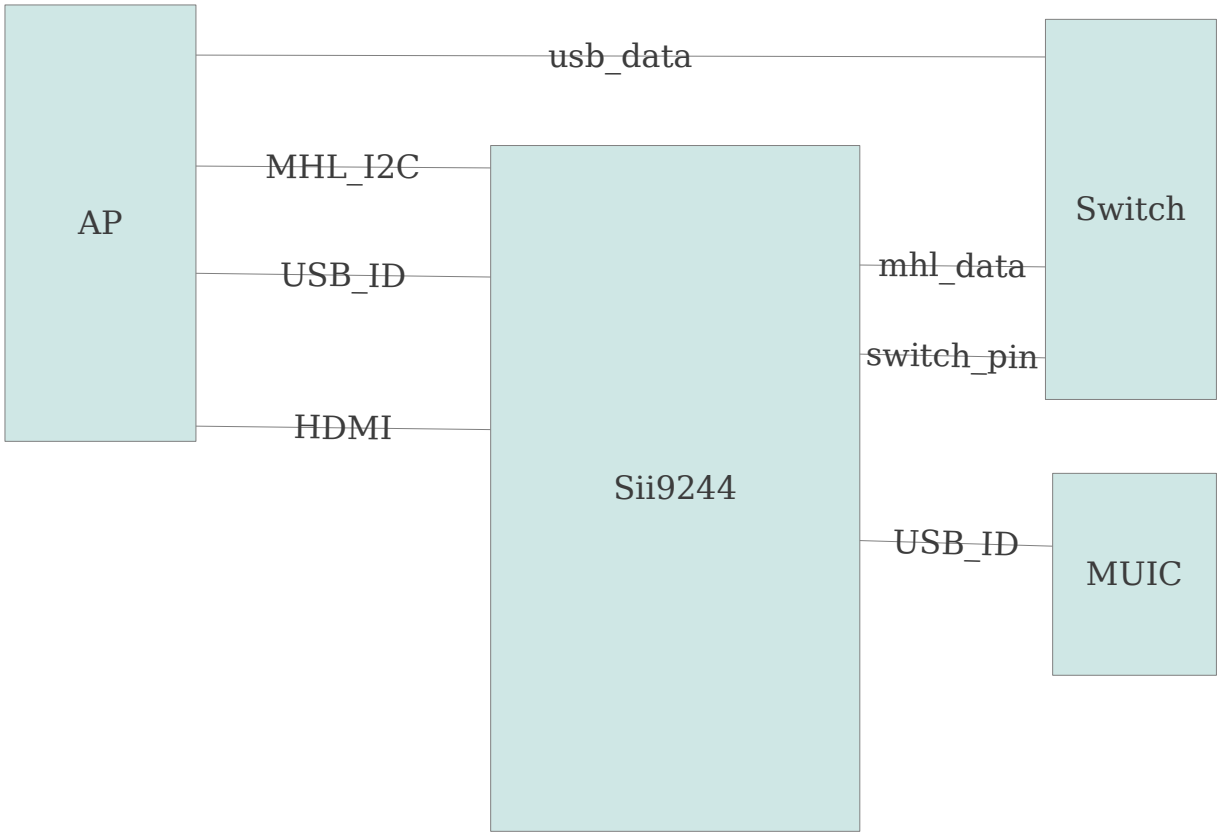
1 硬件连接框图



M040 mhl IC 连接图

一、关于硬件连接图的几点说明：

1）usb_id 这根线，在插入 usb 设备(device/host)的时候,直接通过 sii9244 与 AP 联通。
如果插入的是 mhl 则这根线是作为 sii9244 与 mhl-hdmi-bridge 的通讯线。

2）switch_pin 这条线是有被复用的。它连接到 sii9244 的 CI2CA 脚，拉高拉低会选择 sii9244 的四个 i2c_client 的地址。
同时，通过控制 sii9244 的寄存器，这条线可以控制旁边的 switch 开关。拉高拉低，选择数据是连到 usb_data 或者 mhl_data。从而实现了 micro-usb 的线的复用。

