Kneron Inc

Document Name: **KL520 UART Driver API**

KL520 UART Driver API
Kneron Inc

Table of Contents

1 Introduction

1.1 Overview

This document describes the Kneron Mozard UART driver API design. It can be used by customers as an users guide for Kneron UART API.

The hardware board has 5 UART ports, so the driver can support up to 5 UART devices.

The UART hardware can support SIR (IrDA 1.3 Serial IR) which can run up to 115200bps, and support FIR IrDA Fast IR which can run up to 4M bps. While this version driver API cannot support SIR/FIR yet, will do further development up on customers' requests.

1.2 Acronym

Name	Meaning
Tx	Transmit
Rx	Receive
IrDA	Infrared Data Association
SIR	Serial IrDA
FIR	Fast IrDA
CTS	Clear to Send
DTS	Data Set Ready
RI	Ring Indicator
DCD	Data Carrier Detect
RTS	Request to Send
DTR	Data Terminal Ready

2 High Level Design

2.1 Design Considerations

Here are the design highlight points:

- The architecture adopts a lightweight non-thread design
- ISR driven architecture.
- Can support both synchronous and asynchronous mode
- Utilizes FIFO advantage to reduce interrupts and improve robust to accommodate more latency than normal.

2.2 API functions

2.2.1 Initialization

Prototype:
void kdp_uart_init(void)

This API needs to be called at least once after powering on, it will do necessary initializations.

2.2.2 Open Driver

Prototype:
kdp_uart_hdl_t kdp_uart_open(uint8_t com_port, uint32_t mode, kdp_uart_callback_t cb)

This API will open a UART device (com_port: 0-5) for use. It will return a UART device handle for future device reference.

The client can choose work mode: asynchronization or synchronization.
Synchronization mode will poll the hardware status to determine send/receiving point, it will consume more power and introduce more delay to system execution. But in the case of non-thread light weight environment, such as message log function, this mode is easy and suitable.
Asynchronization mode lets the driver interrupt driven, save more system power and more efficient, the client needs to have a thread to listen/wait for the event/signal sent from callback function.
Callback function parameter 'cb' will be registered with this device which is mandatory for async mode, will be invoked whenever Tx/Rx complete or timeout occur. This callback function should be very thin, can only be used to set flag or send signals.

2.2.3 Query capability

Prototype:
KDP_USART_CAPABILITIES* kdp_uart_get_capability(kdp_uart_hdl_t hanle)

The client may want to know what capabilities are available for the chosen UART port before configuring the device parameters. Here is the list:

Item	Description
uart	supports UART (Asynchronous) mode
sir	supports serial Irda mode
fir	supports fast Irda mode

flow_control_rts	RTS Flow Control available
flow_control_cts	CTS Flow Control available
event_tx_complete	Transmit completed event: ARM_USART_EVENT_TX_COMPLETE
event_rx_timeout	Signal receive character timeout event: ARM_USART_EVENT_RX_TIMEOUT
rts	RTS Line: 0=not available, 1=available
cts	CTS Line: 0=not available, 1=available
dtr	DTR Line: 0=not available, 1=available
dsr	DSR Line: 0=not available, 1=available
dcd	DCD Line: 0=not available, 1=available
ri	RI Line: 0=not available, 1=available
event_cts	Signal CTS change event: ARM_USART_EVENT_CTS
event_dsr	Signal DSR change event: ARM_USART_EVENT_DSR
event_dcd	Signal DCD change event: ARM_USART_EVENT_DCD
event_ri	Signal RI change event: ARM_USART_EVENT_RI
reserved	Reserved (must be zero)

It includes: UART/SIR/FIR work modes, RTS/CTS hardware flow controls, hardware signal events, etc.

Kneron UART does not support all them, the client needs to query the detail via this API if necessary.

2.2.4 Configure & control

Prototype:
int32_t kdp_uart_control(kdp_uart_hdl_t handle, kdp_uart_ctrl_t prop, uint8_t * val)

This API is used to configure the driver and control hardware signals:

A. Working parameters

Includes:

- baud rate
- data length
- stop bits
- parity

B. Hardware Signal Control

Allow the client to set/reset RTS/DTR signal.

C. Miscellaneous Controls

Includes controls:

- Enable Tx
- Enable Rx
- Abort Tx
- Abort Rx
- Tx FIFO configuration
- Rx FIFO configuration
- Loopback enable
- Tx timeout value
- Rx timeout value

2.2.5 Query status

Prototype:
UART_STATUS_t *kdp_uart_get_status(kdp_uart_hdl_t handle)

This API can return the status of the device, here are the status definitions:

Status	Description
tx_busy	Transmitter busy flag
rx_busy	Receiver busy flag
rx_overflow	Receive data overflow detected (cleared on start of next receive operation)
rx_break	Break detected on receive (cleared on start of next receive operation)
rx_framing_error	Framing error detected on receive (cleared on start of next receive operation
rx_parity_error	Parity error detected on receive (cleared on start of next receive operation

2.2.6 Write data

Prototype:
kdp_uart_api_sts_t kdp_uart_write(kdp_uart_hdl_t hdl, uint8_t *buf, uint32_t len)

The client calls this API to send data out to remote side.
Depending on the work mode, a little bit different behavior exists there.
In synchronous mode, the API call will not return until all data was sent out physically;
In asynchronous mode, the API call shall return immediately with UART_API_TX_BUSY. When all the buffer data is sent out, the client registered

callback function will be invoked. The client shall have a very thin code there to set flags/signals. The client thread shall be listening the signal after this API call.

2.2.7 Query sent data length

Prototype:
uint32_t kdp_uart_GetTxCount(kdp_uart_hdl_t handle)
The client can call this API to detect how many bytes are sent out already.

2.2.8 Read data

Prototype:
kdp_uart_api_sts_t kdp_uart_read(kdp_uart_hdl_t handle, uint8_t *buf, uint32_t len)

The client can call this API to receive UART data from remote side.
Depending on the work mode, a little bit different behavior exists there.
In synchronous mode, the API call will not return until all data was received physically.
In asynchronous mode, the API call shall return immediately with UART_API_RX_BUSY. When enough bytes are received or timeout occurs, the client registered callback function will be invoked. The client shall have a very thin code there to set flags/signals. The client thread shall be listening the signal after this API call.

The client shall allocate the receiving buffer with max possible receiving length. When one frame is sent out, after 4 chars transmission time, a timeout interrupt will be generated.

2.2.9 Query received data length

Prototype:
uint32_t kdp_uart_GetRxCount(kdp_uart_hdl_t handle)
The client can call this API to detect how many bytes received already.
This API is helpful especially when Rx timeout event is received.

2.2.10 Power Control

Prototype:
int32_t kdp_uart_power_control(kdp_uart_hdl_t handle, ARM_POWER_STATE pwr_st)

This API provides a measure to switch on/off the individual UART port. This function shall be called after device open and before hardware configuration.

2.2.11 Close driver

Prototype:

int32_t kdp_uart_close(kdp_uart_hdl_t handle)

The API call will release all UART related resources: disable interrupts, release memories, disable hardware signals, etc.

3 Calling Sequence

Here depict the calling sequences for Async mode UART Tx/Rx: