MediaTek

Customer Device Driver Document

UART Driver

Documents Number:

Preliminary Information

Revision: 0.1

Release Date: Nov, 03, 2003

Revision History

Revision	Date	Author	Comments
0.1	11/03/2003	Arthur Shieh	Draft version, revised from MTK Device Driver design document

Table of contents

Figures index

1 Introduction

1.1 Overview

This document describes the function of the UART driver. This driver is a generic interface to work with PPP, CSD, ATCI, L2R, and TST tasks. For system developers, the customization work is to adjust the ring buffer size.

1.2 References

- M16x50, Enhanced UART with FIFOs and IrDA support product specification
- MT6205B GSM baseband Processor Datasheet

2 Architecture

2.1 Block Diagram

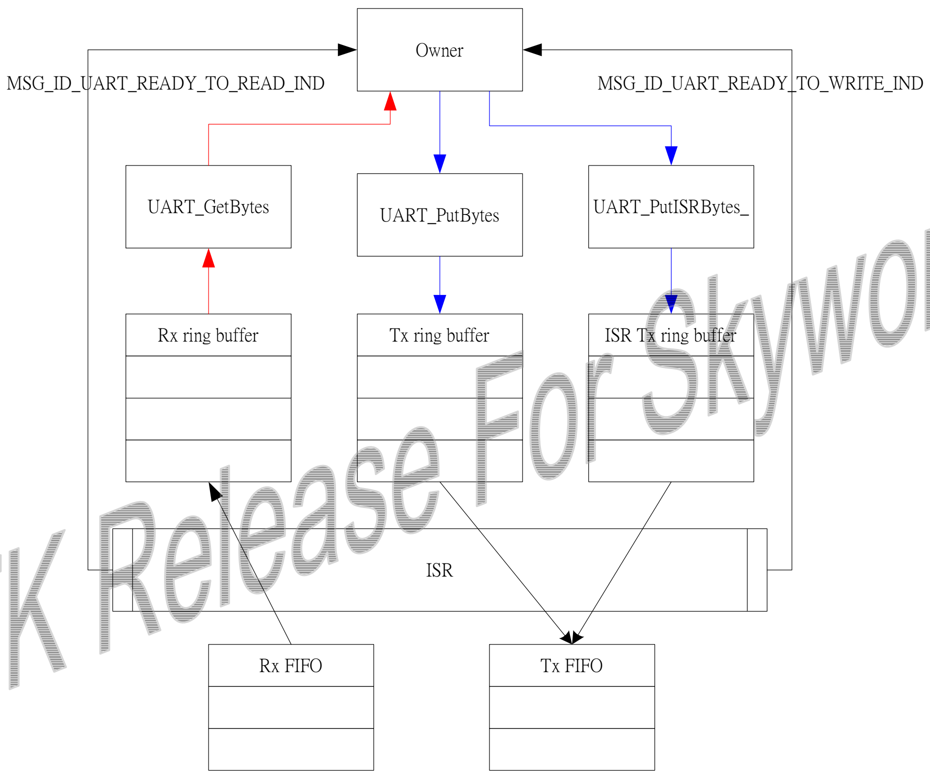

Figure 1 UART Driver Buffer Structure Block Diagram

2.2 Functional Overview

In UART Driver, each UART port has 3 ring buffers, Rx, Tx and ISR Tx. For UART operations, the tasks, which request UART service in system, will send or receive data through Rx or Tx ring buffer. Only the system debug/trace information will go through ISR Tx ring buffer, which has higher priority, to ensure timely delivery of debug/trace information.

For system developers, it is necessary to adjust the ring buffer length if the UART status is always busy, that is, the ring buffer is always stuffed with too many request to handle other requests. The physical UART data transmission and reception is handled by interrupt.

