MT793X IoT SDK for UART

User Guide

Version: 1.0

Release date: 2021-08-02

Use of this document and any information contained therein is subject to the terms and conditions set forth in Exhibit 1. This document is subject to change without notice.

Version History

Version	Date	Description
1.0	2021-08-02	Official release

Table of Contents

1 UART

1.1 Introduction

The Universal Asynchronous Receiver/Transmitter (UART) provides full duplex serial communication channels between the chip and external devices. The UART has M16C450 and M16550A operation modes, which are compatible with a range of standard software drivers. The extensions are designed to be broadly software compatible with 16550A variants, but certain areas offer no consensus.

In common with M16550A, the UART supports word lengths from 5 to 8 bits, an optional parity bit and one or two stop bits, and this word length is fully programmable by a CPU interface. A 16-bit programmable baud rate generator and an 8-bit scratch register are included, along with separate transmission and received FIFOs. Eight modem control lines and a diagnostic loop-back mode are provided. The UART also includes two Direct Memory Access (DMA) handshake lines which are used to indicate when the FIFOs are ready to transfer data to CPU. Interrupts can be generated from any of the several sources.

After hardware reset, the UART is in M16C450 mode. Its FIFOs can be enabled and the UART can enter M16550A mode. The UART adds further functionality beyond M16550A mode. Each of the extended functions can be selected individually under software control.

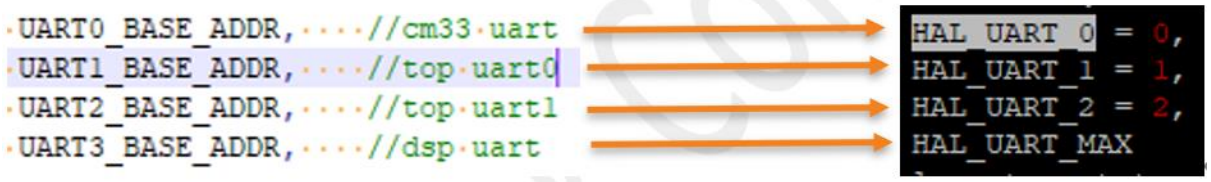
1.2 Features

The main supported features of UART are as follows:

- Provide 4 channels of UARTs
- TOP_UART0 and TOP_UART1 are used for user channels.
- CM33_UART and DSP_UART are used for print log..
- Support both M16C450 and M16550A operation modes
- Compatible with standard software drivers
- Transfer system: Asynchronous
- Data length: 5 to 8 bits
- Hardware flow control: CTS/RTS-based automatic transmission and reception of control
- Software flow control: Use special character Xon/Xoff to do software flow control
- Baud rate is programmable from 300 bps to 3 Mbps.
- Baud rate error: Less than 0.25 %
- Interrupt request: Receive interrupts/transmit interrupts
- Data transfer: DMA (Transmit/Receive) transfer is supported.

Module	HW Flow Control (CTS/RTS)	DMA mode
TOP_UART0	V	V
TOP_UART1	V	V
CM33_UART	V	N
DSP_UART	N	N

The mapping relationship between uart controllers and hal uart numbers are as follows:



