1. RS232驅動:

* UART是很常見的驅動，在Linux中分成TTL和RS232，驅動原理是一樣的，本專案就是要簡單介紹在Linux系統中如何實現Serial Port功能。

1. Linux底下UART框架:

* 和I2C，SPI一樣，Linux Kernel也提供UART驅動框架，我們只需要完成驅動框架即可，uart\_driver 結構體表示 UART 驅動，uart\_driver 定義在 include/linux/serial\_core.h 文件中
* UART驅動加載卸載API函數: 加載驅動的時候通過 uart\_register\_driver 函數向系統註冊這個驅動。註銷驅動的時候也需要註銷掉前面註冊的 uart\_driver，需要用到 uart\_unregister\_driver 函數。

(如下表一整理)

(表一:UART驅動註冊/註銷函式)

|  |  |  |
| --- | --- | --- |
| **操作** | **函數** | **描述** |
| 加載驅動 | uart\_register\_driver | 通過此函數向系統註冊驅動。 |
| 註銷驅動 | uart\_unregister\_driver | 通過此函數註銷之前註冊的 uart\_driver 驅動。 |

* 每個UART都有一個uart\_port，uart\_port表示一個具體的port，uart\_port定義在include/linux/serial\_core.h文件中。而我們需要把uart\_port和uart\_driver結合起來，就需要透過uart\_add\_one\_port函式，卸載 UART 驅動的時候也需要將uart\_port從相應的uart\_driver 中移除，需要用到uart\_remove\_one\_port函數。

(如表二整理)

(表二: uart\_add\_one\_port/uart\_remove\_one\_port功能整理)

|  |  |  |
| --- | --- | --- |
| **操作** | **函數** | **描述** |
| 添加 UART 埠 | uart\_add\_one\_port | 將指定的 UART 埠添加到 UART 驅動程序的埠列表中。 |
| 移除 UART 埠 | uart\_remove\_one\_port | 卸載 UART 驅動時，將 uart\_port 從相應的 uart\_driver 中移除。 |

* uart\_port中的ops成員變數很重要，因為ops包含了針對UART具體的驅動函數，Linux系統收發數據最終調用的都是ops中的函數。ops是uart\_ops類型的結構體指針變量，uart\_ops定義在 include/linux/serial\_core.h文件中。
* 接下來我們看一下STM已經寫好的驅動程式。(如下圖一，表三)



(圖一: uart\_ops結構體內容及對應操作函數)

(表三:uart\_ops結構體內容及對應操作函數)

|  |  |  |
| --- | --- | --- |
| **函數指針** | **對應函數** | **描述** |
| tx\_empty | stm32\_usart\_tx\_empty | 檢查發送緩衝區是否為空。 |
| set\_mctrl | stm32\_usart\_set\_mctrl | 設置 modem 控制信號。 |
| get\_mctrl | stm32\_usart\_get\_mctrl | 獲取 modem 控制信號的狀態。 |
| stop\_tx | stm32\_usart\_stop\_tx | 停止數據發送。 |
| start\_tx | stm32\_usart\_start\_tx | 開始數據發送。 |
| throttle | stm32\_usart\_throttle | 節流發送。 |
| unthrottle | stm32\_usart\_unthrottle | 取消節流發送。 |
| stop\_rx | stm32\_usart\_stop\_rx | 停止接收數據。 |
| break\_ctl | stm32\_usart\_break\_ctl | 控制斷續信號。 |
| startup | stm32\_usart\_startup | 初始化 UART 設備。 |
| shutdown | stm32\_usart\_shutdown | 關閉 UART 設備。 |
| flush\_buffer | stm32\_usart\_flush\_buffer | 清空發送和接收緩衝區。 |
| set\_termios | stm32\_usart\_set\_termios | 設置串口的通信參數。 |
| pm | stm32\_usart\_pm | 電源管理相關的操作。 |
| type | stm32\_usart\_type | 返回 UART 類型。 |
| release\_port | stm32\_usart\_release\_port | 釋放 UART 埠。 |
| request\_port | stm32\_usart\_request\_port | 請求使用 UART 埠。 |
| config\_port | stm32\_usart\_config\_port | 配置 UART 埠的硬體參數。 |
| verify\_port | stm32\_usart\_verify\_port | 驗證 UART 埠是否可用。 |

※總結:

可以看出UART本質上也是platform驅動框架，大致上的驅動編寫流程如下表四。

(表四:Linux下UART驅動開發流程)

|  |  |  |
| --- | --- | --- |
| **步驟** | **描述** | **關鍵函數/結構體** |
| 1. 定義 UART 操作 | 定義 uart\_ops 結構體，設置各種操作函數指針。 | struct uart\_ops |
| 2. 定義 UART 埠 | 定義 uart\_port 結構體，描述具體的 UART 埠硬體信息。 | struct uart\_port |
| 3. 添加 UART 埠 | 使用 uart\_add\_one\_port 函數將 UART 埠添加到驅動程序中。 | uart\_add\_one\_port(struct uart\_driver \*, struct uart\_port \*) |
| 4. 註冊 UART 驅動 | 使用 uart\_register\_driver 函數註冊 UART 驅動。 | uart\_register\_driver(struct uart\_driver \*) |
| 5. 初始化 UART | 通過 startup 函數指針初始化 UART 硬體。 | startup 函數 |
| 6. 配置 UART 埠 | 使用 set\_termios 函數配置 UART 的通信參數。 | set\_termios 函數 |
| 7. 發送數據 | 使用 start\_tx 函數開始數據發送。 | start\_tx 函數 |
| 8. 接收數據 | 使用 stop\_rx 函數停止數據接收。 | stop\_rx 函數 |
| 9. 清空緩衝區 | 使用 flush\_buffer 函數清空發送和接收緩衝區。 | flush\_buffer 函數 |
| 10. 註銷 UART 埠 | 使用 uart\_remove\_one\_port 函數移除 UART 埠。 | uart\_remove\_one\_port(struct uart\_driver \*, struct uart\_port \*) |
| 11. 註銷 UART 驅動 | 使用 uart\_unregister\_driver 函數註銷 UART 驅動。 | uart\_unre |