全 志 科 技
Allwinner Technology

A20 TSC 使用说明

V1.0

2013-08-06

Revision History

Version	Date	Changes compared to previous issue
V1.0	2013-08-06	初建版本

目录

1.前言

1.1. 编写的目的

了解 TSC 在 A20 上面的开发与使用方法。

1.2. 使用范围

Allwinner A20 平台。

1.3. 相关人员

A20 平台 TSC 设备驱动开发人员以及应用使用相关人员。

2.模块介绍

2.1.功能介绍

TSC-transport stream controller（传输流控制器）。可以接收一个同步串行接口或者同步并行接口的 TS 流软件包。TSC 可以将包含一个或者多个节目，多个原始流的软件包通过他们的 PID，将无效的包滤除并且将有用的原始流存储到存储空间中。即将包含多个节目的 TS 流进行分包处理。

2.2 TSC 框架图

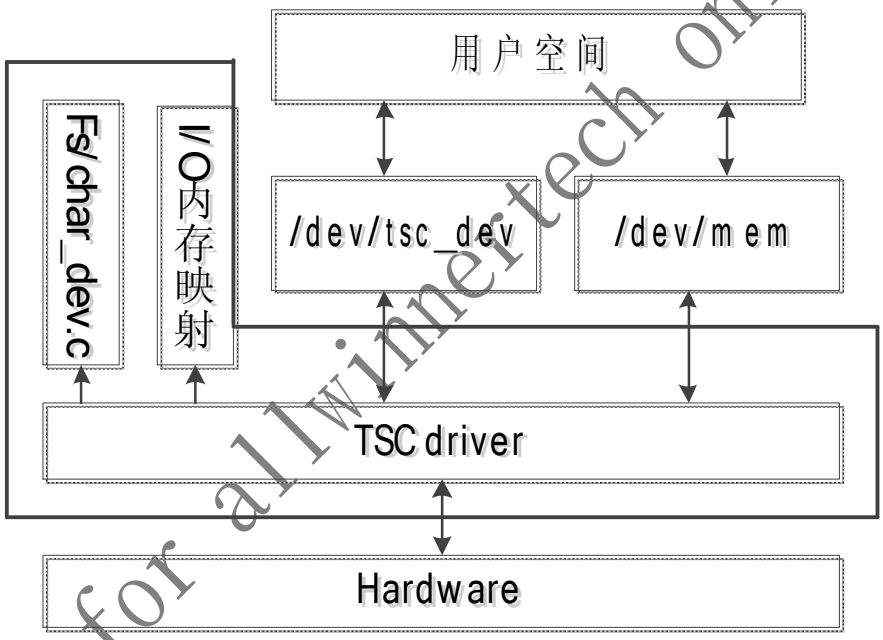


图 1 tsc 框架图

Tsc 中，驱动注册设备节点“/dev/tsc_dev”以及“/dev/men”，用户空间通过控制这两个节点来操作 TSC 进行分包操作。。TS 寄存器的控制主要放在 TS hal 层，即单独包中的 sun7i_maple\frontend\dvb 下。那么对 TS 的操作可以简化为对节点“/dev/tsc_dev”以及“/dev/men”的操作。

2.3.硬件介绍

A20 中支持两路 32 通道的 TSF（TS Filter），TSC 码流发生器与 HOST 的连接有 13 个 pin 脚，分别为 GND，CLk，PSYNC，ERROR, DVALID，DATA0-DATA7。连接如下图所示。