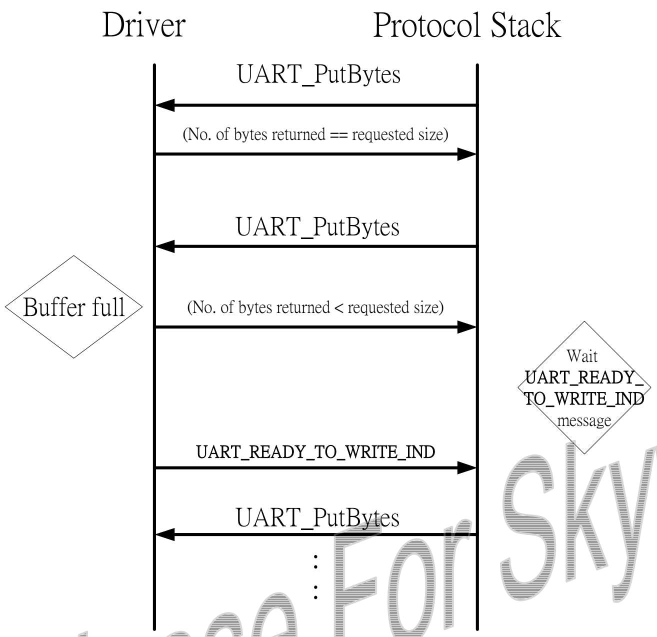
Driver
Protocol Stack
UART_PutBytes
(No. of bytes returned == requested size)
UART_PutBytes
Buffer full
(No. of bytes returned < requested size)
Wait UART_READY_TO_WRITE_IND message
UART_READY_TO_WRITE_IND
UART_PutBytes

Figure 2 UART flow control for transmitting Data

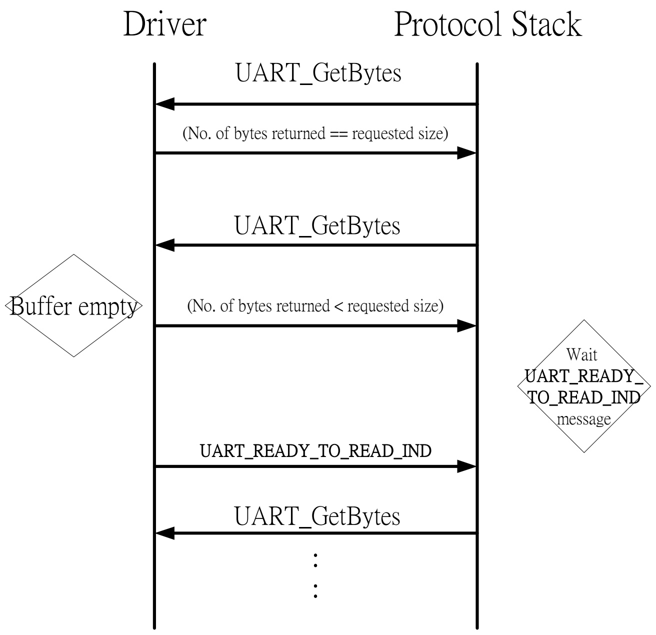
Driver
Protocol Stack
UART_GetBytes
(No. of bytes returned == requested size)
UART_GetBytes
Buffer empty
(No. of bytes returned < requested size)
Wait UART_READY_TO_READ_IND message
UART_READY_TO_READ_IND
UART_GetBytes

Figure 3 UART flow control for receiving Data

The UART driver implements both hardware level and software-level flow control.

The software flow control is implemented over the ring buffer management. The protocol stack, or the tasks requesting UART service have to judge if the ring buffers have enough space for transmitting or receiving data. If returned size information from driver is not matching the requested size , then the task which request the UART service needs to wait until a incoming message showing the Rx buffer is not empty or the Tx buffer is not full. Then the next transaction can be carried on.

For embedded flow control, MT6205 implements MODEM control and Xon/Xoff flow control. For details please refer to M16550/M16C450, Enhanced UART with FIFOs and IrDA support product specification and MT6205B GSM baseband Processor Datasheet.

