```
247         return ret;
248     }
249
250     return 0;
251 }
252
253  /*
254   * @description       : I²C驱动的probe函数,当驱动与
255   *                      设备匹配以后此函数就会执行
256   * @param - client    : I²C设备
257   * @param - id        : I²C设备ID
258   * @return            : 0,成功;负值,失败
259   */
260 static int ft5x06_ts_probe(struct i2c_client *client,
                               const struct i2c_device_id *id)
261 {
262     int ret = 0;
263
264     ft5x06.client = client;
265
266     /* 1、获取设备树中的中断和复位引脚 */
267     ft5x06.irq_pin = of_get_named_gpio(client->dev.of_node,
                         "interrupt-gpios", 0);
268     ft5x06.reset_pin = of_get_named_gpio(client->dev.of_node,
                         "reset-gpios", 0);
269
270     /* 2、复位FT5x06 */
271     ret = ft5x06_ts_reset(client, &ft5x06);
272     if(ret < 0) {
273         goto fail;
274     }
275
276     /* 3、初始化中断 */
277     ret = ft5x06_ts_irq(client, &ft5x06);
278     if(ret < 0) {
279         goto fail;
280     }
281
282     /* 4、初始化FT5X06 */
283     ft5x06_write_reg(&ft5x06, FT5x06_DEVICE_MODE_REG, 0);
284     ft5x06_write_reg(&ft5x06, FT5426_IDG_MODE_REG, 1);
285
286     /* 5,input设备注册 */
287     ft5x06.input = devm_input_allocate_device(&client->dev);
288     if (!ft5x06.input) {
289         ret = -ENOMEM;
290         goto fail;
291     }
292     ft5x06.input->name = client->name;
293     ft5x06.input->id.bustype = BUS_I2C;
294     ft5x06.input->dev.parent = &client->dev;
295
```

USB 3.0 引入了全双工数据传输,USB 2.0 的 480Mbps 为半双工。USB 3.0 中两根线用于发送数据,另外两根用于接收数据。在 USB 3.0 的基础上又提出了 USB 3.1、USB 3.2 等规范,USB 3.1 理论传输速度提升到了 10Gbps,USB 3.2 理论传输速度为 20Gbps。为了规范 USB 3.0 标准的命名,USB-IF 公布了最新的 USB 命名规范,原来的 USB 3.0 和 USB 3.1 命名将不会采用,所有的 3.0 版本的 USB 都命名为 USB 3.2,以前的 USB 3.0、USB 3.1 和 USB 3.2 分别叫作 USB 3.2 Gen1、USB 3.2 Gen2、USB 3.2 Gen 2X2。

USB 4.0:目前还在标准定制中,还没有设备搭载,据说是在 Intel 的雷电接口上改进而来。USB 4.0 的速度将提升到了 40Gbps,最高支持 100W 的供电能力,只需要一根线就可以完成数据传输与供电,极大地简化了设备之间的链接线数。

如果按照接口类型划分:USB 就要分为很多种了。最常见的就是 USB A 插头和插座,如图 30-1 所示。

如果使用过 JLINK 调试器,那么应该见过 USB B 插头和插座,USB B 插头和插座如图 30-2 所示。

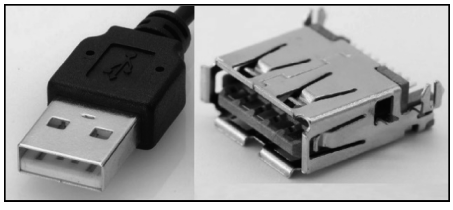

图 30-1　USB A 插头(左)和插座(右)

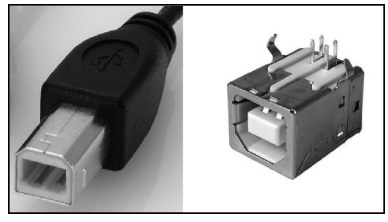

图 30-2　USB B 插头(左)和插座(右)

USB 插头在不断缩小,由此产生了 Mini USB 接口,正点原子的 I.MX6ULL-ALPHA 开发板使用的就是 Mini USB。Mini USB 插头和插座如图 30-3 所示。

比 Mini USB 更小的就是 Micro USB 接口了,以前的智能手机基本都是 Micro USB 接口的。Micro USB 插头和插座如图 30-4 所示。

图 30-3　Mini USB 插头(左)和插座(右)

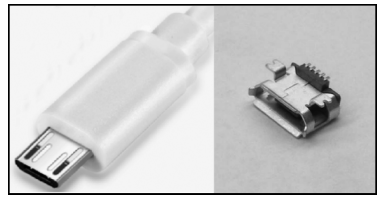

图 30-4　Micro USB 插头(左)和插座(右)