2 Driver Introduction

2.1 Driver API Reference

hal_uart_status_t	**hal_uart_init** (**hal_uart_port_t** uart_port, **hal_uart_config_t** *uart_config) This function initializes the UART hardware with basic functionality. More...
hal_uart_status_t	**hal_uart_deinit** (**hal_uart_port_t** uart_port) This function deinitializes the UART hardware to its default status. More...
void	**hal_uart_put_char** (**hal_uart_port_t** uart_port, char byte) This function places one character at a time to the UART port in a polling mode. More...
char	**hal_uart_get_char** (**hal_uart_port_t** uart_port) This function gets one character from UART port in a polling mode. More...
uint32_t	**hal_uart_get_char_unblocking** (**hal_uart_port_t** uart_port) This function gets one character from UART port in the unblocking polling mode. More...
uint32_t	**hal_uart_send_polling** (**hal_uart_port_t** uart_port, const uint8_t *data, uint32_t size) This function sends user data byte by byte in a polling mode. More...
uint32_t	**hal_uart_send_dma** (**hal_uart_port_t** uart_port, const uint8_t *data, uint32_t size) This function sends user data in a VFIFO DMA mode. More...
uint32_t	**hal_uart_receive_polling** (**hal_uart_port_t** uart_port, uint8_t *buffer, uint32_t size) This function receives data byte by byte in a polling mode. More...
uint32_t	**hal_uart_receive_dma** (**hal_uart_port_t** uart_port, uint8_t *buffer, uint32_t size) This function receives user data in a VFIFO DMA mode. More...
uint32_t	**hal_uart_get_available_send_space** (**hal_uart_port_t** uart_port) This function queries available space in the VFIFO TX buffer. More...
uint32_t	**hal_uart_get_available_receive_bytes** (**hal_uart_port_t** uart_port) This function queries available data in the VFIFO RX buffer. More...
hal_uart_status_t	**hal_uart_register_callback** (**hal_uart_port_t** uart_port, **hal_uart_callback_t** user_callback, void *user_data) This function registers user's callback in the UART driver. More...
hal_uart_status_t	**hal_uart_set_hardware_flowcontrol** (**hal_uart_port_t** uart_port) This function sets and enables hardware flow control of the UART. More...
hal_uart_status_t	**hal_uart_set_software_flowcontrol** (**hal_uart_port_t** uart_port, uint8_t xon, uint8_t xoff, uint8_t escape_character) This function sets and enables software flow control of the UART. More...
hal_uart_status_t	**hal_uart_disable_flowcontrol** (**hal_uart_port_t** uart_port) This function disables flow control of the UART. More...
hal_uart_status_t	**hal_uart_disable_flowcontrol** (**hal_uart_port_t** uart_port) This function disables flow control of the UART. More...
hal_uart_status_t	**hal_uart_set_baudrate** (**hal_uart_port_t** uart_port, **hal_uart_baudrate_t** baudrate) This function sets a baud rate for the UART frame. More...
hal_uart_status_t	**hal_uart_set_format** (**hal_uart_port_t** uart_port, const **hal_uart_config_t** *config) This function sets the UART's frame parameter. More...
hal_uart_status_t	**hal_uart_set_dma** (**hal_uart_port_t** uart_port, const **hal_uart_dma_config_t** *dma_config) This function sets the VFIFO DMA hardware relative to the UART. More...
hal_uart_status_t	**hal_uart_set_auto_baudrate** (**hal_uart_port_t** uart_port, bool is_enable) This function sets auto baud rate for the specific UART port. More...

1.1. Sample Code

2.1.1 Using UART Polling Mode

- Step 1. Call **hal_uart_init()** to configure the UART in polling mode.
- Step 2. Call **hal_uart_get_char()** to get a character from the UART.
- Step 3. Call **hal_uart_put_char()** to put a character into the UART.
- Step 4. Call **hal_uart_receive_polling()** to get data from the UART with a user requested data length.
- Step 5. Call **hal_uart_send_polling()** to put data to the UART with a user requested data length.
- Step 6. Call **hal_uart_deinit()** to de-initialize the UART port.
- sample code:

```
void uart_poll_application(void)
{
    hal_uart_config_t uart_config;
    char data;
    char buffer[10];
    uart_config.baudrate = HAL_UART_BAUDRATE_921600;
    uart_config.parity = HAL_UART_PARITY_NONE;
    uart_config.stop_bit = HAL_UART_STOP_BIT_1;
    uart_config.word_length = HAL_UART_WORD_LENGTH_8;

    hal_uart_init(HAL_UART_0, &uart_config);
    data = hal_uart_get_char(HAL_UART_0);
    hal_uart_put_char(HAL_UART_0, data);
    hal_uart_receive_polling(HAL_UART_0, buffer, 10);
    hal_uart_send_polling(HAL_UART_0, buffer, 10);
    hal_uart_deinit(HAL_UART_0);
}
```