3 Library Module

3.1 Module

drv.lib

Interface\hwdrv
uart_sw.h

custom\drv\misc_drv\{platform}
uart_def.c

3.2 Data Structures

3.2.1 UART_PORT

```
typedef enum {
            uart_port1=0,
            uart_port2
} UART_PORT;
```

3.2.2 IO_level

```
typedef enum {
            io_low,
            io_high
} IO_level;
```

3.2.3 UART_buffer

```
typedef enum {
            RX_BUF,
            TX_BUF
} UART_buffer;
```

3.2.4 UART_baudrate

```
typedef enum
{
            UART_BAUD_AUTO,
            UART_BAUD_75,
            UART_BAUD_150,
            UART_BAUD_300,
            UART_BAUD_600,
            UART_BAUD_1200,
            UART_BAUD_2400,
            UART_BAUD_4800,
        UART_BAUD_7200,
        UART_BAUD_9600,
        UART_BAUD_14400,
        UART_BAUD_19200,
        UART_BAUD_28800,
        UART_BAUD_38400,
        UART_BAUD_57600,
        UART_BAUD_115200
}UART_baudrate;
```

3.2.5 UART_bitsPerCharacter

```
typedef enum {
        len_5=5,
        len_6,
        len_7,
        len_8
} UART_bitsPerCharacter;
```

3.2.6 UART_stopBits

```
typedef enum {
        sb_1=1,
        sb_2,
        sb_1_5
} UART_stopBits;
```

3.2.7 UART_parity

```
typedef enum {
        pa_none=0,
        pa_odd,
        pa_even
} UART_parity;
```

3.2.8 UART_flowCtrlMode

```
typedef enum {
        fc_none=1,
        fc_hw,
        fc_sw
} UART_flowCtrlMode;
```

3.2.9 UARTDCBStruct

Definition:

```
    typedef struct
    {
```

```
                UART_baudrate                 baud;
                UART_bitsPerCharacter         dataBits;
                UART_stopBits                 stopBits;
                UART_parity                   parity;
                UART_flowCtrlMode             flowControl;
                uint8                         xonChar;
                uint8                         xoffChar;
                bool                          DSRCheck;
        } UARTDCBStruct;
```

This structure contains all the parameters used to configure the serial device. The following table contains these data elements and brief descriptions of them.

Members:

Members	Description
baud	Transmission rate(bits/sec).
dataBits	Size of character.
stopBits	Number of stop bits attached to each character.
parity	Type of parity check.
flowControl	Type of flow control.
xonChar	XON character for software flow control.
xoffChar	XOFF character for software flow control.
DSRCheck	DSRCheck =1➔ Target side will check DSR signal to decide whether the PC are connected or not. DSRCheck =0➔ Target side don't use DSR signal to decide whether the PC are connected or not.

3.2.10 UART_ESCDetectStruct

Definition:

```
        typedef struct
        {
                uint8      EscChar;
                uint16   GuardTime;
        } UART_ESCDetectStruct;
```

Members:

Members	Description
EscChar	Three "EscChar" will be treated as an escape sequence.
GuardTime	Denotes the minimal duration of the rest before the first and after the last character of the escape sequence, and the maximal receiving duration of each escape character. The unit of this parameter is milliseconds. If this character is zero, the driver will not detect the escape sequences.

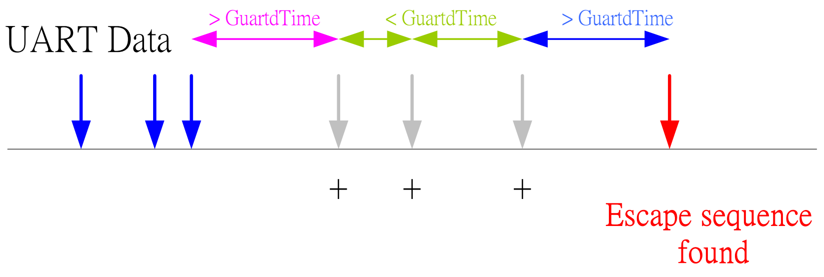

Figure 4 Escape sequence detection

3.3 Function

3.3.1 UART_Open

kal_bool UART_Open(UART_PORT port, module_type owner)

This function is to allocate the memory in UART driver, refresh UART FIFO, enable UART interrupt,…etc. This function must also be called before any action to access UART interface.

Parameters:

Members	Description
Port	UART port.
owner	owner id. The owner after the port is opened.

