AN0099
Application Note

AT32 MCU USART Application Note

Introduction

This application note mainly introduces AT32 MCU USART and BSP demo program design, and demonstrates the usage and test result.

Note: The corresponding code in this application note is developed on the basis of V2.x.x BSP provided by Artery. For other versions of BSP, please pay attention to the differences in usage.

Applicable products:

Part number	AT32Fxx

Contents

1 USART introduction 7

2 USART functions 9

2.1 USART pins 9

2.2 Baud rate configuration 10

2.3 Frame format and configuration 10

2.4 Mode selector 11

2.4.1 Two-wire unidirectional full-duplex mode 11

2.4.2 LIN mode 11

2.4.3 Smartcard mode 12

2.4.4 Infrared mode 12

2.4.5 Hardware flow control mode 13

2.4.6 Silent mode 13

2.4.7 Synchronous mode 13

2.4.8 RS485 mode 13

2.5 Interrupts 13

2.6 DMA 14

2.6.1 Transmission using DMA 14

2.6.2 Reception using DMA 14

2.7 Transmitter/Receiver 15

2.7.1 Transmitter 15

2.7.2 Receiver 16

2.8 Start bit and noise detection 17

2.8.1 Start bit detection 17

2.8.2 Noise detection 18

3 USART application 20

3.1 Example 1: Single-wire bidirectional half-duplex mode 20

3.1.1 Function overview 20

3.1.2 Resources 20

3.1.3 Software design 20

3.1.4 Test result 23

3.2 Example 2: Hardware flow control mode 24
3.2.1 Function overview 24
3.2.2 Resources 24
3.2.3 Software design 24
3.2.4 Test result 26
3.3 Example 3: Idle detection 27
3.3.1 Function overview 27
3.3.2 Resources 27
3.3.3 Software design 27
3.3.4 Test result 29
3.4 Example 4: Communication in interrupt mode 30
3.4.1 Function overview 30
3.4.2 Resources 30
3.4.3 Software design 30
3.4.4 Test result 34
3.5 Example 5: Infrared mode 35
3.5.1 Function overview 35
3.5.2 Resources 35
3.5.3 Software design 35
3.5.4 Test result 38
3.6 Example 6: Communication in polling mode 38
3.6.1 Function overview 38
3.6.2 Resources 38
3.6.3 Software design 38
3.6.4 Test result 42
3.7 Example 7: Serial port (printf) 42
3.7.1 Function overview 42
3.7.2 Resources 42
3.7.3 Software design 42
3.7.4 Test result 45
3.8 Example 8: Mute mode 46
3.8.1 Function overview 46
3.8.2 Resources 46
3.8.3 Software design 46
3.8.4 Test result 49

3.9 Example 9: Smartcard mode 50
3.9.1 Function overview 50
3.9.2 Resources 51
3.9.3 Software design 51
3.9.4 Test result 59
3.10 Example 10: synchronous mode 60
3.10.1 Function overview 60
3.10.2 Resources 61
3.10.3 Software design 61
3.10.4 Test result 65
3.11 Example 11: Transmission using DMA 66
3.11.1 Function overview 66
3.11.2 Resources 67
3.11.3 Software design 67
3.11.4 Test result 74
3.12 Example 12: Communication by DMA polling 74
3.12.1 Function overview 74
3.12.2 Resources 75
3.12.3 Software design 75
3.12.4 Test result 80
3.13 Example 13: Tx/Rx swap 80
3.13.1 Function overview 80
3.13.2 Resources 81
3.13.3 Software design 81
3.13.4 Test result 84
4 Revision history 85

List of tables

Table 1. USART interrupt request 14

Table 2. Data sampling over start bit and noise detection........ 18

Table 3. Data sampling over valid data and noise detection 19

Table 4. Smartcard pins........ 51

Table 5. Document revision history 85

List of figures

Figure 1. USART block diagram ... 9
Figure 2. Example of frame format ... 11
Figure 3. TDC/TDBE behavior when transmitting ... 16
Figure 4. Start bit detection ... 18
Figure 5. half_duplex waveform captured by LA ... 23
Figure 6. hw_flow_control waveform captured by LA ... 26
Figure 7. idle_detection waveform captured by LA ... 29
Figure 8. Interrupt waveform captured by LA ... 35
Figure 9. IrDA DATA (3/16)–normal mode ... 35
Figure 10. irda transmit waveform captured by LA ... 38
Figure 11. Polling waveform captured by LA ... 42
Figure 12. Serial port printed information ... 45
Figure 13. Mute mode using address mark detection ... 46
Figure 14. receiver_mute waveform captured by LA ... 50
Figure 15. Smartcard waveform captured by LA ... 60
Figure 16. USART synchronous mode connection ... 60
Figure 17. USART synchronous mode data clock timing (DBN=0) ... 61
Figure 18. synchronous waveform captured by LA ... 65
Figure 19. Transmission using DMA ... 66
Figure 20. Reception using DMA ... 66
Figure 21. transfer_by_dma_interrupt waveform captured by LA ... 74
Figure 22. transfer_by_dma_polling waveform captured by LA ... 80

1 USART introduction

The universal synchronous/asynchronous receiver/transmitter (USART) serves an interface for communication by means of various configurations and peripherals with different data formats. It supports asynchronous full-duplex and half-duplex as well as synchronous transfer.

With a programmable baud rate generator, USART supports width range of baud rates by setting the system frequency and frequency divider, which is also convenient for users to configure the required communication frequency.

In addition to standard NRZ asynchronous and synchronous receiver/transmitter communication protocols, USART also supports widely-used serial communication protocols such as LIN (Local Interconnection Network), IrDA (Infrared Data Association) SIRENDEC specification, Asynchronous SmartCard protocol defined in ISO7816-3 standard, and CTS/RTS (Clear To Send/Request To Send) hardware flow operation. It also allows multi-processor communication, and supports silent mode waken up by idle frames or ID matching to build up a USART network. Meanwhile, high-speed communication is possible by using DMA.

AT32Fxx series MCUs all support USART, with different available USART interfaces for different AT32 MCU series. For details, refer to the Reference Manual of the corresponding AT32 MCU series.

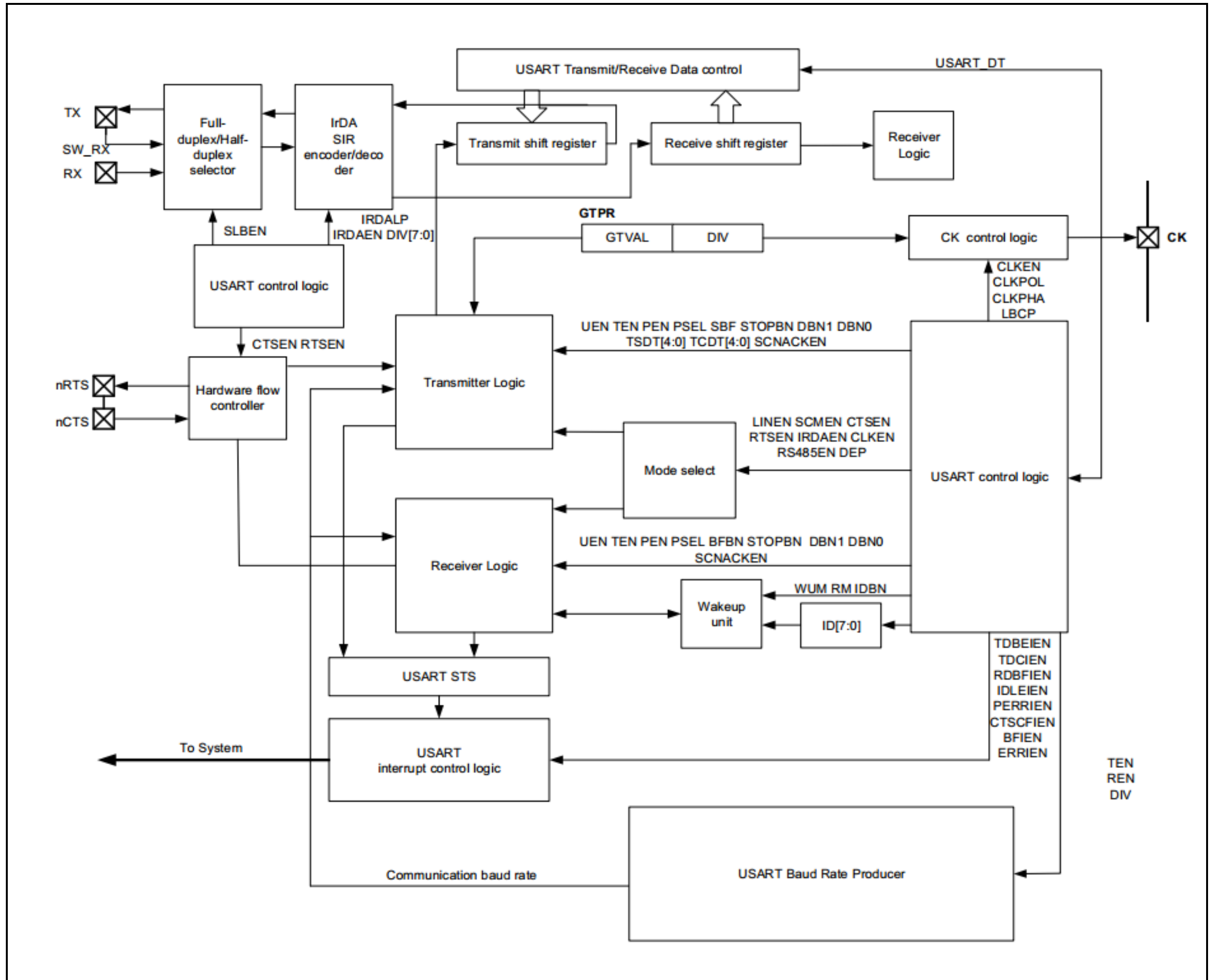
Main features:

- Programmable full-duplex or half-duplex communication
  - Full-duplex, asynchronous communication
  - Half-duplex, single communication
- Programmable communication modes
  - NRZ standard format (Mark/Space)
  - LIN (Local Interconnection Network): LIN master has break frame transmission capability and LIN slave has break frame detection capability.
  - IrDASIR (serial infrared): In normal mode, the transmitted pulse width is specified as 3/16 bit. In infrared low-power mode, the pulse width can be configurable.
  - Asynchronous Smartcard protocol defined in ISO7816-3: supports 0.5 or 1.5 stop bits
  - RS-232 CTS/RTS (Clear To Send/Request To Send) hardware flow operation
  - RS-485 (supported by AT32F435/437 series)
  - Multi-processor communication with silent mode (waken up by configuring ID match and bus idle frame)
  - Synchronous mode
- Programmable baud rate generator
  - Shared by transmission and reception
- Programmable frame format
  - Programmable data word length (8bits or 9 bits; AT32F435/437 series support 7 bits)
  - Programmable stop bits: support 1 or 2 stop bits
  - Programmable parity control: transmitter with parity bit transmission capability, and receiver with received data parity check capability
- Programmable DMA multi-processor communication
- Programmable separate enable bits for transmitter and receiver
- Programmable output CLK phase, polarity and frequency
- Detection flags
  - Receiver buffer full

- Transmit buffer empty
- Transfer complete flag
- Four error detection flags
  - Overrun error
  - Noise error
  - Frame error
  - Parity error
- Programmable 10 interrupt sources with flags
  - CTSF changes
  - LIN break detection
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle bus detected
  - Overrun error
  - Frame error
  - Noise error
  - Parity error
- Tx/Rx swap (supported by AT32F421/435/437 series)

2 USART functions

Figure 1. USART block diagram



2.1 USART pins

The USART bidirectional communication requires two pins, i.e., data input (RX) and data output (TX).

- RX: Serial data input. Use oversampling techniques to identify data and noise to recover data.
- TX: Serial data output. When the transmitter is disabled, the output pin returns to its I/O port configuration. Once the transmitter is activated and does not send data, the TX pin is high. In single-wire and Smartcard modes, the I/O port is used for data transmission and reception.

Pins required in synchronous mode:

- CK: Transmitter clock output. This pin is used for synchronous transfer clock (no clock pulse on the Start and Stop bits; software optionally sends a clock pulse at the last data bit). The data can be received on RX pin synchronously, which can be used to control external devices (such as LCD driver) designed with shift registers. The clock phase and polarity are programmable by software. In Smartcard mode, the CK pin can provides the Smartcard with clock.

Pins required in hardware flow control mode:

- CTS: Transmitter input. Send enable signal in hardware flow control mode. In case of low level, it indicates that the next data transfer continues at the end of the current data transfer. In case of high level, it indicates that the next data transfer is blocked at the end of the current data

transfer.

- RTS: Receiver output. Send request signal in hardware flow control mode. In case of low level, it indicates that USART is ready to receive data.

2.2 Baud rate configuration

USART baud rate generator uses an internal counter based on PCLK. The DIV (USART_BAUDR register) bit represents the overflow value of the counter. Each time the counter is full, it denotes one-bit data. Thus each data bit width refers to PCLK cycles x DIV.

User can program the desired baud rate by setting different system clocks and writing different values into the USART_BAUDR register. Write access to the USART_BAUDR register before UEN. The baud rate register value should not be altered when UEN=1. The calculation format is as follows:

$$\frac{\text{TX}}{\text{RX}}\text{baud rate} = \frac{f_{CK}}{DIV}$$

Where, f_{CK} refers to the USART system clock (i.e., PCLK1/PCLK2).

The receiver and transmitter of USART share the same baud rate generator, and the receiver splits each data bit into 16 equal parts to achieve oversampling, so the data bit width should not be less than 16 PCLK cycles, that is, the DIV value must be equal to or greater than 16.

Note: When USART receiver or transmitter is disabled, the internal counter will be reset, and baud rate interrupt will occur.

2.3 Frame format and configuration

Frame format:

- USART data frame consists of start bit, data bit and stop bit, with the last data bit being as a parity bit. The TX pin is low during start bit, and it is high during stop bit.
- USART idle frame size is equal to that of the data frame under current configuration, but all bits (including the stop bit) are 1.
- USART break frame size is the current data frame size plus its stop bit. All bits before the stop bit are 0.

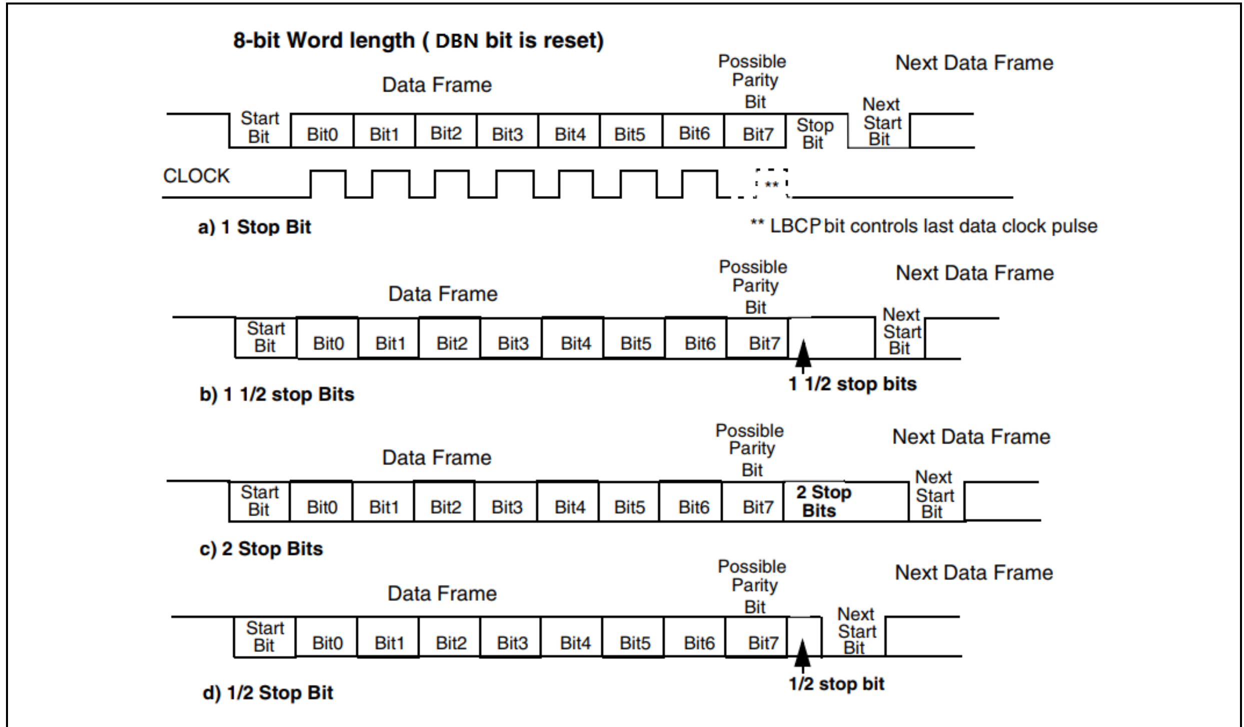
Configuration:

- Configure DBN=0 (8 data bits) or DBN=1 (9 data bits) by setting the bit [12] of the control register 1 (USART_CTRL1).