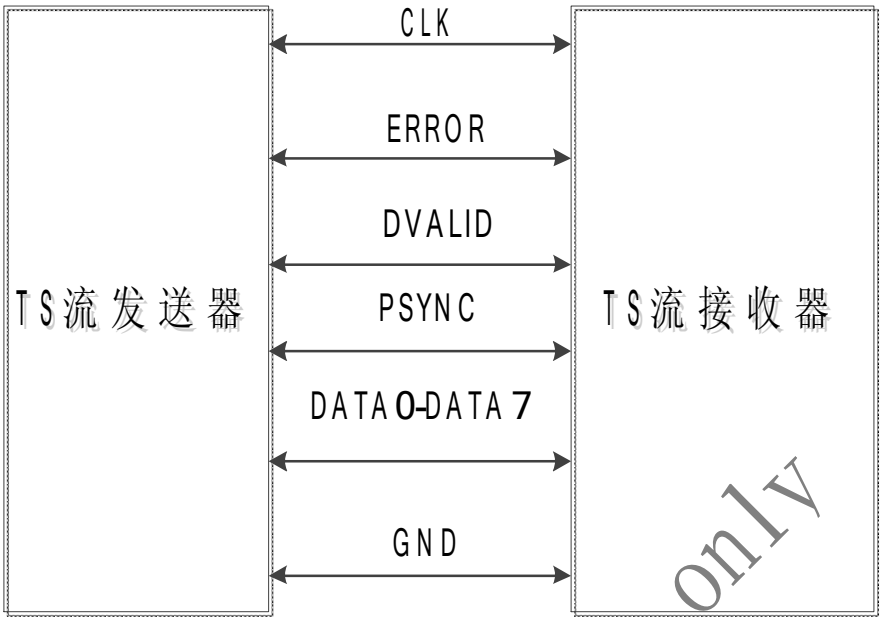


图 1 硬件连接图

TS 的调试中，首先确认硬件的正确性。硬件调试中，需要确认下列项：

（1）确认硬件中使用的为 TS0 接口还是 TS1 接口。

（2）根据硬件要求，确认连接 GPIO 的正确性。

2.4.源码结构介绍

TSC 的 driver 主要负责申请硬件 GPIO 资源，配置模块时钟，申请设备中断，注册字符设备供应用程序使用。TSC 的 linux driver 源码位置：...\lichee\linux-3.x\drivers\media\video\tsc，如下所示：

```
lichee\linux-3.x\driver\media\video\tsc
    |------dvb_drv_sun7i.h
    |------tscdrv.h
    |------tscdrv.c
```

TSC hal 层为提供的单独包，sun7i_maple。主要的作用是用户层设置 TSC 寄存器值，将接收相应的 TS 流软件包进行相对应的分包处理。在 sun7i_maple 单独包中，TSC 的 hal 层即核心代码为 sun7i_maple\frontend\dvb 下的相关代码。TSC 的 hal 用到的头文件存在在 sun7i_maple\include\include_dvb 文件夹中。

TSC 的测试例子为\sun7i_maple\examples\test_dvb\test_tsc 下的相关代码，如下所示：

```
sun7i_maple
    |------frontend
        |------dvb
```

```
                |------tscdev.h
                |------tschal.h
                    |------tscRegs.h
                    |------tscdev.c
                    |------tschal.c
                    |------tscMaintask.c
        |------include
            |------include_dvb
                    |------dvb_drv_sun7i.h
                    |------libtsc.h
        |------examples
            |------test_dvb
                    |------test_tsc
                            |------test_tsc.h
                            |------dvb_drv_sun5i.h
                            |------test_tsc.c
```

2.5.模块配置介绍

2.5.1 sys_config.fex 的配置

(1) TSF0 的配置如下

```
;------------------------------------------------------------------------------
[tsc_para]
tsc_used                    = 1
tsc_clk                     = port:PE00<2><default><default><default>
tsc_err                     = port:PE01<2><default><default><default>
tsc_sync                    = port:PE02<2><default><default><default>
tsc_dvld                    = port:PE03<2><default><default><default>
tsc_d0                      = port:PE04<2><default><default><default>
tsc_d1                      = port:PE05<2><default><default><default>
tsc_d2                      = port:PE06<2><default><default><default>
```

```
tsc_d3                  = port:PE07<2><default><default><default>
tsc_d4                  = port:PE08<2><default><default><default>
tsc_d5                  = port:PE09<2><default><default><default>
tsc_d6                  = port:PE10<2><default><default><default>
tsc_d7                  = port:PE11<2><default><default><default>
```

（2）TSF1 的配置如下

```
;------------------------------------------------------------------------
[tsc_para]
tsc_used                    = 1
tsc_clk                 = port:PG00<2><default><default><default>
tsc_err                 = port:PG01<2><default><default><default>
tsc_sync                = port:PG02<2><default><default><default>
tsc_dvld                    = port:PG03<2><default><default><default>
tsc_d0                  = port:PG04<2><default><default><default>
tsc_d1                  = port:PG05<2><default><default><default>
tsc_d2                  = port:PG06<2><default><default><default>
tsc_d3                  = port:PG07<2><default><default><default>
tsc_d4                  = port:PG08<2><default><default><default>
tsc_d5                  = port:PG09<2><default><default><default>
tsc_d6                  = port:PG10<2><default><default><default>
tsc_d7                  = port:PG11<2><default><default><default>
```

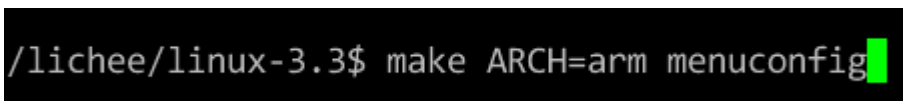
tsc_used 为是否使用 TS 接口，如果设置为 0，则将退出整个 TS 的使用。