Return value:

Return value	Description
KAL_TRUE(1)	UART open successfully.
KAL_FALSE(0)	UART open fail.

Example:
None

3.3.2 UART_Close

void UART_Close (UART_PORT port)

This function is to frees any memory allocated by UART_Open, and disable UART interrupt,…etc. This function must also be called before any actions to access UART interface.

Parameters:

Members	Description
Port	UART port.

Return value:

Return value	Description
TRUE(1)	UART close successfully.
FALSE(0)	UART close fail.

Example:
None

3.3.3 UART_SetOwner

void UART_SetOwner (UART_PORT port, module_type owner)

This function is to register the owner of UART.

Parameters:

Members	Description
port	UART port.
owner	Owner id.

Return value:
None

Example:
None

3.3.4 UART_SetDCBConfig

void UART_SetDCBConfig (UART_PORT port, UARTDCBStruct *DCB)

This function is to setup the driver configuration. The parameters are specified in a UARTDCBStruct structure type. The user should allocate memory for the configuration and driver will set these parameters. Besides, Driver will copy the content locally, and keep this data itself.

Parameters:

Members	Description
port	UART port.
DCB	Pointer to the device control block of UART

Return value:
None

Example:
UARTDCBStruct DCBdata;
UART_SetDCBConfig (uart_port1, &DCBdata);

3.3.5 UART_ReadDCBConfig

void UART_ReadDCBConfig (UART_PORT port, UARTDCBStruct *DCB)

This function is to read the driver configuration. The parameters are specified in a UARTDCBStruct structure type. The user should allocate memory for the configuration, and driver will copy the content of current DCB to the address which the *DCB point to.

Parameters:

Members	Description
port	UART port.
DCB	Pointer to the device control block of UART.

Return value:
None

Example:
UARTDCBStruct DCBdata;
UART_ReadDCBConfig(uart_port1, &DCBdata);

3.3.6 UART_ConfigEscape

void UART_ConfigEscape (UART_PORT port, kal_uint8 EscChar, kal_uint16 ESCGuardtime)

This function is to change the setting of escape sequence. After calling this function, UART will use new escape configuration to detect escape sequence.

Parameters:

Members	Description
port	UART port.
EscChar	The character of the escape sequence
ESCGuardtime	The guard time of the escape sequence

Return value:
None

Example:
None

3.3.7 UART_GetBytes

kal_uint16 UART_GetBytes(UART_PORT port, kal_uint8 *Buffaddr, kal_uint16 Length, kal_uint8 *status)

This function is to receive data from UART.

Parameters:

Members	Description
port	UART port.

Buffaddr	Pointer to the data addr.
Length	the maximum length
status	bit 0 =1, Escape sequence is detected. =0, Escape sequence is not detected. bit 1 =1, break signal is detected. =0, break signal is detected. NOTE: 1.When Escape sequence/Break signal is detected, the data in RX/TX buffer is cleared. 2.When Escape sequence is detected, the next call of UART_GetBytes will return no data but Escape detected status set.

Return value:
The length of bytes actually read.

Example:
None

3.3.8 UART_PutBytes

kal_uint16 UART_PutBytes(UART_PORT port, kal_uint8 *Buffaddr, kal_uint16 Length)

This function is to send data to UART.
Note: This function is only called in **task** level.

Parameters:

Members	Description
port	UART port.
Buffaddr	Pointer to the data addr.
Length	the maximum length

Return value:
The length of bytes actually written.

Example:
None

3.3.9 UART_SetFlowCtrl

void UART_SetFlowCtrl(UART_PORT port, kal_bool XON)

This function is to configure the UART flow control. By default, the driver will keep flow control itself, and this function will be a null function.
Note: XON means flow control on, not means software flow control.
HW or SW flow control is set in DCB structure.

Parameters:

Members	Description
port	UART port.
XON	=1, The flow between PC and Target is connected. =0, The flow between PC and Target is closed.

Return value:
None

Example:
None

3.3.10 UART_CtrlDTR

void UART_CtrlDTR (UART_PORT port, IO_level SDTR)

This function is to control the DTR functional pin.