Note: For the AT32F435/437 series, set bit [28] (DBN1) to program 7 data bits. For details, refer to the corresponding Reference Manual.

- Configure STOPBN=00 (1 stop bit), STOPBN=01 (0.5 stop bit), STOPBN=10 (2 stop bits) or STOPBN=11 (1.5 stop bits) by setting the bit [13:12] of the control register 2 (USART_CTRL2).
- Configure PEN=1 by setting the bit [10] of the control register 1 (USART_CTRL1) to enable parity; select odd parity (PSEL=1) or even parity (PSEL=0) by setting the bit [9] (PSEL bit). When the PEN bit is enabled, the MSB bit of the transmitted data is replaced with the parity bit, that is, the valid data bit reduces one bit.

Figure 2. Example of frame format



2.4 Mode selector

USART mode selector allows USART to work in different operation modes through software configuration so as to enable data exchanges between USART and peripherals with different communication protocols.

2.4.1 Two-wire unidirectional full-duplex mode

USART supports NRZ standard format (Mark/Space). By default, in two-wire unidirectional full-duplex mode, the TX pin is used for data output, while RX pin is used for data input. The USART receiver and transmitter are independent, which allows USART to transmit and receive data synchronously, thus to realize full-duplex communication.

Set the SLBEN bit (bit[3]) in the USART_CTRL3 register to enable single-line bidirectional half-duplex communication. In this case, the LINEN, CLKEN, SCMEN and IRDAEN must be set to 0; the RX pin is inactive, and TX pin is connected to SW_RX pin. For the USART part, the TX pin is used for data output, while the SW_RX pin is used for data input. For the peripheral part, bidirectional data transfer is executed through I/O mapped by TX pin.

2.4.2 LIN mode

USART supports LIN mode.

Typical configuration:

- USART_CTRL2 register
  - bit14: LINEN=1
  - bit11: CLKEN=0
  - bit13:12: STOPBN[1:0]=0
- USART_CTRL3 register
  - bit5: SCMEN=0

- bit3: SLBEN=0
- bit1: IRDAEN=0

- Bit12 in the USART_CTRL1 register: DBN=0
- Bit5 in the USART_CTRL2 register: set BFBN to select 11-bit or 10-bit break frame
- Bit0 in the USART_CTRL1 register: set SBF to transmit 13-bit low LIN synchronous break frame

2.4.3 Smartcard mode

USART supports the asynchronous smartcard protocol defined in the ISO7816-3 standard.

Typical configuration:

- USART_CTRL2 register
  - bit14: LINEN=0
  - bit11: CLKEN=1
  - bit13:12: STOPBN[1:0]=11
- USART_CTRL3 register
  - bit5: SCMEN=1
  - bit3: SLBEN=0
  - bit1: IRDAEN=0
- USART_CTRL1 register
  - bit12: DBN=1
  - bit10: PEN=1
- Set clock polarity, phase and pulse number through the USART_CTRL2 register
  - bit10: CLKPOL
  - bit9: CLKPHA
  - bit8: LBCP
- Set the SCGT[7:0] bit in the USART_GDIV to select the guard time.
- Bit4 in the USART_CTRL3 register: set the SCNACKEN value to select whether to send NACK when parity error occurs.

2.4.4 Infrared mode

USART supports IrDA SIR (serial infrared) mode.

Typical configuration:

- USART_CTRL2 register
  - bit11: CLKEN=0
  - bit13:12: STOPBN[1:0]=0
- USART_CTRL3 register
  - bit1: IRDAEN=1
  - bit5: SCMEN=0
  - bit3: SLBEN=0
- Bit2 in the USART_CTRL3 register: set IRDALP=1 to enable IrDA low-power mode; meanwhile, set ISDIV[7:0] bit in the USART_GDIV to select the desired low-power frequency.

2.4.5 Hardware flow control mode

USART supports CTS/RTS (Clear To Send/Request To Send) hardware flow operation.

Set the RTSEN bit (bit3) and the CTSEN bit (bit9) in the USART_CTRL3 register to enable RTS and CTS flow control, respectively.

2.4.6 Silent mode

USART supports silent mode.

Typical configuration:

- USART_CTRL1 register
  - bit1: RM=1
  - bit11: set WUM=1 or WUM=0 to select wakeup by ID match or by idle line. The ID[3:0] is programmable (through the USART_CTRL2 register). When the wakeup by ID match is selected, MSB=1 of the data bit, indicating that the current data is ID, and the four LSB bits indicate the ID value.

2.4.7 Synchronous mode

USART supports synchronous mode.

Typical configuration:

- USART_CTRL2 register
  - bit11: CLKEN=1
  - bit10: CLKPOL=1 or CLKPOL=0 to select clock output high or low in idle state
  - bit9: CLKPHA=1 or CLKPHA=0 to select data capture on the second or the first clock edge
  - bit8: LBCP=1 or LBCP=0
- Set the ISDIV[7:0] bit in the USART_GDIV register to select the desired output clock frequency.

2.4.8 RS485 mode

The AT32F435/437 series USART also supports RS485 mode.

Set the RS485EN bit (bit14) in the USART_CTRL3 register to enable RS485 mode. The enable signal is output on the RTS pin. The DEP bit (bit15) is used to select the polarity of DE signal. The TSDT[4:0] bit is used to define the latency before the transmission of the start bit on the transmitter side, while the TCDT[4:0] is used to define the latency before the TC flag is set following the stop bit at the end of the last data.

2.5 Interrupts

USART interrupt generator serves as a control center of USART interrupts. It is used to monitor the interrupt source inside the USART in real time and the generation of interrupts according to the programmed interrupt control bits. Table 1 shows the USART interrupt source and interrupt enable control bit. An interrupt will be generated over an event when the corresponding interrupt enable bit is set.

Table 1. USART interrupt request

Interrupt event	Event flag	Enable bit
Transmit data register empty	TDBE	TDBEIEN
CTS flag	CTSCF	CTSCFIEN
Transmit data complete	TDC	TDCIEN
Receive data buffer full	RDBF	RDBFIEN
Receive overflow error	ROERR	
Idle flag	IDLEF	IDLEIEN
Parity error	PERR	PERRIEN
Break frame flag	BFF	BFIEN
Noise error, overflow error or framing error	NERR/ROERR/FERR	ERRIEN DMAREN

2.6 DMA

USART can use DMA to achieve continuous high-speed transmission by using DMA to enable the transmit data buffer and receive data buffer.

2.6.1 Transmission using DMA

1. Select a DMA channel: Select a DMA channel from DMA channel map table.
2. Configure the destination of DMA transfer: Configure the USART_DT register address as the destination address bit of DMA transfer in the DMA control register. Data will be sent to this address after transmit request is received by DMA.
3. Configure the source of DMA transfer: Configure the memory address as the source of DMA transfer in the DMA control register. Data will be loaded into the USART_DT register from the memory address after transmit request is received by DMA.
4. Configure the total number of bytes to be transferred in the DMA control register.
5. Configure the channel priority of DMA transfer in the DMA control register.
6. Configure DMA interrupt generation after half or full transfer in the DMA control register.
7. Enable DMA transfer channel in the DMA control register.

2.6.2 Reception using DMA

1. Select a DMA transfer channel: Select a DMA channel from DMA channel map table.
2. Configure the destination of DMA transfer: Configure the memory address as the destination of DMA transfer in the DMA control register. Data will be loaded from the USART_DT register to the programmed destination after reception request is received by DMA.
3. Configure the source of DMA transfer: Configure the USART_DT register address as the source of DMA transfer in the DMA control register. Data will be loaded from the USART_DT register to the programmed destination after reception request is received by DMA.
4. Configure the total number of bytes to be transferred in the DMA control register.
5. Configure the channel priority of DMA transfer in the DMA control register.
6. Configure DMA interrupt generation after half or full transfer in the DMA control register.

7. Enable a DMA transfer channel in the DMA control register

2.7 Transmitter/Receiver

2.7.1 Transmitter

USART transmitter has its individual TEN control bit. The transmitter and receiver share the same baud rate that is programmable. There is a transmit data buffer (TDR) and a transmit shift register in the USART. The TDBE bit is set whenever the TDR is empty, and an interrupt is generated if the TDBEIEN is set.

The data written by software is stored in the TDR register. When the shift register is empty, the data will be moved from the TDR register to the shift register so that the data in the transmit shift register is output on the TX pin in LSB mode. The output format depends on the programmed frame format.

If synchronous transfer or clock output is selected, the clock pulse is output on the CK pin. If the hardware flow control is selected, the control signal is input on the CTS pin.

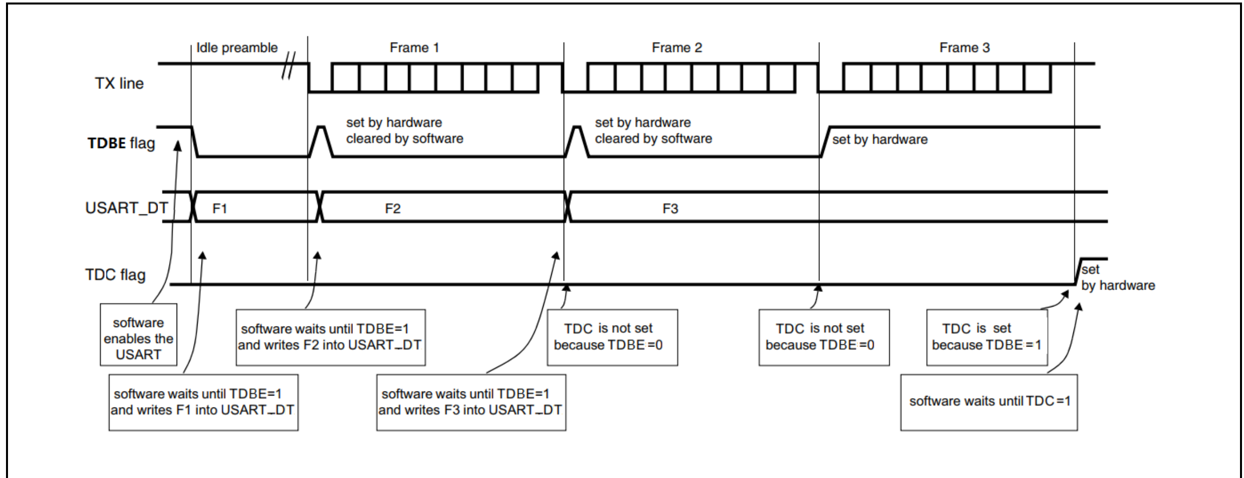
Note: 1. The TEN bit cannot be reset during data transfer; otherwise, the data on the TX pin will be corrupted.

2. After the TEN bit is enabled, the USART will automatically send an idle frame.

Transmitter configuration:

1. USART enable: set the UEN bit in the USART_CTRL1 register.
2. Full-duplex/half-duplex configuration: Refer to 2.4.1.
3. Mode configuration: Refer to 2.4.
4. Frame format configuration: Refer to 2.3.
5. Interrupt configuration: Refer to 2.5.
6. DMA transmission configuration: If the DMA mode is selected, the DMATEN bit (bit7 in the USART_CTRL3 register) is set, and configure DMA register according to 2.6.
7. Baud rate configuration: Refer to 2.2.
8. Transmitter enable: When the TEN bit is set, the USART transmitter will send an idle frame.
9. Write operation: Wait until the TDBE bit is set, the data to be transferred will be loaded into the USART_DT register (this operation will clear the TDBE bit). Repeat this step in non-DMA mode.
10. After the last data expected to be transferred is written, wait until the TDC is set, indicating the end of transfer. The USART cannot be disabled before the flag is set, or transfer error will occur.
11. When TDC=1, read access to the USART_STS register and write access to the USART_DT register will clear the TDC bit; This bit can also be cleared by writing “0”, but this is valid only in DMA mode.

Figure 3. TDC/TDBE behavior when transmitting



2.7.2 Receiver

USART receiver has its individual REN control bit. The transmitter and receiver share the same baud rate that is programmable. There is a receive data buffer (RDR) and a receive shift register in the USART.

The data is input on the RX pin of the USART. When a valid start bit is detected, the receiver ports the data received into the receive shift register in LSB mode. After a full data frame is received, based on the programmed frame format, it will be moved from the receive shift register to the receive data buffer, and the RDBF is set accordingly. An interrupt is generated if the RDBFIEN is set.

If hardware flow control is selected, the control signal is output on the RTS pin. During data reception, the USART receiver will detect whether there are errors to occur, including framing error, overrun error, parity check error or noise error, depending on software configuration, and whether there are interrupts to generate using the interrupt enable bits.

Receiver configuration:

1. USART enable: Set UEN=1 in the USART_CTRL1 register.
2. Full-duplex/half-duplex configuration: Refer to 2.4.1.
3. Mode configuration: Refer to 2.4.
4. Frame format configuration: Refer to 2.3.
5. Interrupt configuration: Refer to 2.5.
6. Reception using DMA: If the DMA mode is selected, the DMAREN bit is set, and configure DMA register according to 2.6.
7. Baud rate configuration: Refer to 2.2.
8. Receiver enable: Set REN=1.

Character reception:

- The RDBF bit is set. It indicates that the content of the shift register is transferred to the RDR (Receiver Data Register). In other words, data is received and can be read (including its associated error flags).
- An interrupt is generated when the RDBFIEN is set.
- The error flag is set when a framing error, noise error or overrun error is detected during

reception.

- In DMA mode, the RDNE bit is set after every byte is received, and it is cleared when the data register is read by DMA.
- In non-DMA mode, the RDBF bit is cleared during read access to the USART_DT register by software. The RDBF flag can also be cleared by writing 0 to it. The RDBF bit must be cleared before the end of next frame reception to avoid overrun error.

Break frame reception

- Non-LIN mode: It is handled as a framing error, and the FERR is set. An interrupt is generated if the corresponding interrupt bit is enabled.
- LIN mode: It is handled as a break frame, and the BFF bit is set. An interrupt is generated if the BFIEN is set.

Idle frame reception:

- It is handled as a data frame, and the IDLEF bit is set. An interrupt is generated if the IDLEIEN is set.

The data is transferred from shift register to the RDR register only when the RDBF bit is cleared. If the received data is not read in time, the RDBF bit is not set, and an overrun error occurs as soon as one character is received.

When an overrun error occurs:

- The ROERR bit is set.
- The data in the RDR is not lost. The previous data is still available when the USART_DT register is read.
- The content in the receive shift register is overwritten. Afterwards, any data received will be lost
- An interrupt is generated if the RDBFIEN is set or both ERRIEN and DMAREN are set
- The ROERR bit is cleared by reading the USART_STS register and then USART_DT register in order.

