tps61165背光ic的驱动流程

本项目使用的背光ic型号为tps6115,采用的控制方式为一线协议调节方式:

其驱动如下:

//add for tps6115 ++

static int tps61165\_write\_bit(u8 b);

static int tps61165\_write\_byte(u8 bytedata);

static int tps61165\_config\_ES\_timing(void);

static int tps61165\_shutdown(void);

static int tps61165\_init(void);

int tps61165\_backlight(unsigned long brightness);

void tps61165\_set\_backlight(unsigned long brightness);

static spinlock\_t s\_tps61165\_lock;

static bool s\_tps61165\_is\_inited = false;

static bool spinlock\_init = false;

static bool s\_bNeedSetBacklight = false;

#define red\_mode (0)

#define write\_mode (1)

static u8 current\_mode = write\_mode;

static u8 current\_brightness = 180;

#define DEFAULT\_BRIGHTNESS (132)

#define TPS61165\_CTRL\_PIN LCD\_BACKLIGHT\_WRITE\_PIN

#define TPS61165\_CTRL\_PIN\_RED LCD\_BACKLIGHT\_RED\_PIN

#define TPS61165\_DEVICE\_ADDR (0x72)

#define BIT\_DELAY\_UNIT (3)

#define LOGIC\_FACTOR (3)

#define CONDITION\_DELAY (3)

#define ES\_DETECT\_DELAY (200)

#define ES\_DETECT\_TIME (300)

#define ES\_TIMING\_WINDOW (1000)

#define TPS61165\_DELAY(n) udelay(n)

int tps61165\_setMode(void){

pr\_err("stephen tps61165\_setMode\n");

tps61165\_shutdown();

current\_mode = !current\_mode;

s\_tps61165\_is\_inited = false;

tps61165\_set\_backlight(current\_brightness);

//tps61165\_set\_backlight(150);

return 0;

}

int tps61165\_getMode(void){

return current\_mode;

}

static int tps61165\_write\_bit(u8 b)

{

if(1 == b)

{

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

TPS61165\_DELAY(BIT\_DELAY\_UNIT);

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1);

TPS61165\_DELAY(LOGIC\_FACTOR\*BIT\_DELAY\_UNIT);

//gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

}

else if(0 == b)

{

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

TPS61165\_DELAY(LOGIC\_FACTOR\*BIT\_DELAY\_UNIT);

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1);

TPS61165\_DELAY(BIT\_DELAY\_UNIT);

//gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

}

else

{

printk("tps61165\_write\_bit: error param!\n");

return -1;

}

return 0;

}

static int tps61165\_write\_byte(u8 bytedata)

{

u8 bit\_cnt = 8;

u8 val = bytedata;

int ret = 0;

unsigned long flags;

spin\_lock\_irqsave(&s\_tps61165\_lock, flags);

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1);

TPS61165\_DELAY(CONDITION\_DELAY); //Start condition, at least 2us

bit\_cnt = 8;

while(bit\_cnt)

{

bit\_cnt--;

if((val >> bit\_cnt) & 1)

{

ret = tps61165\_write\_bit(1);

}

else

{

ret = tps61165\_write\_bit(0);

}

if(ret)

{

printk("[tong]:tps61165\_write\_byte:failed!\n");

spin\_unlock\_irqrestore(&s\_tps61165\_lock, flags);

return ret;

}

}

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

TPS61165\_DELAY(CONDITION\_DELAY); //EOS condition, at least 2us

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1);

spin\_unlock\_irqrestore(&s\_tps61165\_lock, flags);

return 0;

}

static int tps61165\_config\_ES\_timing(void)

{

unsigned long flags;

if(debug\_tps61165)

{

printk("tps61165\_config\_ES\_timing\n");

}

spin\_lock\_irqsave(&s\_tps61165\_lock, flags);

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1); //start ES Timing Window

TPS61165\_DELAY(ES\_DETECT\_DELAY); //at least 100us

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

TPS61165\_DELAY(ES\_DETECT\_TIME); //at least 260us

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1);

TPS61165\_DELAY(ES\_TIMING\_WINDOW - ES\_DETECT\_DELAY - ES\_DETECT\_TIME);

spin\_unlock\_irqrestore(&s\_tps61165\_lock, flags);

return 0;

}

static int tps61165\_shutdown(void)

{

if(debug\_tps61165)

{

printk("tps61165\_shutdown\n");

}

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 0);

mdelay(3); //enter shutdown mode, at least 2.5ms

return 0;

}

static int tps61165\_init(void)

{

tps61165\_shutdown();

tps61165\_config\_ES\_timing();

return 0;

}

int tps61165\_backlight(unsigned long brightness)

{

u8 tps61165\_level;

static u8 old\_level = -1;

/\*

if(DEFAULT\_BRIGHTNESS == brightness)