2.5.2 menuconfig 的配置

在编译服务器上，目录为\lichee\linux-3.x 上，输入命令：

```
make ARCH=arm menuconfig
```

如下所示：



进入目录 Device Drivers\Multimedia support 目录下可以看到 AW tsc module 模块是编译为模块、编译进内核、不编译。使用 TSC 时，将该模块编译进内核中。

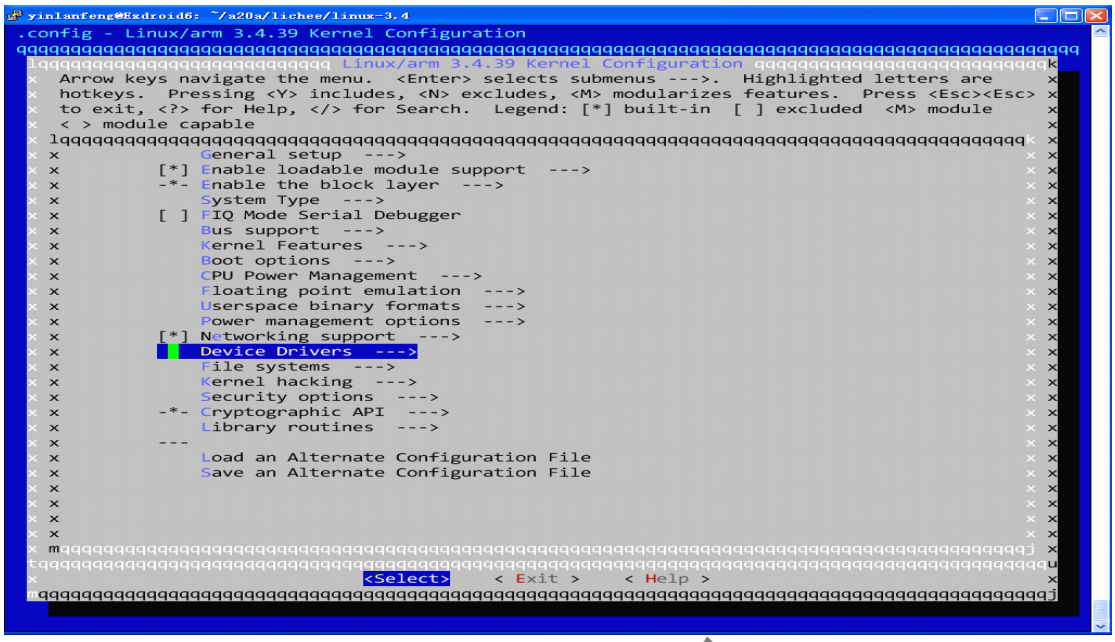


图 2 Device Driver

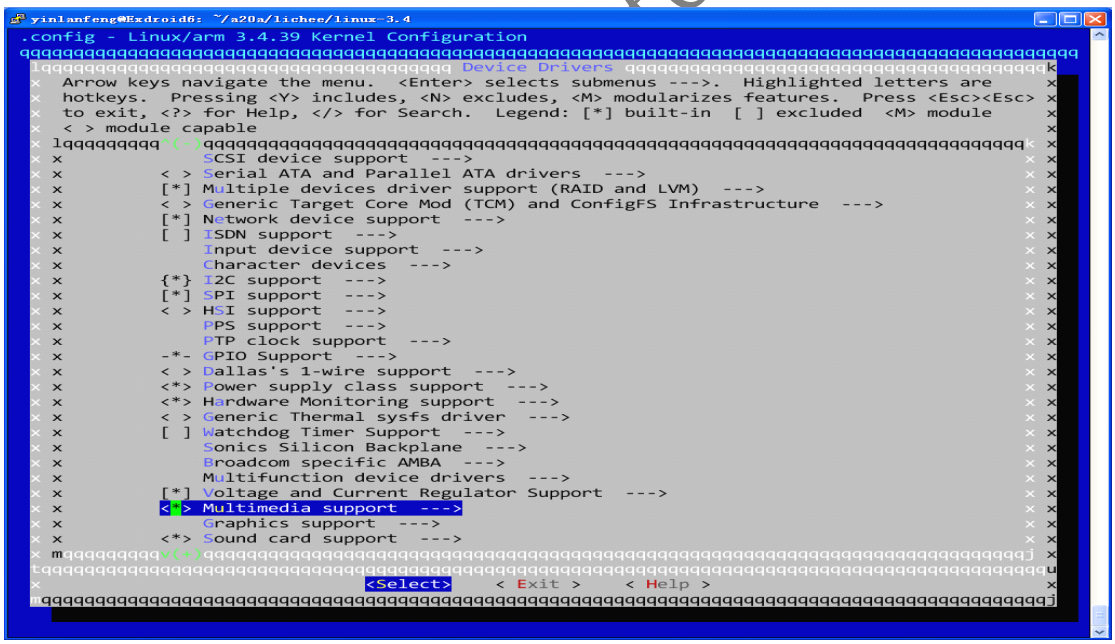


图 3 Multimedia support

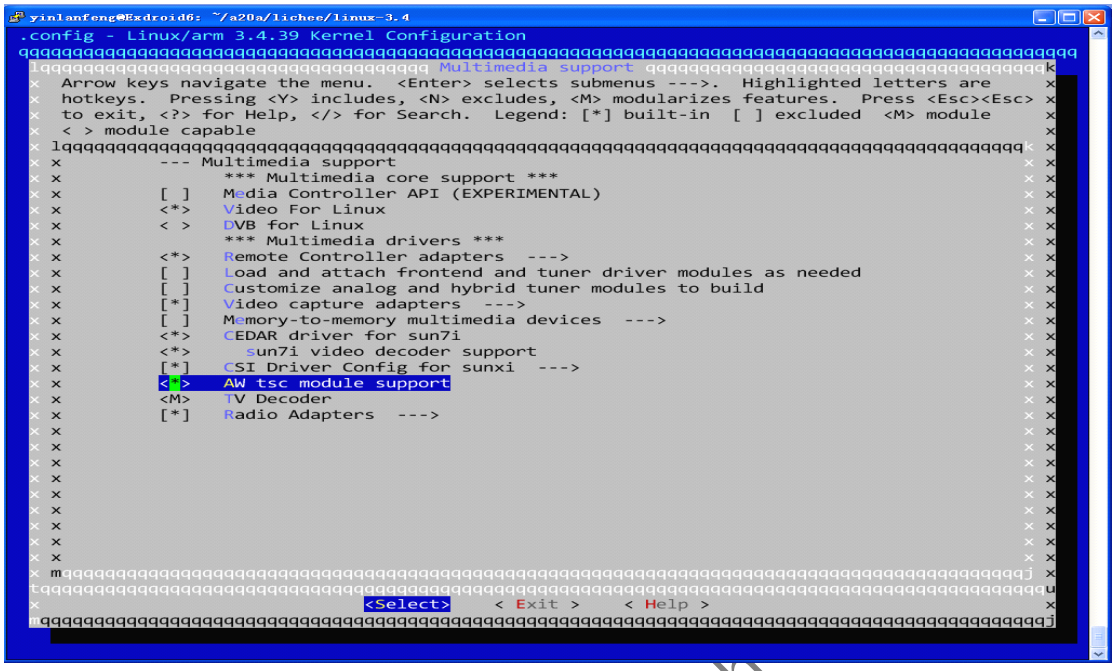


图 4 AW tsc module

2.5.3 单独包 sun7i_maple 的编译

将 sun7i_maple 包放置到编译服务器上面的任意位置上。按照如下步骤进行编译。

（1）进入 sun7i_maple,执行 source env_setup.sh

（2）make clean;make

（3）编译后的结果放置在 sun7i_maple/output

3.模块数据结构介绍

3.1 TSC Driver 数据结构

3.1.1 struct intrstatus

位于 dvb_drv_sun7i.h 中的 struct intrsatus 结构体为记录 TSF0 以及 TSF1 的中断状态的结构体。

```
struct intrstatus {
        unsigned int port0chan;
        unsigned int port0pcr;
        unsigned int port1chan;
        unsigned int port1pcr;
};
```

portXchan：为 channel 的中断状态。
PortXpcr：为 TSFX 端口是否有 pcr 的状态。

3.1.2 struct tsc_dev

位于 tscdrv.h 中的 struct tsc_dev 结构体为 driver 的关键结构体，保存驱动中使用的所有资源。

3.1.3 struct file_operations tscdev_fops

为字符设备的 file_operations 成员，供应用程序使用。

```
static struct file_operations tscdev_fops = {
        .owner                  = THIS_MODULE,
        .mmap                   = tscdev_mmap,
        .poll                   = tscdev_poll,
        .open                   = tscdev_open,
        .release                = tscdev_release,
        .llseek                 = no_llseek,
        .unlocked_ioctl = tscdev_ioctl,
};
```