现在最流行的就是 USB TypeC 了,USB TypeC 插头和插座如图 30-5 所示。

30.1.2　USB 电气特性

由于正点原子 I.MX6U-ALPHA 开发板使用的 Mini USB 接口,因此我们就以 Mini USB 为例讲解一下 USB 的基本电气属性。Mini USB 线一般都是一头为 USB A 插头,另一头为 Mini USB 插头。一共有 4 个触点,也就是 4 根线,这 4 根线的顺序如图 30-6 所示。

```
53          struct mutex mutex;
54          struct {
55              spinlock_t spinlock;
56              unsigned long spinlock_flags;
57          };
58      };
59      regmap_lock lock;
60      regmap_unlock unlock;
61      void * lock_arg;                /* This is passed to lock/unlock functions */
62
63      struct device * dev;            /* Device we do I/O on */
64      void * work_buf;                /* Scratch buffer used to format I/O */
65      struct regmap_format format;    /* Buffer format */
66      const struct regmap_bus * bus;
67      void * bus_context;
68      const char * name;
69
70      bool async;
71      spinlock_t async_lock;
72      wait_queue_head_t async_waitq;
73      struct list_head async_list;
74      struct list_head async_free;
75      int async_ret;
...
89      unsigned int max_register;
90      bool ( * writeable_reg)(struct device * dev, unsigned int reg);
91      bool ( * readable_reg)(struct device * dev, unsigned int reg);
92      bool ( * volatile_reg)(struct device * dev, unsigned int reg);
93      bool ( * precious_reg)(struct device * dev, unsigned int reg);
94      const struct regmap_access_table * wr_table;
95      const struct regmap_access_table * rd_table;
96      const struct regmap_access_table * volatile_table;
97      const struct regmap_access_table * precious_table;
98
99      int ( * reg_read)(void * context, unsigned int reg,
                          unsigned int * val);
100     int ( * reg_write)(void * context, unsigned int reg,
                          unsigned int val);
...
147     struct rb_root range_tree;
148     void * selector_work_buf;   /* Scratch buffer used for selector */
149 };
```

要使用 regmap，肯定要先给驱动分配一个具体的 regmap 结构体实例，稍后会讲解如何分配 regmap 实例。大家可以看到示例代码 31-1 中第 90～100 行有很多的函数以及 table，这些需要驱动编写人员根据实际情况选择性地初始化。regmap 的初始化通过结构体 regmap_config 来完成。

3. regmap_config 结构体

顾名思义，regmap_config 结构体就是用来初始化 regmap 的，这个结构体也定义在 include/linux/regmap.h 文件中，结构体内容如下：

返回值——0,注册成功;负值,注册失败。

4. 注销 net_device

既然有注册,必然有注销,注销 net_device 使用函数 unregister_netdev(),函数原型如下:

```
void unregister_netdev(struct net_device * dev)
```

函数参数含义如下:

dev——要注销的 net_device 指针。

返回值——无。

35.3.2 net_device_ops 结构体

net_device 有一个非常重要的成员变量——netdev_ops,为 net_device_ops 结构体指针类型,这就是网络设备的操作集。net_device_ops 结构体定义在 include/linux/netdevice.h 文件中,net_device_ops 结构体中都是一些以 ndo_开头的函数,这些函数就需要网络驱动编写人员去实现,不需要全部都实现,根据实际驱动情况实现其中一部分即可。结构体内容如下所示(结构体比较大,这里有省略):

示例代码 35-8 net_device_ops 结构体

```
1   struct net_device_ops {
2       int          ( * ndo_init)(struct net_device * dev);
3       void         ( * ndo_uninit)(struct net_device * dev);
4       int          ( * ndo_open)(struct net_device * dev);
5       int          ( * ndo_stop)(struct net_device * dev);
6       netdev_tx_t  ( * ndo_start_xmit) (struct sk_buff * skb,
7                            struct net_device * dev);
8       u16          ( * ndo_select_queue)(struct net_device * dev,
9                            struct sk_buff * skb,
10                           void * accel_priv,
11                           select_queue_fallback_t fallback);
12      void         ( * ndo_change_rx_flags)(struct net_device * dev,
13                               int flags);
14      void         ( * ndo_set_rx_mode)(struct net_device * dev);
15      int          ( * ndo_set_mac_address)(struct net_device * dev,
16                               void * addr);
17      int          ( * ndo_validate_addr)(struct net_device * dev);
18      int          ( * ndo_do_ioctl)(struct net_device * dev,
19                             struct ifreq * ifr, int cmd);
20      int          ( * ndo_set_config)(struct net_device * dev,
21                             struct ifmap * map);
22      int          ( * ndo_change_mtu)(struct net_device * dev,
23                           int new_mtu);
24      int          ( * ndo_neigh_setup)(struct net_device * dev,
25                           struct neigh_parms * );
26      void         ( * ndo_tx_timeout) (struct net_device * dev);
...
36  # ifdef CONFIG_NET_POLL_CONTROLLER
37      void         ( * ndo_poll_controller)(struct net_device * dev);
38      int          ( * ndo_netpoll_setup)(struct net_device * dev,
```