Note:

1. If ROERR is set, it indicates that at least one piece of data is lost.

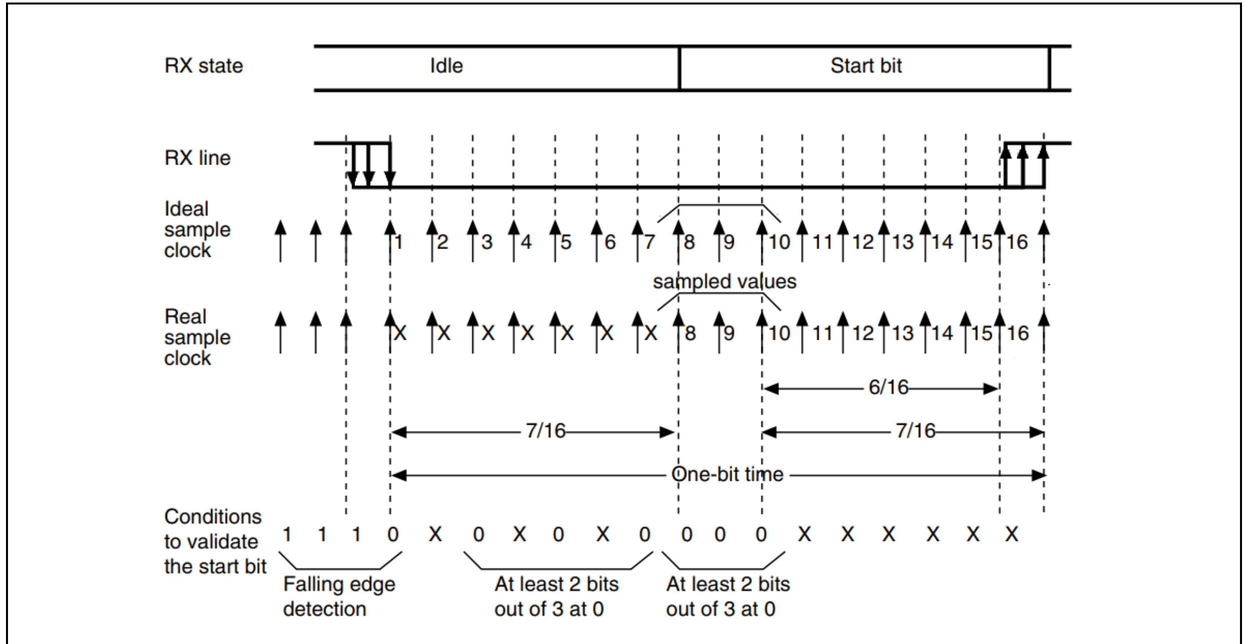
2. The REN bit cannot be reset during data reception, or the byte that is currently being received will be lost

2.8 Start bit and noise detection

2.8.1 Start bit detection

For the USART, a start bit is detected when a special sampling sequence is recognized. The sampling sequence is 1110X0X0X0000.

Figure 4. Start bit detection



A start bit detection occurs when the REN bit is set. With the oversampling techniques, the USART receiver samples data on the 3rd, 5th, 7th, 8th, 9th and 10th bits to detect the valid start bit and noise.

Table 2. Data sampling over start bit and noise detection

Sampled value (3-5-7)	Sampled value (8-9-10)	NERR bit	Start bit validity
000	000	0	Valid
001/010/100	001/010/100	1	Valid
001/010/100	000	1	Valid
000	001/010/100	1	Valid
111/110/101/011	Any value	1	Invalid
Any value	111/110/101/011	1	Invalid

If the sampling values on the 3rd, 5th, 7th, 8th, 9th, and 10th bits do not match the above mentioned requirements, the USART receiver does not think that a correct start bit is received, and thus it will abort the start bit detection and return to idle state waiting for a falling edge.

2.8.2 Noise detection

The USART receiver has the ability to detect noise. In the non-synchronous mode, the USART receiver samples data on the 7th, 8th and 9th bits, with its oversampling techniques, to distinguish valid data input from noise based on different sampling values, and recover data as well as set NERR (Noise Error Flag) bit.

Table 3. Data sampling over valid data and noise detection

Sampled value	NERR bit	Received bit value	Data validity
000	0	0	Valid
001	1	0	Invalid
010	1	0	Invalid
011	1	1	Invalid
100	1	0	Invalid
101	1	1	Invalid
110	1	1	Invalid
111	0	1	Valid

When noise is detected in a data frame:

- The NERR bit is set at the same time as the RDBF bit.
- The invalid data is transferred from the receive shift register to the receive data buffer.
- No interrupt is generated in non-DMA mode. However, since the NERR bit is set at the same time as the RDBF bit, the RDBF bit will generate an interrupt. In DMA mode, an interrupt will be issued if the ERRIEN is set.
- The NERR bit is cleared by read access to USART_STS register followed by the USART_DT read operation.

A framing error is detected when the stop bit is not received and recognized at the expected time due to unsynchronized data or a lot of noise.

When a framing error occurs:

- FERR bit is set.
- USART receiver moves the invalid data from the receive shift register to the receive data buffer.
- In non-DMA mode, both FERR and RDBF are set at the same time. The latter will generate an interrupt. In DMA mode, an interrupt is generated if the ERRIEN.
- The FERR is cleared through the read access to the USART_STS and USART_DT registers in sequence.

3 USART application

Note: All projects are based on Keil 5. If users want to use on other compiling environment, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, Keil 4/5) for simple changes.

3.1 Example 1: Single-wire bidirectional half-duplex mode

3.1.1 Function overview

The single-wire bidirectional half-duplex communication is enabled by setting SLBEN=1. In this case, RX is not used, and TX is released when no data is transmitted. Therefore, TX is used as a standard I/O port when it is in idle state or receive status, which means that this I/O port must be configured as floating input (or open-drain output high) when it is not driven by USART. Apart from that, this communication mode is similar to normal USART mode.

In this example, the USART2 and USART3 are configured as half-duplex mode. The main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on AT-START board will blink to indicate successful communication.

Use a Dupont cable to connect the USART2_TX (PA2) to USART3_TX (PB10), and then connect a 10kΩ pull-up resistor. If there is no available pull-up resistor and it is for communication verification purpose only, users can configure the USART3_TX (PB10) as push-pull for verification (not recommended in actual applications).

3.1.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\half_duplex\mdk_v5

3.1.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters;
   - Main function sends the receive data and compares it to the transmit data

2) Code

   - USART configuration code

```
/* configure USART clock, GPIO, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
```

```
    crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);
    gpio_default_para_init(&gpio_init_struct);
    /* configure the USART2_TX (PA2) pin as multiplexed open-drain pull-up */
    gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_OPEN_DRAIN;
    gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
    gpio_init_struct.gpio_pins = GPIO_PINS_2;
    gpio_init_struct.gpio_pull = GPIO_PULL_UP;
    gpio_init(GPIOA, &gpio_init_struct);
    /* configure the USART3_TX (PB10) pin as multiplexed open-drain pull-up */
    gpio_init_struct.gpio_pins = GPIO_PINS_10;
    gpio_init(GPIOB, &gpio_init_struct);
    /* configure the USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without
parity check */
    usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART2 transmitter and receiver, and configure as half-duplex mode */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    usart_single_line_halfduplex_select(USART2, TRUE);
    usart_enable(USART2, TRUE);
    /* configure the USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without
parity check */
    usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART3 transmitter and receiver, and configure as half-duplex mode */
    usart_transmitter_enable(USART3, TRUE);
    usart_receiver_enable(USART3, TRUE);
    usart_single_line_halfduplex_select(USART3, TRUE);
    usart_enable(USART3, TRUE);
  }
```

- Main function code

```
  /* define buffer */
  #define COUNTOF(a)                   (sizeof(a) / sizeof(*(a)))
  #define USART2_TX_BUFFER_SIZE          (COUNTOF(usart2_tx_buffer) - 1)
  #define USART3_TX_BUFFER_SIZE          (COUNTOF(usart3_tx_buffer) - 1)
  uint8_t usart2_tx_buffer[] = "usart half-duplex transfer: usart2 -> usart3";
  uint8_t usart3_tx_buffer[] = "usart half-duplex transfer: usart3 -> usart2";
  uint8_t usart2_rx_buffer[USART3_TX_BUFFER_SIZE];
  uint8_t usart3_rx_buffer[USART2_TX_BUFFER_SIZE];
```

```
uint8_t data_count;
/*buffer compare function*/
uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
{
  while(buffer_length--)
  {
    if(*pbuffer1 != *pbuffer2)
    {
      return 0;
    }
    pbuffer1++;
    pbuffer2++;
  }
  return 1;
}
/* main function */
int main(void)
{
  /* configure system clock and initialize AT-START board resources */
  system_clock_config();
  at32_board_init();
  /* configure USART clock, GPIO, baud rate and other initialization parameters */
  usart_configuration();

  /* configure USART2 transmitter and USART3 receiver */
  data_count = USART2_TX_BUFFER_SIZE;
  while(data_count)
  {
  /* wait until the TDBE bit of USART2 is set, and transmit data */
    while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
    usart_data_transmit(USART2, usart2_tx_buffer[USART2_TX_BUFFER_SIZE-data_count]);
  /* wait until the RDBF bit of USART3 is set, and receive data */
    while(usart_flag_get(USART3, USART_RDBF_FLAG) == RESET);
    usart3_rx_buffer[USART2_TX_BUFFER_SIZE-data_count] = usart_data_receive(USART3);
    data_count--;
  }

  /* configure USART3 transmitter and USART2 receiver */
```

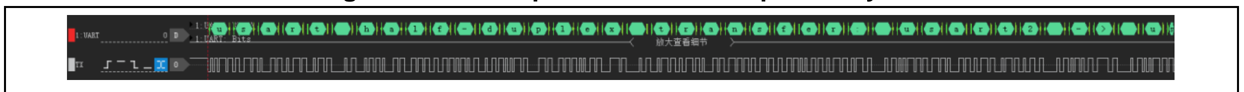
```
        data_count = USART3_TX_BUFFER_SIZE;
        while(data_count)
        {
        /* wait until the TDBE bit of USART3 is set, and transmit data */
          while(usart_flag_get(USART3, USART_TDBE_FLAG) == RESET);
          usart_data_transmit(USART3, usart3_tx_buffer[USART3_TX_BUFFER_SIZE-data_count]);
        /* wait until the RDBF bit of USART2 is set, and receive data */
          while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
          usart2_rx_buffer[USART3_TX_BUFFER_SIZE-data_count] = usart_data_receive(USART2);
          data_count--;
        }

        while(1)
        {
          /* compare the receive data buffer and transmit data buffer */
          if(buffer_compare(usart2_tx_buffer, usart3_rx_buffer, USART2_TX_BUFFER_SIZE) && \
             buffer_compare(usart3_tx_buffer, usart2_rx_buffer, USART3_TX_BUFFER_SIZE))
          {
            at32_led_toggle(LED2);
            at32_led_toggle(LED3);
            at32_led_toggle(LED4);
            delay_sec(1);
          }
        }
      }
```

3.1.4 Test result

The LED on AT-START board blinks, indicating that data is sent and received normally as expected. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 5. half_duplex waveform captured by LA



3.2 Example 2: Hardware flow control mode

3.2.1 Function overview

Set RTSEN=1 and CTSEN=1 to enable RTS and CTS flow control, respectively.

In this example, configure the USART2 as hardware flow control mode. Select RTS and CTS functions in the hyperterminal, and then send www.arterytek.com. The AT32 MCU receives the data from hyperterminal and then sends the received data to the hyperterminal. Meanwhile, three LEDs on AT-START BOARD blink, indicating end of data transmission.

Use serial tools (such as CH340) that supports RTS/CTS functions to connect the USART2 TX(PD5), RX (PD6), CTS(PD3) and RTS (PD4) to the hyperterminal.

3.2.2 Resources

1) Hardware

   AT32F403A AT-START BOARD, hyperterminal, serial tool

2) Software

   \project\at_start_f403a\examples\usart\hw_flow_control\mdk_v5

3.2.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters;
   - Receive data sent from the hyperterminal and then transmit the received data back.

2) Code

   - USART configuration code

```
/* configure USART clock, GPIOs, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
  gpio_default_para_init(&gpio_init_struct);
  /* configure USART2 TX(PD5) and RTS (PD4) pins as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_4 | GPIO_PINS_5;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOD, &gpio_init_struct);
  /* configure USART2 RX(PD6) and CTS(PD3) pins as pull-up input */
```

```
        gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
        gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
        gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
        gpio_init_struct.gpio_pins = GPIO_PINS_3 | GPIO_PINS_6;
        gpio_init_struct.gpio_pull = GPIO_PULL_UP;
        gpio_init(GPIOD, &gpio_init_struct);
        /* configure USART2 pin remap */
        gpio_pin_remap_config(USART2_GMUX_0001, TRUE);

        /*configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
        usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
        /* configure USART2 as hardware flow control mode */
        usart_hardware_flow_control_set(USART2, USART_HARDWARE_FLOW_RTS_CTS);
        /*enable USART2 transmitter and receiver */
        usart_transmitter_enable(USART2, TRUE);
        usart_receiver_enable(USART2, TRUE);
        usart_enable(USART2, TRUE);
    }
```

- Main function code

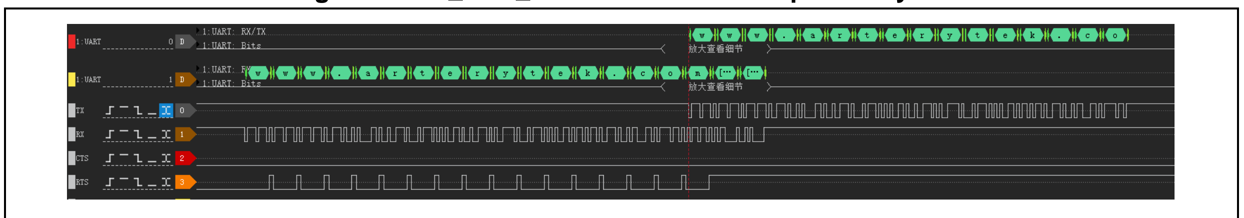
```
/* define buffer */
#define BUFFER_SIZE 16
uint8_t rx_buffer[BUFFER_SIZE];
uint8_t data_count;
/* main function */
int main(void)
{
  /* configure system clock and initialize AT-START board resources */
  system_clock_config();
  at32_board_init();
  /* configure USART clock, GPIO, baud rate and other initialization parameters */
  usart_configuration();
  /* receive a string (BUFFER_SIZE) from the hyperterminal */
  data_count = BUFFER_SIZE;
  while(data_count)
  {
    /* wait until the RDBF bit of USART2 is set, and receive data */
```

```
        while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
        rx_buffer[BUFFER_SIZE-data_count] = usart_data_receive(USART2);
        data_count--;
      }
      /* send back the received string (BUFFER_SIZE) to the hyperterminal */
      data_count = BUFFER_SIZE;
      while(data_count)
      {
        /* wait until the TDBE bit of USART2 is set, and transmit data */
        while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
        usart_data_transmit(USART2, rx_buffer[BUFFER_SIZE-data_count]);
        data_count--;
      }
      while(1)
      {
        at32_led_toggle(LED2);
        at32_led_toggle(LED3);
        at32_led_toggle(LED4);
        delay_sec(1);
      }
    }
```

3.2.4 Test result

The hyperterminal sends “www.arterytek.com” and then receives the data from AT32; meanwhile, the LED on AT-START board blinks, as expected. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 6. hw_flow_control waveform captured by LA



It can be found from the waveform that the hyperterminal only receives the first 16 characters sent back from AT32. The reason is that the BUFFER_SIZE is programmed as 16; therefore, the AT32 actually receives the first 16 characters from the hyperterminal and then send them back.