{

if(!s\_bNeedSetBacklight)

{

s\_bNeedSetBacklight = true;

return 0;

}

}

\*/

tps61165\_level = (brightness & 0xFF) >> 3;/\*convert level 0~255 to 0~31\*/

if(old\_level == tps61165\_level)

{

//printk("tps61165\_set\_backlight: the same level as before, nothing done!level=%d\n", tps61165\_level);

//return 0;

}

if(debug\_tps61165)

{

printk("tps61165\_set\_backlight: brightness=%d, tps61165\_level=%d\n", brightness, tps61165\_level);

}

if(tps61165\_level)

{

if(!s\_tps61165\_is\_inited)

{

tps61165\_init();

s\_tps61165\_is\_inited = true;

}

}

else

{

tps61165\_shutdown();

old\_level = tps61165\_level;

s\_tps61165\_is\_inited = false;

return 0;

}

tps61165\_write\_byte(TPS61165\_DEVICE\_ADDR);

tps61165\_write\_byte(tps61165\_level);

old\_level = tps61165\_level;

current\_brightness = old\_level << 3;

return 0;

}

//add for tps6115 --

void tps61165\_set\_backlight(unsigned long brightness)

{

//for one wire fix

if(debug\_tps61165)

pr\_err("stephen tps61165\_set\_backlight current\_mode:%d,current\_brightness:%d\n",current\_mode,current\_brightness);

if(!spinlock\_init)

{

pr\_err("stephen tps61165 init TPS61165\_CTRL\_PIN:%d,%d\n",TPS61165\_CTRL\_PIN,TPS61165\_CTRL\_PIN\_RED);

gpio\_request(TPS61165\_CTRL\_PIN,"write");

gpio\_request(TPS61165\_CTRL\_PIN\_RED,"red");

spin\_lock\_init(&s\_tps61165\_lock);

gpio\_direction\_output(current\_mode ? TPS61165\_CTRL\_PIN:TPS61165\_CTRL\_PIN\_RED, 1);

s\_tps61165\_is\_inited = false;

spinlock\_init = true;

}

tps61165\_backlight(brightness);

}

背光驱动的设置:

static struct comip\_backlight\_platform\_data comip\_backlight\_data = {

#if defined(CONFIG\_LCD\_HS\_NT35517) //[zhangxiang 20151208] for support LCD\_HS\_NT35517

.ctrl\_type = CTRL\_PWM,

.gpio\_en = -1,

.pwm\_en = 1,

.pwm\_id = 0,

.pwm\_clk = 32500, //32.5KhZ

.pwm\_ocpy\_min = 10,

#elif defined(CONFIG\_LCD\_BT\_ILI9806E)

.ctrl\_type = CTRL\_PWM,

.gpio\_en = -1,

.pwm\_en = 1,

.pwm\_id = 0,

.pwm\_clk = 32500, //32.5KhZ

//#if CONFIG\_PWM\_OCPY

// .pwm\_ocpy\_min = 10,

//#else

.pwm\_ocpy\_min = 10/CONFIG\_PWM\_OCPY,

//#endif

#elif defined(CONFIG\_LCD\_RD\_RM69052)

.ctrl\_type = CTRL\_LCDC,

#elif defined(CONFIG\_LCD\_ILI9881C)

.ctrl\_type = CTRL\_EXTERNAL,

.bl\_set\_external = tps61165\_set\_backlight,

#else

.ctrl\_type = CTRL\_LCDC,

#endif

.bl\_control = comip\_lcd\_bl\_control,

.key\_bl\_control = comip\_key\_bl\_control,

};

static struct platform\_device comip\_backlight\_device = {

.name = "comip-backlight",

.id = -1,

.dev = {

.platform\_data = &comip\_backlight\_data,

}

};

背光驱动的切换:

背光驱动的切换通过接收到gpio中断后，在中断响应中处理

int tps61165\_setMode(void){

pr\_err("stephen tps61165\_setMode\n");

tps61165\_shutdown();

current\_mode = !current\_mode;

s\_tps61165\_is\_inited = false;

tps61165\_set\_backlight(current\_brightness);

//tps61165\_set\_backlight(150);

return 0;

}

static struct work\_struct tps6115\_work;

static struct workqueue\_struct \*tps6115\_wq;

中断响应

static void tps6115\_work\_func(struct work\_struct \*work)

{

pr\_err("stephen tps6115\_work\_func\n");

tps61165\_setMode();

}

static irqreturn\_t tps6115\_isr(int irq, void \*dev\_id){

pr\_err("stephen tps6115\_isr\n");

queue\_work(tps6115\_wq, &tps6115\_work);

return IRQ\_HANDLED;

}

error = request\_any\_context\_irq(irq, tps6115\_isr, IRQF\_TRIGGER\_FALLING, "tps6115 mode select", NULL);