3.2 TSC HAL 数据结构

（1）struct TSC_TSC_REG_LIST ，tscRegs.h 头文件中。该结构体包含了 TSC 的相关寄存器。
（2）struct TSC_TSG_REG_LIST，tscRegs.h 头文件中。该结构体包含了 TSG 的相关寄存器。
（3）struct TSC_TSF_REG_LIST，tscRegs.h 头文件中。该结构体包含了 TSF 的相关寄存器。
（4）struct TSC_REG_LIST，tscRegs.h 头文件中。该结构体包含了 TSC，TSG，TSF 的寄存器。

（5）enum TSF_CHAN_TYPE_E，tschal.h 头文件中。该枚举结构表示拿到的 TS 流的包的类型。
TSF_TP_CHAN：从 TSC 拿到的就是 TS 包数据。
TSF_PES_CHAN：从 TSC 拿到的就是 TS PES 数据。
（6）enum TSF_PORT_E，tschal.h 头文件中。该枚举结构 TSF 的端口。
TSF_PORT_0：TSF0 端口。
TSF_PORT_0：TSF1 端口。

（7）struct __DVB_RX， tscdev.h 头文件中。该结构体为整个 HAL 层的核心结构体。
（8）enum PID_FILTER_TYPE，libtsc.h 头文件中，定义 PID 的类型。

4.模块接口描述

4.1 TSC driver 关键接口描述

4.1.1 static int tscdev_open

函数原型：

```
static int tscdev_open(struct inode *inode, struct file *filp)
```

file_operations 结构体中的 open 函数的具体实现，函数的主要功能为申请以及配置 TSC 模块的时钟。

4.1.2 long tscdev_ioctl

函数原型：

```
long tscdev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
```

file_operations 结构体中的 unlocked_ioctl 函数的具体实现，函数的主要功能为通过命令提供相对应的信息给应用层。

4.1.3 static irqreturn_t tscdriverinterrupt

函数原型：

```
static irqreturn_t tscdriverinterrupt(int irq, void *dev_id)
```

为中断服务程序，读取 TSC 的相关状态且清楚状态位。

4.2 TSC HAL 关键接口描述

4.2.1 void *tscopen()

tscopen 函数获取相关节点的句柄，打开 TSC driver，设置相应寄存器等初始化工作。

4.2.2 __s32 tscioctl

函数原型：

```
__s32 tscioctl(void *handle, __u32 cmd, __s32 aux, void *pbuffer)
```

关键函数接口，根据命令进行相对应的操作。

4.2.3 static __s32 open_filter

函数原型：

```
static __s32 open_filter(void * handle, filter_register_t* filter_register)
```

根据需要获取的流进行相对应的设置，并且打开相对应的通道数。

4.2.4 __s32 tsfOpenChan

函数原型：

```
__s32 tsfOpenChan(void *                handle,
                  __u32                 pid,
                  __u8*                 pBuf,
                  __u32                 bufSize,
                  tsf_chan_type_e chanType,
                  tsf_port_e            portId,
                  __u32                 chanId,
                  __u32                 intrEnable
                  )
```

该函数主要根据 PID 使能相应寄存器。

4.2.5 __s32 tscInit(void * handle)

该函数根据 TSF 的 port 选择使能相对应的寄存器值。

4.2.6 void * dvb_rx_maintask(void * arg)

主线程回调函数，根据读取回来的中断状态，进行相对应的处理。

5.TSC 使用步骤

使用 TSC 时的步骤如下：

（1）sys_config.fex 的配置

文件位置：\lichee\tools\pack\chips\sun7i\configs\android\wing-xxx，xxx 为相对应的配置项目名称。

根据硬件设计，TSC 使用的为 port0 还是 port1，选择相应的 GPIO 接口，将 2.4.1 节描述的配置添加到相对应的 sys_config.fex 文件中。注意：tsc_used 项必须设置为 1。

（2）编译驱动，按照 2.4.2 中 menuconfig 的配置项，将 TSC 驱动编译到内核中。

（3）TSC hal 层的编译。将 TSC hal 层与应用一起编译。

6.模块开发 demo

sun7i_maple\examples\test_dvb\test_tsc 下即为 TSC 的测试历程。过程如下所示：

（1）tscopen，打开 TSC，设置相关的基础寄存器。

（2）根据需要建立音视频的分包处理线程。

1）使用命令 TSC_CMD_OPEN_PCR_DETECT，打开 pcr detect。

2）使用命令 TSC_CMD_REQUEST_CHAN_DATA，申请通道数据。

3）将数据写到指定的文件中

4）使用命令 TSC_CMD_FLUSH_CHAN_DATA，刷新 TSF 数据

7. Declaration

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.