3.3 Example 3: Idle detection

3.3.1 Function overview

The IDLEF bit is set when an idle frame is received, and an interrupt is generated if the IDLEIEN bit is set.

In this example, use USART2 and set the IDLEIEN bit, and an interrupt occurs when an idle signal is detected. In the interrupt handler function, the three LEDs on AT-START board are ON, indicating that idle state is detected.

Use a Dupont cable to connect USART2_TX(PA2) and USART2_RX(PA3).

3.3.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\idle_detection\mdk_v5

3.3.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters;
   - Transmit data and wait for the idle state;
   - Interrupt handler function.

2) Code

   - USART configuration code

```
/* configure USART clock, GPIOs, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure the USART2 TX(PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure the USART2 RX(PA3) pin as pull-up input */
```

```
    gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
    gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
    gpio_init_struct.gpio_pins = GPIO_PINS_3;
    gpio_init_struct.gpio_pull = GPIO_PULL_UP;
    gpio_init(GPIOA, &gpio_init_struct);
    /* configure USART2 interrupt priority */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
    nvic_irq_enable(USART2_IRQn, 0, 0);
    /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART2 transmitter and receiver */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    /* enable USART2 idle detection interrupt */
    usart_interrupt_enable(USART2, USART_IDLE_INT, TRUE);
    usart_enable(USART2, TRUE);
  }
```

- Main function code

```
  /* main function */
  int main(void)
  {
    /* configure system clock and initialize resources on AT-START board */
    system_clock_config();
    at32_board_init();
    /* confirm on-board LEDs are OFF */
    at32_led_off(LED2);
    at32_led_off(LED3);
    at32_led_off(LED4);
    /* configure USART clock, GPIOs, baud rate and other initialization parameters */
    usart_configuration();
    /* TDBE reset value = 0x01*/
    while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
    /* send data and the data is then moved to the shift register, and the TDBE bit is kept at 1 */
    usart_data_transmit(USART2, 0x55);
    /* RDBF bit is set to 1 after data is received from TX pin */
```

```
    while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
    /* clear the RDBF flag */
    usart_data_receive(USART2);
    while(1)
    {
    }
}
```

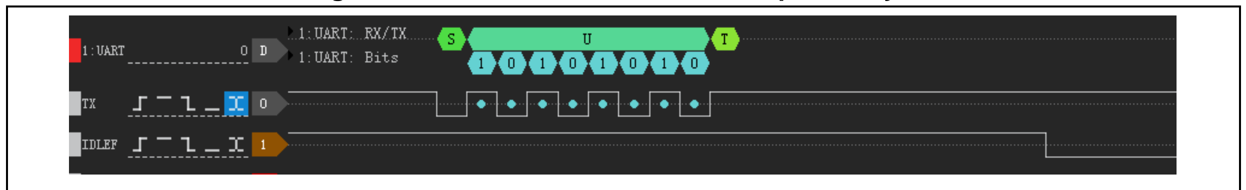
■ Interrupt handler function code

```
/* interrupt handler function */
void USART2_IRQHandler(void)
{
  if(usart_flag_get(USART2, USART_IDLEF_FLAG))
  {
    /* clear the RDBF flag */
    usart_data_receive(USART2);
    /* Turn on on-board LEDs to indicate IDLE detection interrupt */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
  }
}
```

3.3.4 Test result

LEDs on the AT-START board light up, as expected. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 7. idle_detection waveform captured by LA



It can be found from the waveform that AT32 sends the data (0x55) and detects an idle state (all bits are 1, and the length is the programmed data frame length), and then the IDLEF is set (obtained by capturing the LED pin).

3.4 Example 4: Communication in interrupt mode

3.4.1 Function overview

The TDBE bit is set when the transmit data register is empty, and an interrupt is generated if the TDBEIEN bit is set. The RDBF bit is set when the receive data buffer full, and an interrupt is generated if the RDBFIEN bit is set.

In this example, both USART2 and USART3 are configured as two-wire unidirectional full-duplex mode. In the interrupt handler function, the data transmission and reception, and interrupt flags are managed. The main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on AT-START board will blink to indicate successful communication.

Use a Dupont cable to connect USART2_TX(PA2) and USART3_RX(PB11), USART2_RX(PA3) and USART3_TX(PB10).

3.4.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\interrupt\mdk_v5

3.4.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters;
   - Compare the transmit and receive related parameters;
   - Transmit and receive data in interrupt handler function;

2) Code

   - USART configuration code

```
/* configure USART clock, GPIOs, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure USART2 TX(PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
```

```
    gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
    gpio_init_struct.gpio_pins = GPIO_PINS_2;
    gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
    gpio_init(GPIOA, &gpio_init_struct);
    /* configure USART3 TX(PB10) pin as multiplexed push-pull output */
    gpio_init_struct.gpio_pins = GPIO_PINS_10;
    gpio_init(GPIOB, &gpio_init_struct);

    /* configure USART2 RX(PA3) pin as pull-up input */
    gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
    gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
    gpio_init_struct.gpio_pins = GPIO_PINS_3;
    gpio_init_struct.gpio_pull = GPIO_PULL_UP;
    gpio_init(GPIOA, &gpio_init_struct);
    /* configure USART3 RX(PB11) pin as pull-up input */
    gpio_init_struct.gpio_pins = GPIO_PINS_11;
    gpio_init(GPIOB, &gpio_init_struct);
    /* configure USART interrupt priority */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
    nvic_irq_enable(USART2_IRQn, 0, 0);
    nvic_irq_enable(USART3_IRQn, 0, 0);
    /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART2 transmitter and receiver */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    /* configure USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART3 transmitter and receiver */
    usart_transmitter_enable(USART3, TRUE);
    usart_receiver_enable(USART3, TRUE);
    /* enable USART receive data buffer full interrupt, and enable USART */
    usart_interrupt_enable(USART2, USART_RDBF_INT, TRUE);
    usart_enable(USART2, TRUE);
    usart_interrupt_enable(USART3, USART_RDBF_INT, TRUE);
    usart_enable(USART3, TRUE);
```

```
    /* enable USART transmit data register empty interrupt */
    usart_interrupt_enable(USART2, USART_TDBE_INT, TRUE);
    usart_interrupt_enable(USART3, USART_TDBE_INT, TRUE);
}
```

- Main function code

```
/* define buffer */
#define COUNTOF(a)                          (sizeof(a) / sizeof(*(a)))
#define USART2_TX_BUFFER_SIZE                  (COUNTOF(usart2_tx_buffer) - 1)
#define USART3_TX_BUFFER_SIZE                  (COUNTOF(usart3_tx_buffer) - 1)
uint8_t usart2_tx_buffer[] = "usart transfer by interrupt: usart2 -> usart3 using interrupt";
uint8_t usart3_tx_buffer[] = "usart transfer by interrupt: usart3 -> usart2 using interrupt";
uint8_t usart2_rx_buffer[USART3_TX_BUFFER_SIZE];
uint8_t usart3_rx_buffer[USART2_TX_BUFFER_SIZE];
volatile uint8_t usart2_tx_counter = 0x00;
volatile uint8_t usart3_tx_counter = 0x00;
volatile uint8_t usart2_rx_counter = 0x00;
volatile uint8_t usart3_rx_counter = 0x00;
uint8_t usart2_tx_buffer_size = USART2_TX_BUFFER_SIZE;
uint8_t usart3_tx_buffer_size = USART3_TX_BUFFER_SIZE;
/*buffer compare function */
uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
{
  while(buffer_length--)
  {
    if(*pbuffer1 != *pbuffer2)
    {
      return 0;
    }
    pbuffer1++;
    pbuffer2++;
  }
  return 1;
}
/* main function */
int main(void)
{
  /* configure system clock and initialize resources on AT-START board */
```

```
    system_clock_config();
    at32_board_init();
    /* configure USART clock, GPIOs, baud rate and other initialization parameters */
    usart_configuration();
    /* wait until the end of data transfer from USART2_TX to USART3_RX */
    while(usart3_rx_counter < usart2_tx_buffer_size);
    /* wait until the end of data transfer from USART3_TX to USART2_RX */
    while(usart2_rx_counter < usart3_tx_buffer_size);
    while(1)
    {
      /* compare transmit and receive data buffers */
      if(buffer_compare(usart2_tx_buffer, usart3_rx_buffer, USART2_TX_BUFFER_SIZE) && \
         buffer_compare(usart3_tx_buffer, usart2_rx_buffer, USART3_TX_BUFFER_SIZE))
      {
        at32_led_toggle(LED2);
        at32_led_toggle(LED3);
        at32_led_toggle(LED4);
        delay_sec(1);
      }
    }
  }
```

- Interrupt handler function code

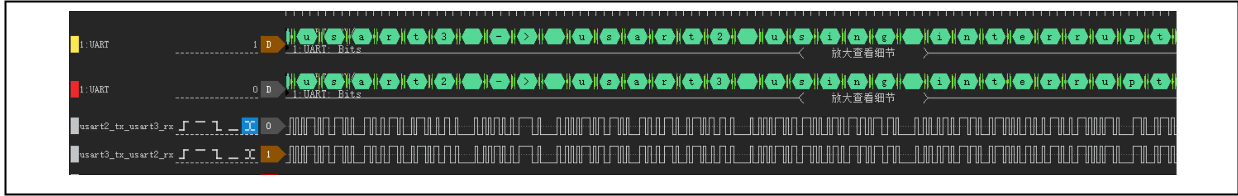
```
  /* interrupt handler function */
  void USART2_IRQHandler(void)
  {
    if(usart_flag_get(USART2, USART_RDBF_FLAG) != RESET)
    {
      /* confirm that USART2 RDBF bit is set, and receive data */
      usart2_rx_buffer[usart2_rx_counter++] = usart_data_receive(USART2);
      if(usart2_rx_counter == usart3_tx_buffer_size)
      {
        /* disable the interrupt enable bit after the transfer of programmed number of data is completed */
        usart_interrupt_enable(USART2, USART_RDBF_INT, FALSE);
      }
    }
    if(usart_flag_get(USART2, USART_TDBE_FLAG) != RESET)
    {
```

```
            /* confirm that USART2 TDBE bit is set, and send data */
            usart_data_transmit(USART2, usart2_tx_buffer[usart2_tx_counter++]);
            if(usart2_tx_counter == usart2_tx_buffer_size)
            {
                /* disable the interrupt enable bit after the transfer of programmed number of data is completed */
                usart_interrupt_enable(USART2, USART_TDBE_INT, FALSE);
            }
        }
    }
    void USART3_IRQHandler(void)
    {
        if(usart_flag_get(USART3, USART_RDBF_FLAG) != RESET)
        {
            /* confirm that USART3 RDBF bit is set, and receive data */
            usart3_rx_buffer[usart3_rx_counter++] = usart_data_receive(USART3);
            if(usart3_rx_counter == usart2_tx_buffer_size)
            {
                /* disable the interrupt enable bit after the transfer of programmed number of data is completed */
                usart_interrupt_enable(USART3, USART_RDBF_INT, FALSE);
            }
        }
        if(usart_flag_get(USART3, USART_TDBE_FLAG) != RESET)
        {
            /* confirm that USART3 TDBE bit is set, and send data */
            usart_data_transmit(USART3, usart3_tx_buffer[usart3_tx_counter++]);
            if(usart3_tx_counter == usart3_tx_buffer_size)
            {
                /* disable the interrupt enable bit after the transfer of programmed number of data is completed */
                usart_interrupt_enable(USART3, USART_TDBE_INT, FALSE);
            }
        }
    }
```

3.4.4 Test result

LEDs on the AT-START board light up, indicating that data is sent and received normally as expected. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 8. Interrupt waveform captured by LA



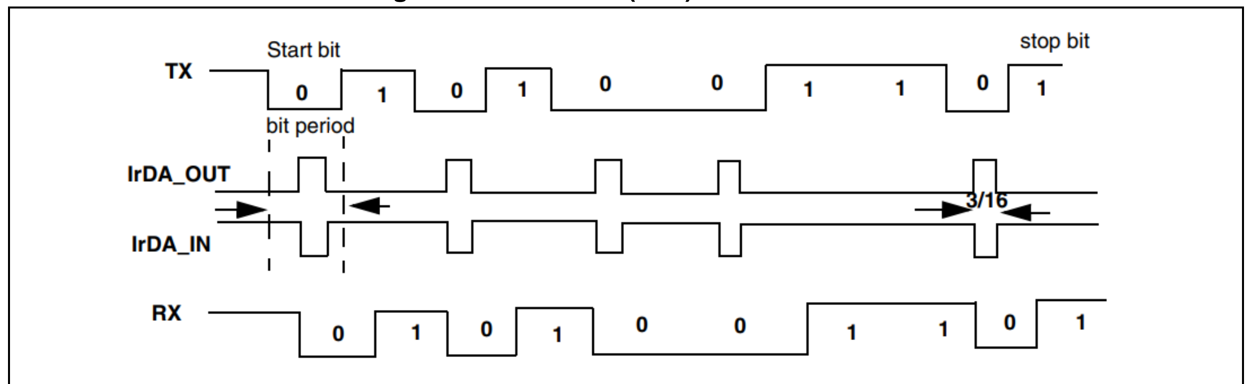
3.5 Example 5: Infrared mode

3.5.1 Function overview

The IrDA SIR physical layer specifies use of the Return-to-Zero-Inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse. The SIR transmit encoder modulates the Non-Return-to-Zero (NRZ) transmit bit stream output from the USART. The modulated output pulse stream is transmitted to an external output driver and infrared Light Emitting Diode (LED). The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to the USART. In idle state, the decoder input is high. A start bit is detected when the decoder input is low.

In normal mode, the transmitted pulse width is specified as 3/16 of a bit period. The transmit encoder output polarity is opposite to the decoder input polarity.

Figure 9. IrDA DATA (3/16)–normal mode



In this example, the USART2 is used to demonstrate IrDA basic communication functions. This project contains two targets (i.e., transmit and receive). In the transmit target, AT32 sends data (0x55) and then LED4 on the AT-START board lights up; in the receive target, AT32 receives data that is equal to 0x55, and then LED4 on the AT-START board lights up.

3.5.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\irda\mdk_v5

3.5.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters;

- Transmit/receive data (0x55).

2) Code

- USART configuration code

```
/* configure the USART clock, GPIOs, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure the USART2 TX (PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure the USART2 RX (PA3) pin as pull-up input */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
  gpio_init_struct.gpio_pins = GPIO_PINS_3;
  gpio_init_struct.gpio_pull = GPIO_PULL_UP;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
  usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
  /* enable USART2 transmitter and receiver */
  usart_transmitter_enable(USART2, TRUE);
  usart_receiver_enable(USART2, TRUE);
  /* configure infrared mode frequency division factor; in IrDA normal mode, the DIV must be 0x01 */
  usart_irda_smartcard_division_set(USART2, 0x01);
  /* enable infrared mode */
  usart_irda_mode_enable(USART2, TRUE);
  /* enable USART */
  usart_enable(USART2, TRUE);
```

```
}
```

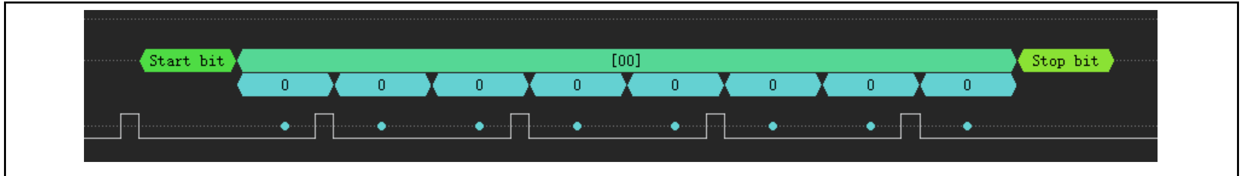
- Main function code

```
/* main function */
int main(void)
{
    /* configure system clock and initialize resources on AT-START board */
    system_clock_config();
    at32_board_init();
    /* confirm that on-board LEDs are OFF */
    at32_led_off(LED2);
    at32_led_off(LED3);
    at32_led_off(LED4);
    /* configure USART clock, GPIO, baud rate and other initialization parameters */
    usart_configuration();
/* defineTRANSMIT in Option for Target --> C/C++ --> Preprocessor Symbols */
#if defined(TRANSMIT)
    /* TDBE reset value=0x01*/
    while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
    /* transmit data */
    usart_data_transmit(USART2, 0x55);
    at32_led_on(LED4);
#endif
/* define RECEIVE in Option for Target --> C/C++ --> Preprocessor Symbols */
#if defined(RECEIVE)
    /* RDBF bit is set to 1 after the data is received via TX pin */
    while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
    /* LED4 lights up if the received data is equal to 0x55 */
    if(usart_data_receive(USART2) == 0x55)
    {
        at32_led_on(LED4);
    }
#endif
    while(1)
    {
    }
}
```

3.5.4 Test result

In the transmit target, AT32 sends data (0x55), and then LED4 on the AT-START board lights up. In the receive target, AT32 receives data that is equal to 0x55, and then LED4 on the AT-START board lights up. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 10. irda transmit waveform captured by LA



It can be found from the captured transmit waveform that AT32 sends data 0x55 (LA does not support IrDA SIR decoding).