2.1.2 Using UART DMA Mode

- Step 1. Call **hal_uart_init()** to configure the UART in DMA mode.
- Step 2. Call **hal_uart_set_dma()** to configure the VFIFO DMA hardware. Note: the address should be non_cacheable, for more information, please refer to GDMA part.
- Step 3. Call **hal_uart_register_callback()** to register a callback function.
- Step 4. Call **hal_uart_receive_dma()** to receive data from the UART.
- Step 5. Call **hal_uart_send_dma()** to transmit data to the UART.
- Step 6. Call **hal_uart_deinit()** to de-initialize the UART port.
- sample code:

```
static volatile uint32_t receive_notice = 0;
static volatile uint32_t send_notice = 0;
static char rx_vfifo_buffer[512] __attribute__ ((section(".noncached_zidata")));
static char tx_vfifo_buffer[512] __attribute__ ((section(".noncached_zidata")));

static void uart_dma_application(void)
{
    hal_uart_config_t uart_config;
    char buffer[64];
    char *pbuf;
    uint32_t left, snd_cnt, rcv_cnt;
    hal_uart_dma_config_t dma_config;
    char uart_prompt[] = "UART DMA mode begin\n";

    uart_config.baudrate = HAL_UART_BAUDRATE_921600;
    uart_config.parity = HAL_UART_PARITY_NONE;
    uart_config.stop_bit = HAL_UART_STOP_BIT_1;
    uart_config.word_length = HAL_UART_WORD_LENGTH_8;
    hal_uart_init(HAL_UART_0, &uart_config);

    dma_config.receive_vfifo_alert_size = 50;
    dma_config.receive_vfifo_buffer = rx_vfifo_buffer;
    dma_config.receive_vfifo_buffer_size = 512;
    dma_config.receive_vfifo_threshold_size = 128;
    dma_config.send_vfifo_buffer = tx_vfifo_buffer;
    dma_config.send_vfifo_buffer_size = 512;
    dma_config.send_vfifo_threshold_size = 51;
    hal_uart_set_dma(HAL_UART_0, &dma_config);
    hal_uart_register_callback(HAL_UART_0, user_uart_callback, NULL);

    snd_cnt = hal_uart_send_dma(HAL_UART_0, uart_prompt, sizeof(uart_prompt));

    left = 64;
    pbuf = buffer;

    while(1){
        //Note: In this sample code, while(1) loop is used to wait for the interrupt,
        //but user can do anything before the interrupt is triggered.

        rcv_cnt = hal_uart_receive_dma(HAL_UART_0, pbuf, left);
        left -= rcv_cnt;
        pbuf += rcv_cnt;
        if(left == 0)
            break;

        while(!receive_notice);
        receive_notice = 0;
    }

    left = 64;
    pbuf = buffer;
    while(1){
        //Note: In this sample code, while(1) loop is used to wait for the interrupt,
        //but user can do anything before the interrupt is triggered.

        snd_cnt = hal_uart_send_dma(HAL_UART_0, pbuf, left);
        left -= snd_cnt;
        pbuf += snd_cnt;
        if(left == 0)
            break;

        while(!send_notice);
        send_notice = 0;
    }

    hal_uart_deinit(HAL_UART_0);
}
```

```
//Callback function. This function should be registered with #hal_uart_register_callback().
static void user_uart_callback(hal_uart_callback_event_t status, void *user_data)
{
    if(status == HAL_UART_EVENT_READY_TO_WRITE)
        send_notice = 1;
    else if(status == HAL_UART_EVENT_READY_TO_READ)
        receive_notice = 1;
}
```

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.