Parameters:

Members	Description
Port	UART port.
SDTR	= IO_high, DTR on (DTR line is at low level) = IO_low, DTR off (DTR line is at high level)

Return value:
None

Example:
None

3.3.11 UART_CtrlDCD

void UART_CtrlDCD (UART_PORT port, IO_level SDCD)

This function is to control the DCD functional pin.

Parameters:

Members	Description
Port	UART port.
SDCD	= IO_high, DCD on = IO_low, DCD off

Return value:
None

Example:

None

3.3.12 UART_CtrlRI

void UART_CtrlRI (UART_PORT port, IO_level SRI)

This function is to control the RING functional pin.

Parameters:

Members	Description
Port	UART port.
SRI	= IO_high, RING on = IO_low, RING off

Return value:
None

Example:
None

3.3.13 UART_ReadHWStatus

void UART_ReadHWStatus (UART_PORT port, IO_level *SDSR, IO_level *SCTS)

This function is to read the status of the DSR and CTS functional pin.

Parameters:

Members	Description
port	UART port.
SDSR	= IO_high, DSR on (DSR line is at low level) = IO_low, DSR off (DSR line is at high level)
SCTS	= IO_high, CTS on (CTS line is at low level) = IO_low, CTS off (CTS line is at high level)

Return value:
None

Example:
None

3.3.14 UART_CtrlBreak

void UART_CtrlBreak (UART_PORT port, IO_level SBREAK)

This function is to read the status of the DSR and CTS functional pin.

Parameters:

Members	Description
port	UART port.
SBREAK	= IO_high, set break signal = IO_low, clear break signal

Return value:
None

Example:
None

3.3.15 UART_Purge

void UART_Purge (UART_PORT port, UART_buffer dir)

This function is to clean the UART FIFO, not UART ring buffer.

Parameters:

Members	Description
port	UART port.
dir	UART_buffer (RX_BUF=0/TX_BUF=1)

Return value:
None

Example:
None

3.3.16 UART_GetBytesAvail

kal_uint16 UART_GetBytesAvail(UART_PORT port)

This function is to specify how many bytes in UART Rx ring buffer.

Parameters:

Members	Description
port	UART port.

Return value:

Return value	Description
0~0xffff	The total number of received data in ring buffer

Example:
None

3.3.17 UART_GetTxRoomLeft

kal_uint16 UART_GetTxRoomLeft(UART_PORT port)

This function is to specify how much space residual in UART Tx ring buffer.

Parameters:

Members	Description
port	UART port.

Return value:

Return value	Description
0~0xffff	The residual space in Tx ring buffer

Example:
None

3.3.18 UART_GetTxISRRoomLeft

kal_uint16 UART_GetTxISRRoomLeft(UART_PORT port)

This function is to specify how much space residual in UART TxISR ring buffer.

Parameters:

Members	Description
port	UART port.

Return value:

Return value	Description
0~0xffff	The residual space in TxISR ring buffer

Example:
None

3.3.19 UART_ClrRxBuffer

void UART_ClrRxBuffer(UART_PORT port)

This function is to clear UART Rx ring buffer.

Parameters:

Members	Description
port	UART port.

Return value:
None

Example:
None

3.3.20 UART_ClrTxBuffer

void UART_ClrTxBuffer(UART_PORT port)

This function is to clear UART Tx ring buffer.

Parameters:

Members	Description
port	UART port.

Return value:
None

Example:
None

3.4 Message

3.4.1 UART_READY_TO_WRITE_IND

Description:
This primitive is used to notify protocol stack to write data to driver.

Local Parameter:
N/A

Reference:
N/A

3.4.2 UART_READY_TO_READ_IND

Description:
This primitive is used to notify protocol stack to read data to driver

Local Parameter:
N/A

Reference:
N/A

3.4.3 UART_ESCAPE_DETECTED_IND

Description:

This primitive is used to notify protocol stack that escape sequence is detected.

Local Parameter:

N/A

Reference:

N/A

3.4.4 UART_DSR_CHANGE_IND

Description:

This primitive is used to notify protocol stack about changing of DSR status.

Local Parameter:

N/A

Reference:

N/A