Note that the polarity of transmit encoder output is opposite to that of decoder input. Therefore, an inverter is needed when two AT-START boards are connected for testing.

3.6 Example 6: Communication in polling mode

3.6.1 Function overview

The TDBE bit is set when the transmit data register is empty, and an interrupt is generated if the TDBEIEN bit is set. The RDBF bit is set when the receive data buffer full, and an interrupt is generated if the RDBFIEN bit is set.

In this example, both USART2 and USART3 are configured as two-wire unidirectional full-duplex mode. In the main function, check the corresponding flag and then perform data transmission and reception. The main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on AT-START board will blink to indicate successful communication.

Use a Dupont cable to connect USART2_TX(PA2) and USART3_RX(PB11), and USART2_RX (PA3) and USART3_TX(PB10).

3.6.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\polling\mdk_v5

3.6.3 Software design

1) Configuration process
   - Configure clocks, corresponding USART GPIOs and initialization parameters;
   - Main function transmits and receives data, and compare the transmit and receive data.
2) Code
   - USART configuration code

```
/* configure USART clock, GPIO, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure USART2 TX(PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure USART3 TX(PB10) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_pins = GPIO_PINS_10;
  gpio_init(GPIOB, &gpio_init_struct);

  /* configure USART2 RX(PA3) pin as pull-up input */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
  gpio_init_struct.gpio_pins = GPIO_PINS_3;
  gpio_init_struct.gpio_pull = GPIO_PULL_UP;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure USART3 RX(PB11) pin as pull-up input */
  gpio_init_struct.gpio_pins = GPIO_PINS_11;
  gpio_init(GPIOB, &gpio_init_struct);
  /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
  usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
  usart_parity_selection_config(USART2, USART_PARITY_NONE);
  /* enable USART2 transmitter and receiver */
```

```
        usart_transmitter_enable(USART2, TRUE);
        usart_receiver_enable(USART2, TRUE);
        /* enable USART */
        usart_enable(USART2, TRUE);
        /* configure USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
        usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
        usart_parity_selection_config(USART3, USART_PARITY_NONE);
        /* enable USART3 transmitter and receiver */
        usart_transmitter_enable(USART3, TRUE);
        usart_receiver_enable(USART3, TRUE);
        /* enable USART */
        usart_enable(USART3, TRUE);
    }
```

- Main function code

```
    /* define buffer */
    #define COUNTOF(a)                      (sizeof(a) / sizeof(*(a)))
    #define USART2_TX_BUFFER_SIZE              (COUNTOF(usart2_tx_buffer) - 1)
    #define USART3_TX_BUFFER_SIZE              (COUNTOF(usart3_tx_buffer) - 1)
    uint8_t usart2_tx_buffer[] = "usart transfer by polling: usart2 -> usart3 using polling ";
    uint8_t usart3_tx_buffer[] = "usart transfer by polling: usart3 -> usart2 using polling ";
    uint8_t usart2_rx_buffer[USART3_TX_BUFFER_SIZE];
    uint8_t usart3_rx_buffer[USART2_TX_BUFFER_SIZE];
    uint8_t usart2_tx_counter = 0x00;
    uint8_t usart3_tx_counter = 0x00;
    uint8_t usart2_rx_counter = 0x00;
    uint8_t usart3_rx_counter = 0x00;
    uint8_t usart2_tx_buffer_size = USART2_TX_BUFFER_SIZE;
    uint8_t usart3_tx_buffer_size = USART3_TX_BUFFER_SIZE;
    /*buffer compare function*/
    uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
    {
      while(buffer_length--)
      {
        if(*pbuffer1 != *pbuffer2)
        {
          return 0;
```

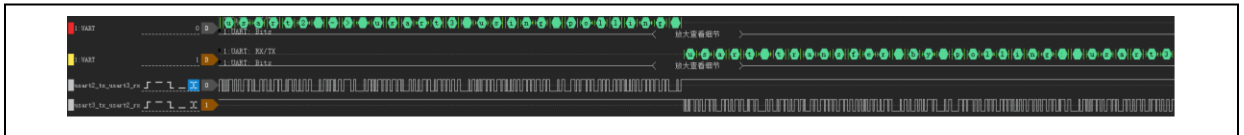
```
        }
        pbuffer1++;
        pbuffer2++;
    }
    return 1;
}
/* main function */
int main(void)
{
    /* configure system clock and initialize resources on AT-START board */
    system_clock_config();
    at32_board_init();
    /* configure USART clock, GPIO, baud rate and other initialization parameters */
    usart_configuration();
    /* wait until the end of data transfer from USART2_TX to USART3_RX */
    while(usart3_rx_counter < usart2_tx_buffer_size)
    {
        /* wait until the USART2 TDBE bit is set, and send data */
        while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
        usart_data_transmit(USART2, usart2_tx_buffer[usart2_tx_counter++]);
        /* wait until the USART3 RDBF bit is set, and receive data */
        while(usart_flag_get(USART3, USART_RDBF_FLAG) == RESET);
        usart3_rx_buffer[usart3_rx_counter++] = usart_data_receive(USART3);
    }
    /* wait until the end of data transfer from USART3_TX to USART2_RX */
    while(usart2_rx_counter < usart3_tx_buffer_size)
    {
        /* wait until the USART3 TDBE bit is set, and send data */
        while(usart_flag_get(USART3, USART_TDBE_FLAG) == RESET);
        usart_data_transmit(USART3, usart3_tx_buffer[usart3_tx_counter++]);
        /* wait until the USART2 RDBF bit is set, and receive data */
        while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
        usart2_rx_buffer[usart2_rx_counter++] = usart_data_receive(USART2);
    }
    while(1)
    {
        /* compare the receive and transmit data buffers */
        if(buffer_compare(usart2_tx_buffer, usart3_rx_buffer, USART2_TX_BUFFER_SIZE) && \
```

```
                buffer_compare(usart3_tx_buffer, usart2_rx_buffer, USART3_TX_BUFFER_SIZE))
            {
                at32_led_toggle(LED2);
                at32_led_toggle(LED3);
                at32_led_toggle(LED4);
                delay_sec(1);
            }
        }
    }
```

3.6.4 Test result

LEDs on the AT-START board blink, indicating that data is sent and received normally as expected. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis. In addition, users can compare the polling mode, interrupt mode and DMA mode.

Figure 11. Polling waveform captured by LA



3.7 Example 7: Serial port (printf)

3.7.1 Function overview

Redirect the printf function to the USART.

In this example, call the corresponding function in at32f4xx_board.c file (AT32 BSP) to initialize USART1, and add code segment supporting Printf function to the BSP to redirect Printf as serial port.

Connect the USART1_TX(PA9) with the hyperterminal via serial port tool. The on-board AT-LINK-EZ supports serial port function; therefore, it can be connected to the hyperterminal directly.

Note: Refer to AN0015_AT32_Printf_Debug_Demo for more details. This application note provides a complete set of example codes describing how to output the debug process information, especially when the serial port debugging assistant is not available.

3.7.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\printf\mdk_v5

3.7.3 Software design

1) Configuration process

   ■ Configure clocks, corresponding USART GPIOs and initialization parameters;

- Printf redirect and codes supporting Printf function;
- Main function printed information.

2) Code

- USART configuration code

```
/*************** define print uart *****************/
#define PRINT_UART                              USART1
#define PRINT_UART_CRM_CLK                      CRM_USART1_PERIPH_CLOCK
#define PRINT_UART_TX_PIN                       GPIO_PINS_9
#define PRINT_UART_TX_GPIO                      GPIOA
#define PRINT_UART_TX_GPIO_CRM_CLK              CRM_GPIOA_PERIPH_CLOCK
/* configure USART clock, GPIO, baud rate and other initialization parameters */
void uart_print_init(uint32_t baudrate)
{
  gpio_init_type gpio_init_struct;
  /* enable USART1 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(PRINT_UART_CRM_CLK, TRUE);
  crm_periph_clock_enable(PRINT_UART_TX_GPIO_CRM_CLK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure USART1 TX(PA9) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = PRINT_UART_TX_PIN;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(PRINT_UART_TX_GPIO, &gpio_init_struct);
  /* configure USART1 parameters, baud rate, 8-bit data bit, 1-bit stop bit, without parity check */
  usart_init(PRINT_UART, baudrate, USART_DATA_8BITS, USART_STOP_1_BIT);
  /* enable USART1 transmitter */
  usart_transmitter_enable(PRINT_UART, TRUE);
  /* enable USART */
  usart_enable(PRINT_UART, TRUE);
}
```

- Printf redirect and code supporting Printf function

```
/* retargets the c library printf function to the usart*/
PUTCHAR_PROTOTYPE
{
```

```
    while(usart_flag_get(PRINT_UART, USART_TDBE_FLAG) == RESET);
    usart_data_transmit(PRINT_UART, ch);
    return ch;
  }
  /* support printf function, usemicrolib is unnecessary */
  #if (__ARMCC_VERSION > 6000000)
    __asm (".global __use_no_semihosting\n\t");
    void _sys_exit(int x)
    {
      x = x;
    }
    /* __use_no_semihosting was requested, but _ttywrch was */
    void _ttywrch(int ch)
    {
      ch = ch;
    }
    FILE __stdout;
  #else
   #ifdef __CC_ARM
    #pragma import(__use_no_semihosting)
    struct __FILE
    {
      int handle;
    };
    FILE __stdout;
    void _sys_exit(int x)
    {
      x = x;
    }
    /* __use_no_semihosting was requested, but _ttywrch was */
    void _ttywrch(int ch)
    {
      ch = ch;
    }
   #endif
  #endif
  #if defined (__GNUC__) && !defined (__clang__)
    #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
```

```
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif
```

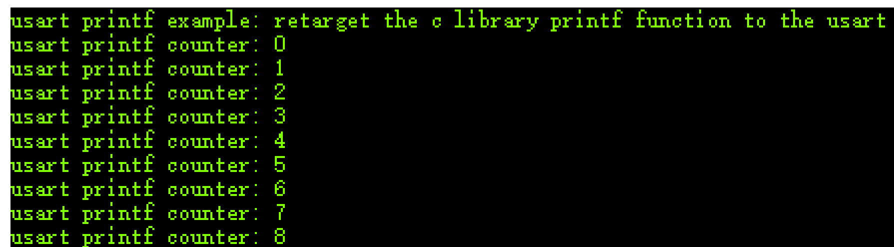
- Main function code

```
/* main function */
int main(void)
{
  /* configure system clock and initialize resources on AT-START board */
  system_clock_config();
  at32_board_init();
  /* configure USART clock, GPIO, baud rate and other initialization parameters */
  uart_print_init(115200);
  /* use printf function to print a string to the hyperterminal */
  printf("usart printf example: retarget the c library printf function to the usart\r\n");
  while(1)
  {
    printf("usart printf counter: %u\r\n",time_cnt++);
    delay_sec(1);
  }
}
```

3.7.4 Test result

The programmed array information is exactly printed in the hyperterminal, as expected.

Figure 12. Serial port printed information



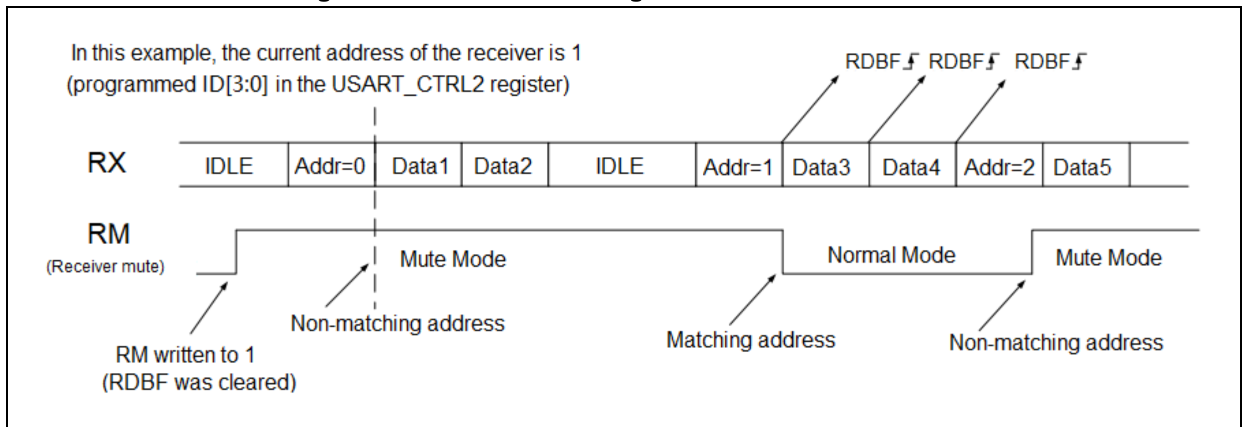
3.8 Example 8: Mute mode

3.8.1 Function overview

Mute mode can be entered by setting RM=1 in the USART_CTRL1 register. It is possible to wake up from mute mode by setting WUM=1 (ID match) and WUM=0 (idle bus), respectively. When address mismatches, the RM bit is set by hardware to enter mute mode again.

When ID match is selected, if the MSB of data bit is set, it indicates that the current data stands for ID. In one address byte, the target receiver address is stored in four LSB bits. This 4-bit address is compared with the receiver address. The ID[3:0] field in the USART_CTRL2 register holds the lower four bits of USART ID.

Figure 13. Mute mode using address mark detection



In this example, use USART3 and set the WUM bit to select mute mode wakeup by ID match. In the interrupt handler function, data received via RX pin from USART2 is stored in the array. Only the data after USART3 wakeup by ID match can be received. The main function executes parity check for the receive data. If there is no parity error, three LEDs on AT-START board will blink to indicate successful communication in mute mode.

Use a Dupont cable to connect USART2_TX(PA2) and USART3_RX(PB11).

3.8.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\receiver_mute\mdk_v5

3.8.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters (USART3 in mute mode);
   - USART2 sends address/data, and the main function executes parity check for USART3 receive data;
   - USART3 interrupt handler function receives data.