二、关于 mhl-hdmi-bridge 插入后，mhl 的发现及启动过程：

1)驱动加载后，sii9244 处于低功耗模式(low power mode) D3

2)D3 模式下，当检测到 USB_ID 脚被拉低。会激活 sii9244 进入 D2 模式

3)sii9244 测量 R_ID 的阻抗值，并保存到寄存器中。

4)sii9244 触发 RGND_MEASURED 中断。

5)AP 把 sii9244 调整到 full power mode(D0),并初始化所有寄存器。

6)HOST 会读取 R_ID 的阻抗，看是否符合 MHL 的要求。如果匹配成功，AP 向 sii9244 发送

唤醒脉冲，用来唤醒 mhl-hdmi-bridge;否则，AP 让 sii9244 返回到 D3 低功耗模式。

7)当唤醒脉冲被 peer (即 bridge)正确解读后，peer 会向 sii9244 返回一个 ACK 的信号。

8)sii9244 此时可以确定连接的建立，向 AP 发送一个 MHL_EST 的中断。

9)驱动可以按照 MHL spec 开始工作了。

三、关于 peer 唤醒：

peer 即 mhl-hdmi-bridge 是需要 sii9244 发送唤醒脉冲，才能工作的。

而 sii9244 有两种发送唤醒脉冲的方式。

A)通过 sii9244 的 WAKE_UP 引脚.

WAKE_UP 脚连接到 AP 的 GPIO 脚，让 AP 发送下列事件：

i)阻止 RGND 中断

ii)根据外部 VBUS 的提供情况，提供或阻止内部的 VBUS.

iii)AP 拉按如下时序拉 GPIO 脚(20ms high, 20ms low, 20ms high, 60ms low,20ms high, 20ms low,20ms high)

i)清楚 RGND 中断

B)通过写 I2C.(这时 WAKE_UP 脚需要连接到 GND)

过程与 A 类似，只是时序是通过写 I2C 实现。

2 软件架构

2.1 驱动软件架构

1）Sii9244 是通过一根 i2c 总线与 AP 进行通讯，sii9244 里面的寄存器分为四个页，每个页的地址不相同，功能也不一样。所以一个驱动里面有四个 i2c_client 的注册。

```
static const struct i2c_device_id sii9234_mhl_tx_id[] = {
        {"sii9244_mhl_tx", 0},
        {}
};

static const struct i2c_device_id sii9234_tpi_id[] = {
        {"sii9244_tpi", 0},
        {}
```

```
};

static const struct i2c_device_id sii9234_hdmi_rx_id[] = {
	{"sii9244_hdmi_rx", 0},
	{}
};

static const struct i2c_device_id sii9234_cbus_id[] = {
	{"sii9244_cbus", 0},
	{}
};
MODULE_DEVICE_TABLE(i2c, sii9234_tpi_id);
MODULE_DEVICE_TABLE(i2c, sii9234_hdmi_rx_id);
MODULE_DEVICE_TABLE(i2c, sii9234_cbus_id);
MODULE_DEVICE_TABLE(i2c, sii9234_mhl_tx_id);
```

2）sii9244 的驱动是通过中断驱动设备工作的。probe 中通过简单的数据结构的初始化，其他代码都在中断处理函数中。

```
request_threaded_irq(pdata->eint, NULL, sii9234_irq_thread,
		IRQF_TRIGGER_HIGH | IRQF_ONESHOT,  "sii9234", sii9234);
```

