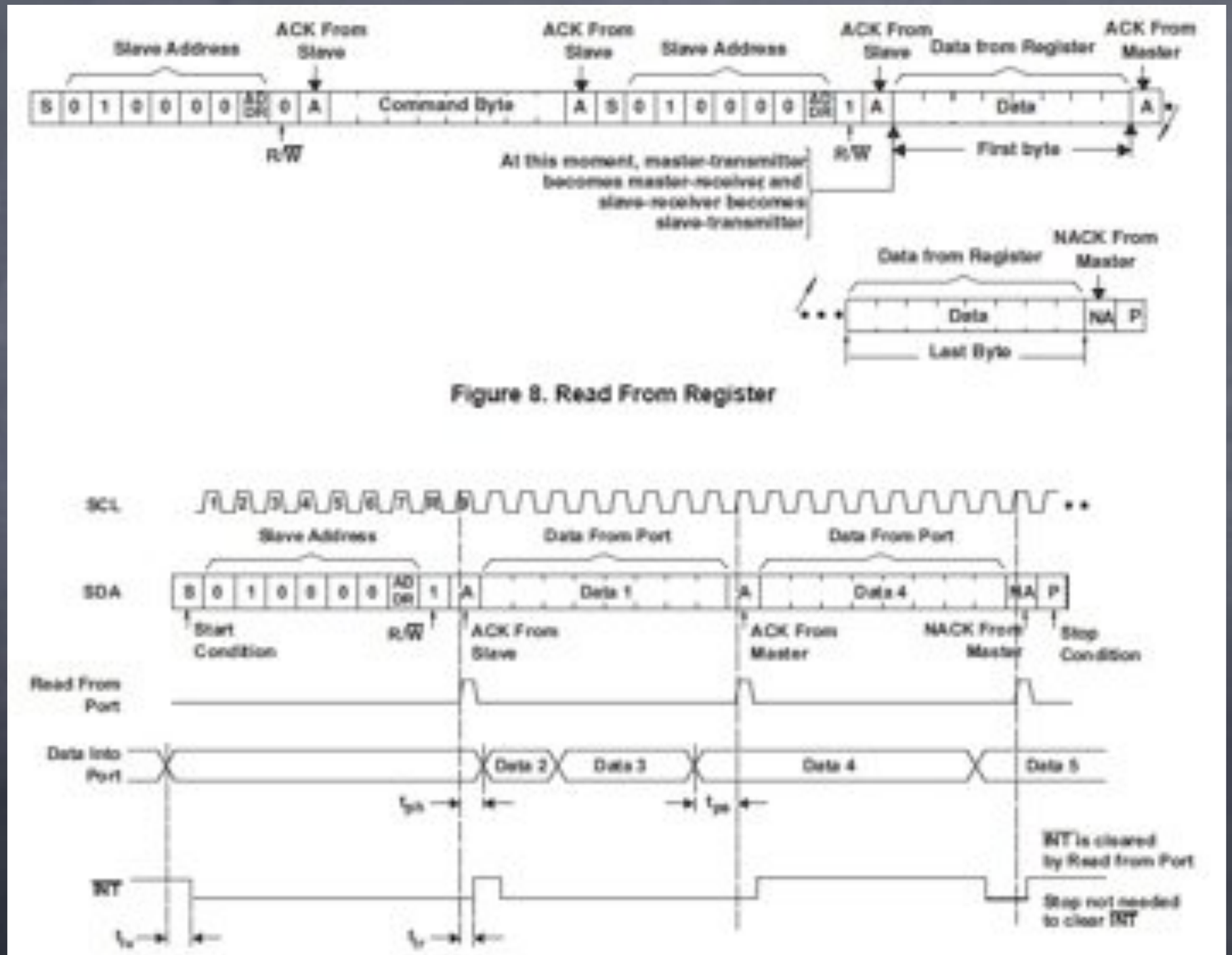


Figure 8. Read From Register

讀取輸入資料 (接上頁)

- u8 in;
- tca6408_i2c_read(&client,0x2,&in,1);
- int tca6408_i2c_read(struct i2c_client *client, u8 cmd, u8 *data, u8 len)
  - struct i2c_msg msg;
  - msg.addr = client->addr;
  - msg.flags = 1;
  - msg.len = len;
  - msg.buf = data;
  - *data = cmd;
  - i2c_transfer(client->adapter, &msg, 1);