2) Code

- USART configuration code

```
/* configure USART clock, GPIOs, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure USART2 TX(PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure USART3 RX(PB11) pin as pull-up input */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
  gpio_init_struct.gpio_pins = GPIO_PINS_11;
  gpio_init_struct.gpio_pull = GPIO_PULL_UP;
  gpio_init(GPIOB, &gpio_init_struct);
  /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
  usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
  /* enable USART2 transmitter */
  usart_transmitter_enable(USART2, TRUE);
  usart_enable(USART2, TRUE);
  /* configure USART3 interrupt priority*/
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  nvic_irq_enable(USART3_IRQn, 0 ,0);
  /* configure USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
  usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
```

```
    /* enable USART3 receiver */
    usart_receiver_enable(USART3, TRUE);
    /* enable USART3 RDBF interrupt */
    usart_interrupt_enable(USART3, USART_RDBF_INT, TRUE);
    /* configure USART3 wakeup ID */
    usart_wakeup_id_set(USART3, 0x01);
    /* configure USART3 wakeup from mute mode by ID match */
    usart_wakeup_mode_set(USART3, USART_WAKEUP_BY_MATCHING_ID);
    /* enable USART3 mute mode */
    usart_receiver_mute_enable(USART3, TRUE);
    /* enable USART3 */
    usart_enable(USART3, TRUE);
}
```

- Main function code

```
/* define buffer */
#define BUFFER_SIZE 7
#define MATCH_ID_VAL  0x81
#define ERROR_ID_VAL 0x82
#define DATA1_VAL  0x7F
#define DATA2_VAL  0x7A
#define DATA3_VAL  0x7B
#define DATA4_VAL  0x7C
uint8_t usart2_tx_buffer[BUFFER_SIZE]={DATA1_VAL,MATCH_ID_VAL,DATA2_VAL,
ERROR_ID_VAL,DATA3_VAL,MATCH_ID_VAL,DATA4_VAL};
uint8_t usart3_rx_buffer[BUFFER_SIZE];
uint8_t tx_counter = 0;
uint8_t rx_counter = 0;
/* main function */
int main(void)
{
    /* configure system clock and initialize resources on AT-START board */
    system_clock_config();
    at32_board_init();
    /* configure USART clock, GPIOs, baud rate and other initialization parameters */
    usart_configuration();

    /* USART2 transmits address/data */
```

```
        while(tx_counter < BUFFER_SIZE)
        {
          while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
          usart_data_transmit(USART2, usart2_tx_buffer[tx_counter++]);
        }
        /* wait for the end of USART2 data transfer */
        while(usart_flag_get(USART2, USART_TDC_FLAG) == RESET);
        while(1)
        {
          /* execute parity check for the USART3 receive data; if it is equal to the transmit data, LED blinks */
          if((usart3_rx_buffer[0] == MATCH_ID_VAL) && (usart3_rx_buffer[1] == DATA2_VAL) && \
            (usart3_rx_buffer[2] == MATCH_ID_VAL) && (usart3_rx_buffer[3] == DATA4_VAL))
          {
            at32_led_toggle(LED2);
            at32_led_toggle(LED3);
            at32_led_toggle(LED4);
            delay_sec(1);
          }
        }
      }
```

- Interrupt handler function code

```
/* interrupt handler function */
void USART3_IRQHandler(void)
{
  /* confirm that USART3 RDBF bit is set; receive data and clear the flag */
  if(usart_flag_get(USART3, USART_RDBF_FLAG) != RESET)
  {
    usart3_rx_buffer[rx_counter++] = usart_data_receive(USART3);
  }
//  at32_led_toggle(LED3);
}
```

3.8.4 Test result

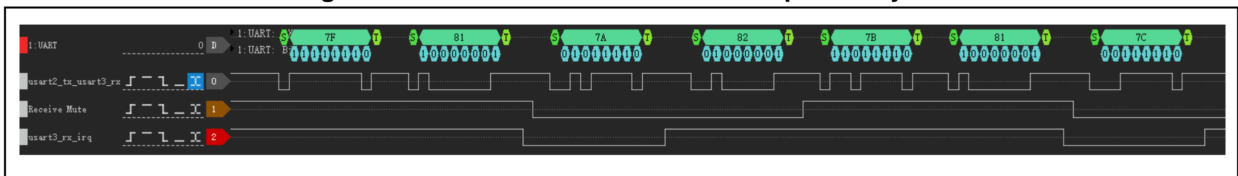
LEDs on the AT-START board blink, indicating normal communication. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

The below code is added to the main function to generate the Receive Mute waveform indicating the USART3 RM bit status. In this case, the LA is connected to PD13 corresponding to LED2. Besides, the code “at32_led_toggle(LED3)” is added to the USART3 interrupt handler function to generate

usart3_rx_irq waveform indicating the USART3 receive data. In this case, the LA is connected to PD14 corresponding to LED3.

```
/* usart2 transmit data */
while(tx_counter < BUFFER_SIZE)
{
    usart_data_receive(USART3);
    /* Indicate Receiver Mute */
    if(SET==USART3->ctrl1_bit.rm)
          at32_led_off(LED2);
    else
          at32_led_on(LED2);
   while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
   usart_data_transmit(USART2, usart2_tx_buffer[tx_counter++]);
   while(usart_flag_get(USART2, USART_TDC_FLAG) == RESET);
}
```

Figure 14. receiver_mute waveform captured by LA



The waveform shows that:

- USART2 transmit data (0x7F) is ignored by USART3;
- When the USART2 transmit address (0x81) matches the USART3 wakeup address ID (0x01), USART3 wakes up from mute mode and receives the data (0x7A);
- When the USART2 transmit address (0x82) does not match the USART3 wakeup address ID (0x01), USART3 enters mute mode and does not receive the subsequent transmit data (0x7B);
- When the address (0x81) transmitted from USART2 matches the USART3 wakeup address ID (0x01), USART3 wakes up from mute mode again and receives the data (0x7C).

3.9 Example 9: Smartcard mode

3.9.1 Function overview

The Smartcard is a single-wire half duplex communication protocol. The USART supports Smartcard asynchronous protocol and can be connected to the Smartcard that conforms to ISO7816-3 standard. The USART TX drives a bidirectional wire that is driven by Smartcard, and the TX should be configured as open-drain.

In this example, the USART2 is configured as Smartcard mode. The main function executes read operation, decoding and parity check. If there is no parity error, three LEDs on the AT-START board

blink, indicating successful communication.

The hardware connection is shown in Table 4.

Table 4. Smartcard pins

AT32	Smartcard	Function
PA2(sc_usart_tx)	I/O	MCU data I/O line (internal pull-up resistor connected to VDD)
PA4(sc_usart_clk)	CLK/CLKDIV	Clock to card
PA5(sc_usart_3_5v)	5V\3V	VCC selection pin
PA6(sc_usart_reset)	RST/RSTIN	Card reset (input from MCU)
PA7(sc_usart_cmdvcc)	\CMDVCC	Start activation sequence input
PA8(sc_usart_off)	\OFF	Interrupt to MCU

In this example, the USART2 USART_DIV_VAL value can be configurable, and the sc_clock_freq = PCLK/(2*USART_DIV_VAL) can be calculated according to the current PCLK clock, and conforms to the specification (within 1~5 MHz) defined in the ISO7816-3 standard. Then, calculate the initial baud rate (sc_baud_rate = sc_clock_freq/372) of communication with the Smartcard according to the sc_clock_freq and the F=372 and D=1 defined in the ISO7816-3.

3.9.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\smartcard\mdk_v5

3.9.3 Software design

1) Configuration process
   - Configure clocks, USART, GPIO and initialization parameters
   - Decode and verify
   - USART2 interrupt handler function receives data
2) Code
   - smartcard_config.h

```
/* smartcard interface usart pins */
#define SC_USART                 USART2
#define SC_USART_CLK             CRM_USART2_PERIPH_CLOCK
#define SC_USART_GPIO_CLK        CRM_GPIOA_PERIPH_CLOCK
#define SC_USART_TX_PIN          GPIO_PINS_2
#define SC_USART_TX_PORT         GPIOA
#define SC_USART_CLK_PIN         GPIO_PINS_4
#define SC_USART_CLK_PORT        GPIOA
```

```
#define SC_USART_IRQn                    USART2_IRQn
#define SC_USART_IRQHandler              USART2_IRQHandler
/* smartcard interface gpio pins */
#define SC_3_5V_PIN                      GPIO_PINS_5
#define SC_RESET_PIN                     GPIO_PINS_6
#define SC_CMDVCC_PIN                    GPIO_PINS_7
#define SC_OFF_PIN                       GPIO_PINS_8
#define SC_3_5V_PORT                     GPIOA
#define SC_RESET_PORT                    GPIOA
#define SC_CMDVCC_PORT                   GPIOA
#define SC_OFF_PORT                      GPIOA
#define SC_3_5V_CLK                      CRM_GPIOA_PERIPH_CLOCK
#define SC_RESET_CLK                     CRM_GPIOA_PERIPH_CLOCK
#define SC_CMDVCC_CLK                    CRM_GPIOA_PERIPH_CLOCK
#define SC_OFF_CLK                       CRM_GPIOA_PERIPH_CLOCK
#define SC_OFF_EXINT                     EXINT_LINE_8
#define SC_OFF_PORTSOURCE                GPIO_PORT_SOURCE_GPIOA
#define SC_OFF_PINSOURCE                 GPIO_PINS_SOURCE8
#define SC_OFF_EXINT_IRQ                 EXINT9_5_IRQn
```

■ Smartcard related configuration code

```
/* configure USART clock, GPIO, baud rate and initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  exint_init_type exint_init_struct;
  crm_clocks_freq_type crm_clocks_struct;

  /* enable Smartcard related GPIO peripheral clock and IOMUX clock */
  crm_periph_clock_enable(SC_3_5V_CLK, TRUE);
  crm_periph_clock_enable(SC_USART_GPIO_CLK, TRUE);
  crm_periph_clock_enable(SC_RESET_CLK, TRUE);
  crm_periph_clock_enable(SC_CMDVCC_CLK, TRUE);
  crm_periph_clock_enable(SC_OFF_CLK, TRUE);
  crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
  /* enable USART2 clock */
  crm_periph_clock_enable(SC_USART_CLK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
```

```
/* configure sc_usart_clk(PA4) pin as multiplexed push-pull output */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_pins = SC_USART_CLK_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init(SC_USART_CLK_PORT, &gpio_init_struct);
/* configure sc_usart_tx(PA2) pin as multiplexed open-drain output */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_OPEN_DRAIN;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_pins = SC_USART_TX_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init(SC_USART_TX_PORT, &gpio_init_struct);
/* configure sc_usart_reset(PA6) pin as push-pull output */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
gpio_init_struct.gpio_pins = SC_RESET_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init(SC_RESET_PORT, &gpio_init_struct);
/* configure sc_usart_reset(PA6) pin as output high */
gpio_bits_set(SC_RESET_PORT, SC_RESET_PIN);
/* configure sc_usart_3_5v(PA5) pin as push-pull output */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
gpio_init_struct.gpio_pins = SC_3_5V_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init(SC_3_5V_PORT, &gpio_init_struct);
/* configure sc_usart_3_5v(PA5) pin as output high, and select smartcard Vcc=5V */
gpio_bits_set(SC_3_5V_PORT, SC_3_5V_PIN);
/* configure sc_usart_cmdvcc(PA7) pin as push-pull output */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
gpio_init_struct.gpio_pins = SC_CMDVCC_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
```

```
gpio_init(SC_CMDVCC_PORT, &gpio_init_struct);
/* configure sc_usart_cmdvcc(PA7) pin as output high */
gpio_bits_set(SC_CMDVCC_PORT, SC_CMDVCC_PIN);
/* configure sc_usart_off(PA8) pin as input */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
gpio_init_struct.gpio_pins = SC_OFF_PIN;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init(SC_OFF_PORT, &gpio_init_struct);
/* configure USART2 interrupt priority */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
NVIC_ClearPendingIRQ(SC_OFF_EXINT_IRQ);
nvic_irq_enable(SC_OFF_EXINT_IRQ, 0, 0);
nvic_irq_enable(SC_USART_IRQn, 1, 0);
/* configure sc_usart_off interrupt */
gpio_exint_line_config(SC_OFF_PORTSOURCE, SC_OFF_PINSOURCE);
exint_default_para_init(&exint_init_struct);
exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
exint_init_struct.line_polarity= EXINT_TRIGGER_RISING_EDGE;
exint_init_struct.line_select = SC_OFF_EXINT;
exint_init_struct.line_enable = TRUE;
exint_init(&exint_init_struct);
/* clear sc_usart_off EXINT_LINE_8 external interrupt flag */
exint_flag_clear(SC_OFF_EXINT);
/* set sc_usart_clock =apbclk / (2 * SC_USART_DIV_VAL)  */
usart_irda_smartcard_division_set(SC_USART, SC_USART_DIV_VAL);
/* set smartcard guard time sc_usart =2 */
usart_smartcard_guard_time_set(SC_USART, 0x2);
/* configure USART2 clock polarity low, data capture on the first edge of clock phase, and the last bit
clock pulse output */
usart_clock_config(SC_USART,USART_CLOCK_POLARITY_LOW,
USART_CLOCK_PHASE_1EDGE, USART_CLOCK_LAST_BIT_OUTPUT);
/* enable USART2 clock output */
usart_clock_enable(SC_USART, TRUE);
crm_clocks_freq_get(&crm_clocks_struct);
/* calculate smartcard clock (1-5 MHz as defined in the ISO 7816-3 standard) */
sc_clock_freq = crm_clocks_struct.apb1_freq / (2 * SC_USART_DIV_VAL);
```

```
        /* calculate the baud rate for communication with smartcard */
        sc_baud_rate = sc_clock_freq / SC_F_DIV_D;
        /* configure USART2 parameters, and set sc_baud_rate, 9-bit data bit, 1.5-bit stop bit, with even parity check */
        usart_init(SC_USART, sc_baud_rate, USART_DATA_9BITS, USART_STOP_1_5_BIT);
        usart_parity_selection_config(SC_USART, USART_PARITY_EVEN);
        /* enable USART2 transmitter and receiver */
        usart_transmitter_enable(SC_USART, TRUE);
        usart_receiver_enable(SC_USART, TRUE);
        /* enable USART2 parity error interrupt */
        usart_interrupt_enable(SC_USART, USART_PERR_INT, TRUE);
        /* enable USART2 */
        usart_enable(SC_USART, TRUE);
        /* configure smartcard NACK enable bit, and send NACK in case of parity error */
        usart_smartcard_nack_set(SC_USART, TRUE);
        /* enable smartcard mode */
        usart_smartcard_mode_enable(SC_USART, TRUE);
    }
```