中断得到事件后，处理调度相关的 work struct 进行处理。

整个驱动是围绕下面这个数据结构进行操作

```
struct sii9234_data {
	struct mhl_platform_data	*pdata;
	wait_queue_head_t		wq;
#ifdef CONFIG_SAMSUNG_MHL_9290
	struct notifier_block		acc_con_nb;
#endif
	bool				claimed;
	u8				cbus_connected; /* wolverin */
	enum mhl_state			state;
	enum rgnd_state			rgnd;
	bool				rsen;
	atomic_t			is_irq_enabled;

	struct mutex			lock;
	struct mutex			cbus_lock;
```

```
	struct mutex			mhl_status_lock; /* wolverin */
	struct cbus_packet		cbus_pkt;
	struct cbus_packet		cbus_pkt_buf[CBUS_PKT_BUF_COUNT];
	struct device_cap		devcap;
	struct mhl_tx_status_type mhl_status_value;
#ifdef CONFIG_SII9234_RCP
	u8 error_key;
	struct input_dev	*input_dev;
#endif
#ifdef __MHL_NEW_CBUS_MSC_CMD__
	struct completion		msc_complete;
	struct work_struct		msc_work;
	int				vbus_owner;
	int				dcap_ready_status;
#endif

	struct work_struct		mhl_restart_work;
	struct work_struct		mhl_end_work;
	struct work_struct		rgnd_work;
	struct work_struct		mhl_cbus_write_stat_work;
	struct work_struct		mhl_d3_work;
#ifdef __CONFIG_TMDS_OFFON_WORKAROUND__
	struct work_struct		tmds_offon_work;
#endif
	struct timer_list	cbus_command_timer;
#ifdef CONFIG_MACH_MIDAS
	struct wake_lock	mhl_wake_lock;
#endif
	struct work_struct mhl_tx_init_work; /* wolverin */
	struct workqueue_struct *mhl_tx_init_wq;
	struct work_struct mhl_400ms_rsen_work;
	struct workqueue_struct *mhl_400ms_rsen_wq;

#ifdef CONFIG_EXTCON
	/* Extcon */
	struct extcon_specific_cable_nb extcon_dev;
```

```
        struct notifier_block extcon_nb;
        struct work_struct extcon_wq;
        bool extcon_attached;
#endif
#ifdef CONFIG_HAS_EARLYSUSPEND
        struct early_suspend early_suspend;
        bool suspend_state;
#endif
#ifdef __CONFIG_TMDS_OFFON_WORKAROUND__
        bool tmds_state;
#endif
};
```

其中有几个 work struct 经常被调度到

struct work_struct mhl_restart_work; //IC 重新加电时调用，实际调用 mhl_onoff_ex(1);

struct work_struct mhl_end_work;//IC 掉电的时候调用，实际调用 mhl_onoff_ex(0);

struct work_struct rgnd_work; // 当 mhl 插入时调用，实际调用 sii9234_detection_callback();

struct work_struct mhl_d3_work; //当 mhl 进入低功耗模式时调用 实际调用 goto_d3();

3)CBUS 通讯线需要完成的功能：MSC，RCP，RAP

MSC ： MHL Sideband Channel 主要是一些交互命令:

WRITE_BURST

WRITE_STAT_INT

READ_DEVCAP

MSC_MSG

RCP: Remote Control Protocol (遥控协议)

这个协议用于把遥控器的按键值传到 MHL，让手机端来处理按键。

包括命令：

RCP

RCPK

RCPE

RAP: Request Action Protocol

这个协议用于在请求者和应答者之间传递命令

包括如下命令：

RAP

RAPK

struct sii9234_data 结构体中的成员

```
struct cbus_packet              cbus_pkt;//当前处理的命令
struct cbus_packet              cbus_pkt_buf[CBUS_PKT_BUF_COUNT];//命令缓存
```

就是用来处理相关命令的。

这些命令通过内核 list 结构组织成为一个队列。然后在成员

```
struct work_struct              msc_work;
```

被调度时处理这些命令。

整个驱动结构清晰，但相关文档介绍不够详细，很多寄存器没有介绍到，或者有很少的描述。

2.2 Android 端软件架构

由于 MHL 是由硬件即驱动直接实现，上层并不与之交互。此处略过。