讀取輸入資料 (接上頁)

- u8 in;
- tca6408_i2c_read(&client,0x2,&in,1);
- int tca6408_i2c_read(struct i2c_client *client, u8 cmd, u8 *data, u8 len)
  - struct i2c_msg msg[2];
  -
  - msg[0].addr = client->addr;
  - msg[0].flags = 0;
  - msg[0].len = 1;
  - msg[0].buf = &cmd;
  -
  - msg[1].addr = client->addr;
  - msg[1].flags = 1;
  - msg[1].len = len;
  - msg[1].buf = data;
  -
  - i2c_transfer(client->adapter, msg, 2);

總結

- u8 conf = 0x??;
- u8 out = 0x??;
- u8 in;
- tca6408_i2c_write(&client,0x3,conf);
- tca6408_i2c_write(&client,0x1,out);
- tca6408_i2c_read(&client,0x2,&in,1);

總結 (接上頁)

- 中斷
- 延遲

新舊版差異

- Documentation/i2c/upgrading-clients

舊版

- struct example_state {
- 	struct i2c_client	client;
- 	....
- };
- static struct i2c_driver example_driver;
- static unsigned short ignore[] = { I2C_CLIENT_END };
- static unsigned short normal_addr[] = { OUR_ADDR, I2C_CLIENT_END };
- I2C_CLIENT_INSMOD;

舊版 (接上頁)

```
static struct i2c_driver example_driver = {
        .driver         = {
                .owner          = THIS_MODULE,
                .name           = "example",
        },
        .attach_adapter = example_attach_adapter,
        .detach_client  = __devexit_p(example_detach),
        .suspend        = example_suspend,
        .resume         = example_resume,
};
```

舊版 (接上頁)

```
static int __init example_init(void)
{
        return i2c_add_driver(&example_driver);
}

static void __exit example_exit(void)
{
        i2c_del_driver(&example_driver);
}
module_init(example_init);
module_exit(example_exit);
```

舊版 (接上頁)

```
static int example_attach_adapter(struct i2c_adapter *adap)
{
        return i2c_probe(adap, &addr_data, example_actually_execute);
}

static int __devexit example_detach(struct i2c_client *client)
{
        struct example_state *state = i2c_get_clientdata(client);


        i2c_detach_client(client);
        kfree(state);
        return 0;
}
```

舊版 (接上頁)

- static int example_actually_execute(struct i2c_adapter *adap, int addr, int kind)
- {
- ...
- };

新版

- static struct i2c_driver example_driver = {
- +	.probe		= example_probe,
- +	.remove		= __devexit_p (example_remove),
- }

新版 (接上頁)

- - static int example_actually_execute(struct i2c_adapter *adap, int addr, int kind)
- + static int example_probe(struct i2c_client *client, const struct i2c_device_id *id)

新版 (接上頁)

```
-       example->client.addr    = addr;
-       example->client.flags   = 0;
-       example->client.adapter = adap;
-
-       strlcpy(client->i2c_client.name, "example", I2C_NAME_SIZE);
-       ret = i2c_attach_client(&state->i2c_client);
-       if (ret < 0) {
-               dev_err(dev, "failed to attach client\n");
-               kfree(state);
-               return ret;
-       }
```

新版 (接上頁)

- struct example_state {
- -	struct i2c_client	client;
- +	struct i2c_client	*client;
- ...
- };

新版 (接上頁)

- - static int __devexit example_detach(struct i2c_client *client)
- + static int __devexit example_remove(struct i2c_client *client)
- {
- 	struct example_state *state = i2c_get_clientdata(client);
-
- -	i2c_detach_client(client);
-

新版 (接上頁)

```
+ struct i2c_device_id example_idtable[] = {
+       { "example", 0 },
+       { }
+};
+
+MODULE_DEVICE_TABLE(i2c, example_idtable);
static struct i2c_driver example_driver = {
        .driver         = {
                .owner          = THIS_MODULE,
                .name           = "example",
        },
+       .id_table       = example_idtable,
```

代碼分析 tca6408.c

- 代碼風格 (coding style)
  - #include
  - #define
    - 函數名稱 (function name)
    - 變數名稱 (variable name)
    - ...
- struct
- delay
- 陷阱