- Main function decode and parity check code

```
    /* atr structure - answer to reset */
    typedef struct
    {
        uint8_t ts;                 /* bit convention */
        uint8_t t0;                 /* high nibble = n. of setup byte; low nibble = n. of historical byte */
        uint8_t t[SETUP_LENGTH]; /* setup array */
        uint8_t h[HIST_LENGTH];   /* historical array */
        uint8_t tlength;            /* setup array dimension */
        uint8_t hlength;            /* historical array dimension */
    } sc_atr_type;

    sc_atr_type sc_a2r_struct;
    uint32_t sc_baud_rate = 0;
    uint32_t sc_clock_freq = 0;
    uint32_t counter = 0;
    error_status atr_decode_status = ERROR;
    uint32_t card_inserted = 0, card_protocol = 1;
    uint8_t dst_buffer[50]= {
```

```
                        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                        1, 1, 1, 1, 1, 1, 1, 1, 1, 1
                    };

/* smartcard decode */
uint8_t sc_response_decode(__IO uint8_t *card)
{
  uint32_t i = 0, flag = 0, buf = 0, protocol = 0;
  sc_a2r_struct.ts = 0;
  sc_a2r_struct.t0 = 0;
  for(i = 0; i < SETUP_LENGTH; i++)
  {
    sc_a2r_struct.t[i] = 0;
  }
  for(i = 0;i < HIST_LENGTH; i++)
  {
    sc_a2r_struct.h[i] = 0;
  }
  sc_a2r_struct.tlength = 0;
  sc_a2r_struct.hlength = 0;
  /* initial character */
  sc_a2r_struct.ts = card[0];
  /* format character */
  sc_a2r_struct.t0 = card[1];
  sc_a2r_struct.hlength = sc_a2r_struct.t0 & 0x0F;
  if((sc_a2r_struct.t0 & 0x80) == 0x80) flag = 1;
  for(i = 0; i < 4; i++)
  {
    sc_a2r_struct.tlength = sc_a2r_struct.tlength + (((sc_a2r_struct.t0 & 0xF0) >> (4 + i)) & 0x1);
  }
  for(i = 0; i < sc_a2r_struct.tlength; i++)
  {
    sc_a2r_struct.t[i] = card[i + 2];
  }
  protocol = sc_a2r_struct.t[sc_a2r_struct.tlength - 1] & 0x0F;
  while(flag)
  {
```

```
            if((sc_a2r_struct.t[sc_a2r_struct.tlength - 1] & 0x80)== 0x80)
            {
              flag = 1;
            }
            else
            {
              flag = 0;
            }
            buf = sc_a2r_struct.tlength;
            sc_a2r_struct.tlength = 0;
            for(i = 0; i < 4; i++)
            {
              sc_a2r_struct.tlength = sc_a2r_struct.tlength + (((sc_a2r_struct.t[buf - 1] & 0xF0) >> (4 + i)) & 0x1);
            }
            for(i = 0; i < sc_a2r_struct.tlength; i++)
            {
              sc_a2r_struct.t[buf + i] = card[i + 2 + buf];
            }
            sc_a2r_struct.tlength += buf;
          }
          for(i = 0;i < sc_a2r_struct.hlength; i++)
          {
            sc_a2r_struct.h[i] = card[i + 2 + sc_a2r_struct.tlength];
          }
          return ((uint8_t)protocol);
        }
        /* main function */
        int main(void)
        {
          uint32_t index = 0;
           /* configure system clock and initialize resources on AT-START board */
          system_clock_config();
          at32_board_init();
           /* configure USART clock, GPIOs, baud rate and other initialization parameters */
          usart_configuration();
           /* Loop and await when smartcard is not detected on EXINT  */
          while(card_inserted == 0)
          {
```

```
    }
     /* read smartcard ATR(Answer To Request) response */
    for(index = 0; index < 40; index++, counter = 0)
    {
        /* wait until the USART2 RDBF flag is cleared or timeout */
      while((usart_flag_get(SC_USART,USART_RDBF_FLAG)== RESET)
      && (counter != SC_RECEIVE_TIMEOUT))
      {
        counter++;
      }
        /* USART2 receive data and stores into buffer */
      if(counter != SC_RECEIVE_TIMEOUT)
      {
        dst_buffer[index] = usart_data_receive(SC_USART);
      }
    }
    /* decode ATR */
    card_protocol = sc_response_decode(dst_buffer);
    /* check whether the inserted card conforms to the ISO7816-3 standard */
    if(card_protocol == 0)
    {
      atr_decode_status = SUCCESS;
    }
    else
    {
      atr_decode_status = ERROR;
    }
    /* on-board LEDs light up if the card conforms to the ISO7816-3 standard */
    if(atr_decode_status == SUCCESS)
    {
      at32_led_on(LED2);
      at32_led_on(LED3);
      at32_led_on(LED4);
    }
    while(1)
    {
    }
  }
```

- Interrupt handler function code

```
/* interrupt handler function */
extern uint32_t card_inserted;
void EXINT9_5_IRQHandler(void)
{
  /* reset the smartcard sc_usart_cmdvcc pin */
  gpio_bits_reset(SC_CMDVCC_PORT, SC_CMDVCC_PIN);
  /* reset smartcard sc_usart_rst pin */
  gpio_bits_reset(SC_RESET_PORT, SC_RESET_PIN);
  /* set smartcard sc_usart_rst pin */
  gpio_bits_set(SC_RESET_PORT, SC_RESET_PIN);
  /* clear EXINT_LINE_8 flag */
  exint_flag_clear(SC_OFF_EXINT);
  /* set the card_inserted flag */
  card_inserted = 1;
}
void SC_USART_IRQHandler(void)
{
  /* USART2 detects parity error */
  if(usart_flag_get(SC_USART, USART_PERR_FLAG) != RESET)
  {
    /* enable USART2 RDBF interrupt (until the error data is received) */
    usart_interrupt_enable(SC_USART, USART_RDBF_INT, TRUE);
    /* read error data, clear receive data buffer and clear the flag */
    usart_data_receive(SC_USART);
  }
  if(usart_flag_get(SC_USART, USART_RDBF_FLAG) != RESET)
  {
     /* confirm that the USART2 RDBF bit is set; disable RDBF interrupt, receive data and clear the flag
*/
    usart_interrupt_enable(SC_USART, USART_RDBF_INT, FALSE);
    usart_data_receive(SC_USART);
  }
}
```

3.9.4 Test result

LEDs on the AT-START board light up, indicating normal communication. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

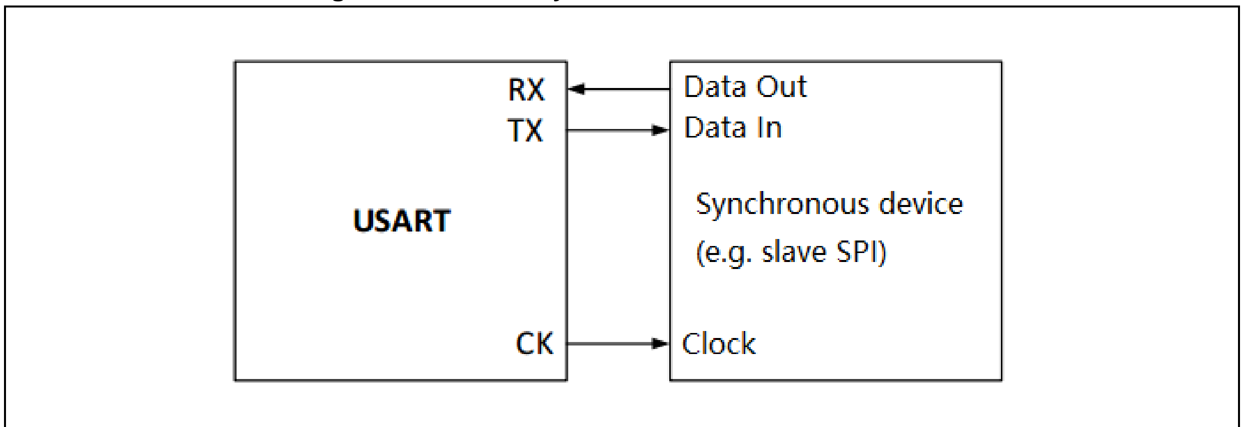
Figure 15. Smartcard waveform captured by LA

3.10 Example 10: synchronous mode

3.10.1 Function overview

USART supports users to control bidirectional synchronous serial communication in the master mode. The CK pin is used for USART transmitter clock output, and there is no clock pulse on CK pin during the start and stop bits. The LBCP bit in the USART_CTRL2 is used to select whether or not the clock pulse of the last data bit is output on the clock pin.

Figure 16. USART synchronous mode connection

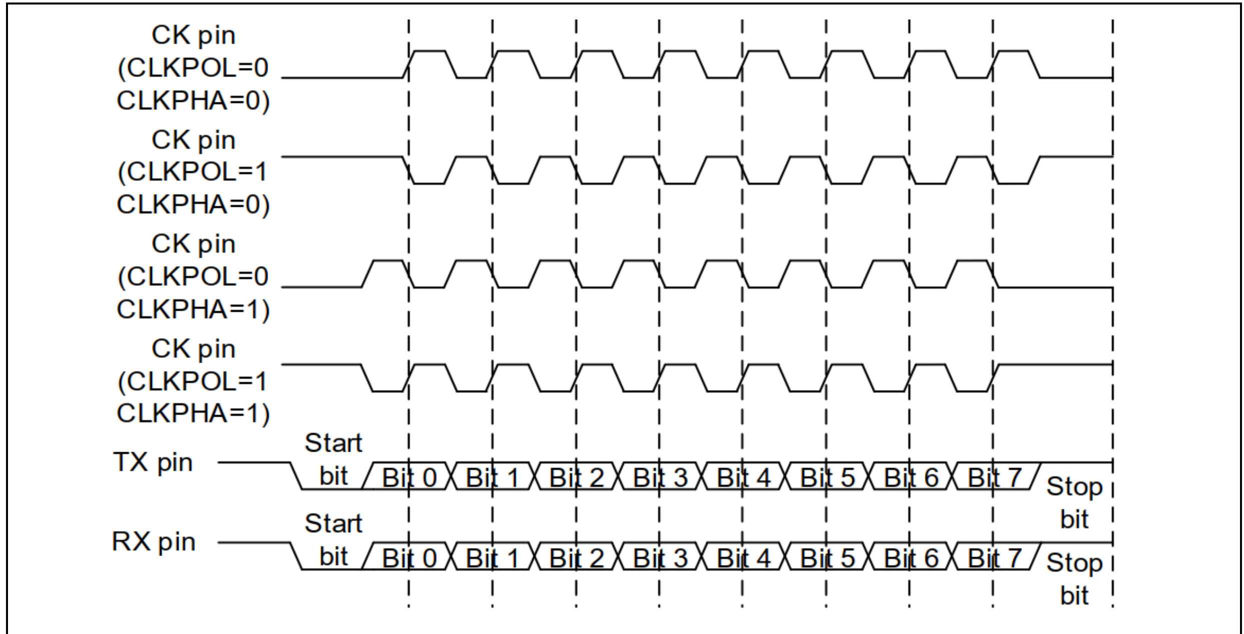


When the bus is idle, the external CK clock is not activated until the actual data arrives and when the break frame is transmitted. In synchronous mode, the USART transmitter works as it does in asynchronous mode, except that the data on TX is sent synchronously with the data on CK because the CK and TX are synchronized (dependent on the CLKPOL and CLKPHA).

In this example, the USART2 is configured as synchronous mode to communicate with SPI2 (serves as the slave). In the main function, check the corresponding flags and then execute data transmission and reception. The main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on the AT-START blink, indicating successful communication.

Use a Dupont cable to connect USART2_TX(PA2) and SPI2_MOSI(PB15), USART2_RX(PA3) and SPI2_MISO(PB14), USART2_CK(PA4) and SPI2_SCK(PB13).

Figure 17. USART synchronous mode data clock timing (DBN=0)



3.10.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\synchronous\mdk_v5

3.10.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART and SPI GPIOs, and initialization parameters
   - The main function transmits and receives data, and compares the transmit and receive data

2) Code

   - USART/SPI configuration code

```
/* configure USART and SPI clock, GPIO and other initialization parameters */
void usart_spi_configuration(void)
{
  gpio_init_type gpio_init_struct;
  spi_init_type spi_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable SPI2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);
```

```
        gpio_default_para_init(&gpio_init_struct);
        /* configure USART2 TX(PA2) and CK (PA4) pins as multiplexed push-pull output   */
        gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
        gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
        gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
        gpio_init_struct.gpio_pins = GPIO_PINS_2 | GPIO_PINS_4;
        gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
        gpio_init(GPIOA, &gpio_init_struct);
        /* configure USART2 RX(PA3) pin as pull-up input */
        gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
        gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
        gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
        gpio_init_struct.gpio_pins = GPIO_PINS_3;
        gpio_init_struct.gpio_pull = GPIO_PULL_UP;
        gpio_init(GPIOA, &gpio_init_struct);
        /* configure SPI2 MOSI(PB15) and SCK(PB13) pins as pull-up input */
        gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
        gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
        gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
        gpio_init_struct.gpio_pins = GPIO_PINS_15 | GPIO_PINS_13;
        gpio_init_struct.gpio_pull = GPIO_PULL_UP;
        gpio_init(GPIOB, &gpio_init_struct);
        /* configure SPI2 MISO(PB14) pin as multiplexed push-pull output */
        /* configure the spi2 miso pin */
        gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
        gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
        gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
        gpio_init_struct.gpio_pins = GPIO_PINS_14;
        gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
        gpio_init(GPIOB, &gpio_init_struct);
        /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
        usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
        /* configure USART2 clock polarity high, data capture on the second edge of clock phase, and the last
bit clock pulse output */
        usart_clock_config(USART2,USART_CLOCK_POLARITY_HIGH,
        SART_CLOCK_PHASE_2EDGE,USART_CLOCK_LAST_BIT_OUTPUT);
        /* enable USART2 clock output*/
        usart_clock_enable(USART2, TRUE);
```

```
    /* enable USART2 transmitter and receiver */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    /* enable USART */
    usart_enable(USART2, TRUE);
    /* configure SPI2 parameters, two-wire unidirectional full-duplex, slave mode, LSB first, 8-bit data bit*/
    spi_default_para_init(&spi_init_struct);
    spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;
    spi_init_struct.master_slave_mode =SPI_MODE_SLAVE;
    spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;
    spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;
    /* clock polarity high, data capture on the second edge of clock phase, software CS mode enable */
    spi_init_struct.clock_polarity= SPI_CLOCK_POLARITY_HIGH;
    spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;
    spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;
    spi_init(SPI2, &spi_init_struct);
    /* enable SPI */
    spi_enable(SPI2, TRUE);
}
```

- Main function code

```
/* define buffer */
#define COUNTOF(a)                      (sizeof(a) / sizeof(*(a)))
#define USART2_TX_BUFFER_SIZE           (COUNTOF(usart2_tx_buffer) - 1)
#define SPI2_TX_BUFFER_SIZE             (COUNTOF(spi2_tx_buffer) - 1)
#define DYMMY_BYTE                      0x00

uint8_t usart2_tx_buffer[] = "usart synchronous example: usart2 -> spi2";
uint8_t spi2_tx_buffer[] = "usart synchronous example: spi2 -> usart2";
uint8_t usart2_rx_buffer[SPI2_TX_BUFFER_SIZE];
uint8_t spi2_rx_buffer[USART2_TX_BUFFER_SIZE];
uint8_t usart2_tx_counter = 0x00;
uint8_t spi2_tx_counter = 0x00;
uint8_t usart2_rx_counter = 0x00;
uint8_t spi2_rx_counter = 0x00;
uint8_t usart2_tx_buffer_size = USART2_TX_BUFFER_SIZE;
uint8_t spi2_tx_buffer_size = SPI2_TX_BUFFER_SIZE;
/*buffer compare function */
```

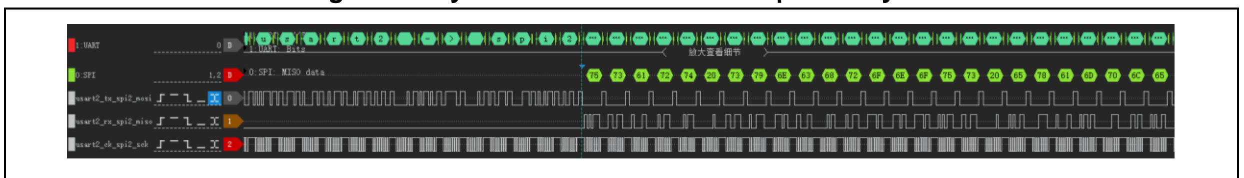
```
uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
{
  while(buffer_length--)
  {
    if(*pbuffer1 != *pbuffer2)
    {
      return 0;
    }
    pbuffer1++;
    pbuffer2++;
  }
  return 1;
}
/* main function */
int main(void)
{
  /* configure system clock and initialize resources on the AT-START board */
  system_clock_config();
  at32_board_init();
  /* configure USART and SPI clocks, GPIO and related initialization parameters */
  usart_spi_configuration();
  /* wait for the end of data transfer from USART2_TX to SPI2_MOSI */
  while(spi2_rx_counter < usart2_tx_buffer_size)
  {
    /* wait until the USART2 TDBE bit is set, and transmit data */
    while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
    usart_data_transmit(USART2, usart2_tx_buffer[usart2_tx_counter++]);
    /* wait until the SPI2 RDBF bit is set, and receive data */
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);
    spi2_rx_buffer[spi2_rx_counter++] = spi_i2s_data_receive(SPI2);
    /* wait until the USART2 RDBF bit is set, read the receive data buffer, clear null byte and flag */
    while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
    usart_data_receive(USART2);
  }
  /* wait for the end of data transfer from SPI2_MISO to USART2_RX */
  while(usart2_rx_counter < spi2_tx_buffer_size)
  {
    /* wait until the SPI2 TDBE bit is set, and transmit data */
```

```
            while(spi_i2s_flag_get(SPI2, SPI_I2S_TDBE_FLAG) == RESET);
            spi_i2s_data_transmit(SPI2, spi2_tx_buffer[spi2_tx_counter++]);
            /* wait until the USART2 TDBE bit is set, write null byte to the transmit data buffer to provide clock
for the slave */
            while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
            usart_data_transmit(USART2, DYMMY_BYTE);
            /* wait until the USART2 RDBF bit is set, and receive data */
            while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
            usart2_rx_buffer[usart2_rx_counter++] = usart_data_receive(USART2);
        }
        while(1)
        {
            /* compare transmit and receive data buffers */
            if(buffer_compare(usart2_tx_buffer, spi2_rx_buffer, USART2_TX_BUFFER_SIZE) && \
                buffer_compare(spi2_tx_buffer, usart2_rx_buffer, SPI2_TX_BUFFER_SIZE))
            {
                at32_led_toggle(LED2);
                at32_led_toggle(LED3);
                at32_led_toggle(LED4);
                delay_sec(1);
            }
        }
    }
```

3.10.4 Test result

LEDs on the AT-START board blink, indicating normal data transmission and reception. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 18. synchronous waveform captured by LA



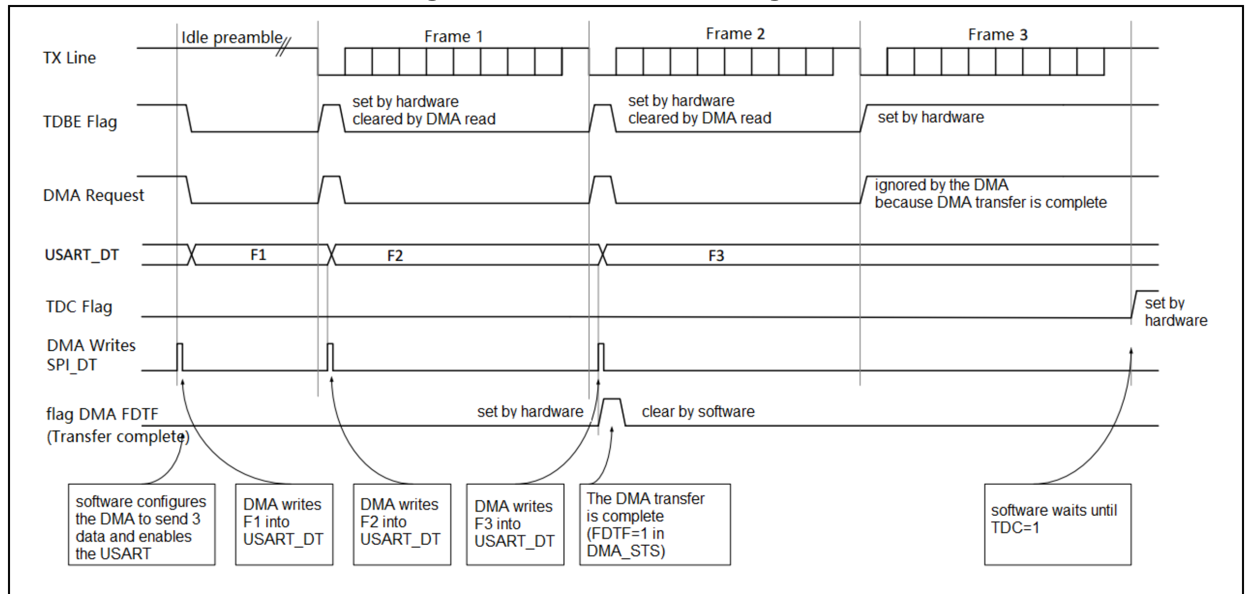
3.11 Example 11: Transmission using DMA

3.11.1 Function overview

Users can use DMA to achieve continuous high-speed transmission for USART, and DMA requests for RX buffer and TX buffer are generated separately.

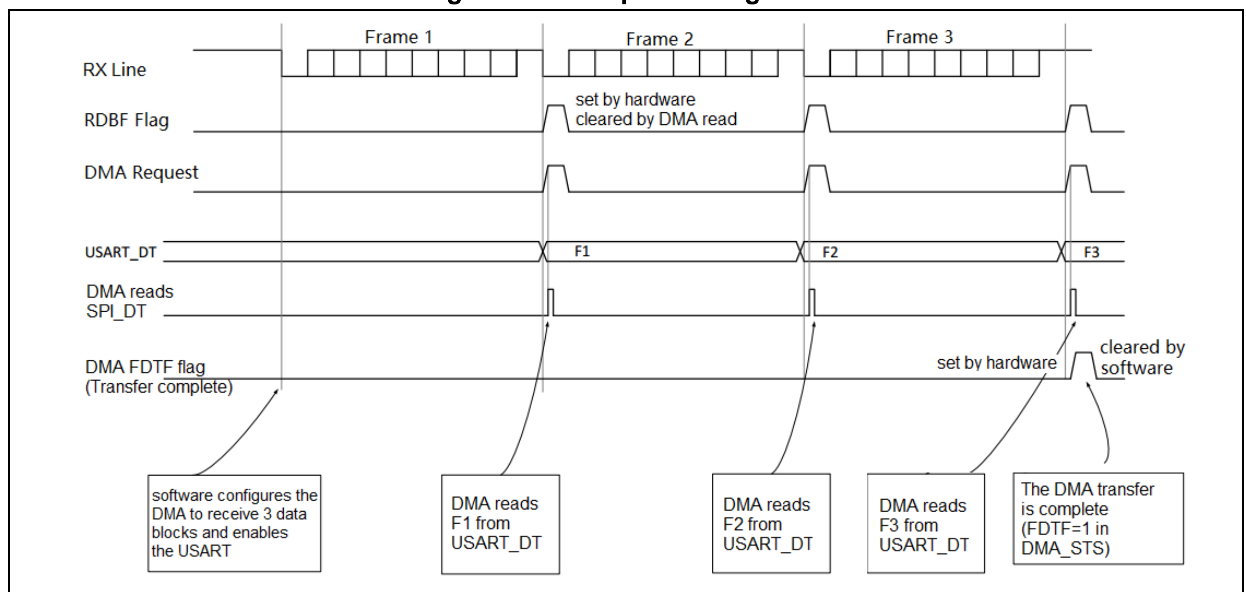
Set the DMATEN bit in the USART_CTRL3 register to enable DMA transmitter. The TDBE bit is set when the transmit data register is empty, and data is transmitted from the specified SRAM area to the USART_DT register.

Figure 19. Transmission using DMA



Set the DMAREN bit in the USART_CTRL3 register to enable DMA receiver. The RDBF bit is set each time the USART receives a byte, and data is transmitted from the USART_DT register to the specified SRAM area.

Figure 20. Reception using DMA



In this example, USART2 and USART3 are configured as two-wire unidirectional full-duplex mode

and use DMA for data transmission. The DMA interrupt handler function contains DMA transfer status and interrupt flags. The main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on the AT-START blink, indicating successful communication.

Use a Dupont cable to connect USART2_TX(PA2) and USART3_RX(PB11), USART2_RX(PA3) and USART3_TX(PB10).

3.11.2 Resources

1) Hardware

   AT32F403A AT-START BOARD

2) Software

   \project\at_start_f403a\examples\usart\transfer_by_dma_interrupt\mdk_v5

3.11.3 Software design

1) Configuration process
   - Configure clocks, the corresponding USART GPIOs and initialization parameters
   - Configure DMA and interrupt parameters
   - Compare transmit and receive data
   - Interrupt handler function

2) Code
   - USART configuration code

```
/* configure USART clock, GPIO and initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 and its GPIO peripheral clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure USART2 TX(PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
```

```
    /* configure USART3 TX(PB10) pin as multiplexed push-pull output */
    gpio_init_struct.gpio_pins = GPIO_PINS_10;
    gpio_init(GPIOB, &gpio_init_struct);
    /* configure USART2 RX(PA3) pin as pull-up input */
    gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
    gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
    gpio_init_struct.gpio_pins = GPIO_PINS_3;
    gpio_init_struct.gpio_pull = GPIO_PULL_UP;
    gpio_init(GPIOA, &gpio_init_struct);
    /* configure USART3 RX(PB11) pin as pull-up input */
    gpio_init_struct.gpio_pins = GPIO_PINS_11;
    gpio_init(GPIOB, &gpio_init_struct);

    /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART2 transmitter and receiver */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    /* enable DMATEN/DMAREN */
    usart_dma_transmitter_enable(USART2, TRUE);
    usart_dma_receiver_enable(USART2, TRUE);
    /* enable USART2 */
    usart_enable(USART2, TRUE);

    /* configure USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART3 transmitter and receiver */
    usart_transmitter_enable(USART3, TRUE);
    usart_receiver_enable(USART3, TRUE);
    /* enable DMATEN/DMAREN */
    usart_dma_transmitter_enable(USART3, TRUE);
    usart_dma_receiver_enable(USART3, TRUE);
    /* enable USART3 */
    usart_enable(USART3, TRUE);
  }
```

- USART configuration code

```
/* configure DMA clock, channel and initialization parameters */
void dma_configuration(void)
{
  dma_init_type dma_init_struct;
  /* enable DMA1 clock */
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
  /* configure USART2_TX DMA1 CHANNEL1 */
  dma_reset(DMA1_CHANNEL1); /* reset DMA1 channel1 to default configuration */
  dma_default_para_init(&dma_init_struct);
  dma_init_struct.buffer_size = USART2_TX_BUFFER_SIZE; /* set DMA buffer size*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /*data transfer direction: memory-to-peripheral */
  dma_init_struct.memory_base_addr = (uint32_t)usart2_tx_buffer; /* memory address */
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_ BYTE; /* data width: in BYTE */
  dma_init_struct.memory_inc_enable = TRUE; /* memory address increment: incremented by 1 after one data transmission */
  dma_init_struct.peripheral_base_addr = (uint32_t)&USART2->dt; /* peripheral address USART2_DT register */
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE; /*data width*/
  dma_init_struct.peripheral_inc_enable = FALSE; /*peripheral address increment disabled, always USART2_DT*/
  dma_init_struct.priority = DMA_PRIORITY_MEDIUM; /*DMA priority: medium */
  dma_init_struct.loop_mode_enable = FALSE; /* loop mode disabled */
  dma_init(DMA1_CHANNEL1, &dma_init_struct); /*configure DMA1 channel1 as mentioned above*/
  /* enable DMA1 CHANNEL1 full data transfer interrupt */
  dma_interrupt_enable(DMA1_CHANNEL1, DMA_FDT_INT, TRUE);
  /* configure DMA1 CHANNEL1 interrupt priority */
  nvic_irq_enable(DMA1_Channel1_IRQn, 0, 0);
  /* configure DMA1 CHANNEL1 flexible map to USART2_TX*/
  dma_flexible_config(DMA1, FLEX_CHANNEL1, DMA_FLEXIBLE_UART2_TX);

  /* configure DMA1 CHANNEL2 for USART2_RX */
  dma_reset(DMA1_CHANNEL2);
  dma_default_para_init(&dma_init_struct);
  dma_init_struct.buffer_size = USART3_TX_BUFFER_SIZE;
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;
  dma_init_struct.memory_base_addr = (uint32_t)usart2_rx_buffer;
```

```
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)&USART2->dt;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA1_CHANNEL2, &dma_init_struct);
/* enable DMA1 CHANNEL2 full data transfer interrupt */
dma_interrupt_enable(DMA1_CHANNEL2, DMA_FDT_INT, TRUE);
/* configure DMA1 CHANNEL2 interrupt priority */
nvic_irq_enable(DMA1_Channel2_IRQn, 0, 0);
/* configure DMA1 CHANNEL2 flexible map to USART2_RX*/
dma_flexible_config(DMA1, FLEX_CHANNEL2, DMA_FLEXIBLE_UART2_RX);

/* configure DMA1 CHANNEL3 for USART3_TX */
dma_reset(DMA1_CHANNEL3);
dma_default_para_init(&dma_init_struct);
dma_init_struct.buffer_size = USART3_TX_BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_base_addr = (uint32_t)usart3_tx_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)&USART3->dt;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA1_CHANNEL3, &dma_init_struct);
/* enable DMA1 CHANNEL3 full data transfer interrupt */
dma_interrupt_enable(DMA1_CHANNEL3, DMA_FDT_INT, TRUE);
/* configure DMA1 CHANNEL3 interrupt priority */
nvic_irq_enable(DMA1_Channel3_IRQn, 0, 0);
/* configure DMA1 CHANNEL3 flexible map to USART3_TX*/
dma_flexible_config(DMA1, FLEX_CHANNEL3, DMA_FLEXIBLE_UART3_TX);

/* configure DMA1 CHANNEL4 for USART3_RX*/
dma_reset(DMA1_CHANNEL4);
```

```
    dma_default_para_init(&dma_init_struct);
    dma_init_struct.buffer_size = USART2_TX_BUFFER_SIZE;
    dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;
    dma_init_struct.memory_base_addr = (uint32_t)usart3_rx_buffer;
    dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
    dma_init_struct.memory_inc_enable = TRUE;
    dma_init_struct.peripheral_base_addr = (uint32_t)&USART3->dt;
    dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
    dma_init_struct.peripheral_inc_enable = FALSE;
    dma_init_struct.priority= DMA_PRIORITY_MEDIUM;
    dma_init_struct.loop_mode_enable = FALSE;
    dma_init(DMA1_CHANNEL4, &dma_init_struct);
    /* enable DMA1 CHANNEL4 full data transfer interrupt */
    dma_interrupt_enable(DMA1_CHANNEL4, DMA_FDT_INT, TRUE);
    /* configure DMA1 CHANNEL4 interrupt priority */
    nvic_irq_enable(DMA1_Channel4_IRQn, 0, 0);
    /* configure DMA1 CHANNEL4 flexible map to USART3_RX*/
    dma_flexible_config(DMA1, FLEX_CHANNEL4, DMA_FLEXIBLE_UART3_RX);
    /* enable DMA1 CHANNELx */
    dma_channel_enable(DMA1_CHANNEL2, TRUE); /* usart2 rx begin dma receiving */
    dma_channel_enable(DMA1_CHANNEL4, TRUE); /* usart3 rx begin dma receiving */
    dma_channel_enable(DMA1_CHANNEL1, TRUE); /* usart2 tx begin dma transmitting */
    dma_channel_enable(DMA1_CHANNEL3, TRUE); /* usart3 tx begin dma transmitting */
}
```

- Main function code

```
/* define buffer */
#define COUNTOF(a)                    (sizeof(a) / sizeof(*(a)))
#define USART2_TX_BUFFER_SIZE           (COUNTOF(usart2_tx_buffer) - 1)
#define USART3_TX_BUFFER_SIZE           (COUNTOF(usart3_tx_buffer) - 1)
uint8_t usart2_tx_buffer[] = "usart transfer by dma interrupt: usart2 -> usart3 using dma";
uint8_t usart3_tx_buffer[] = "usart transfer by dma interrupt: usart3 -> usart2 using dma";
uint8_t usart2_rx_buffer[USART3_TX_BUFFER_SIZE];
uint8_t usart3_rx_buffer[USART2_TX_BUFFER_SIZE];
volatile uint8_t usart2_tx_dma_status = 0;
volatile uint8_t usart2_rx_dma_status = 0;
volatile uint8_t usart3_tx_dma_status = 0;
volatile uint8_t usart3_rx_dma_status = 0;
```

```
/*buffer compare function */
uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
{
  while(buffer_length--)
  {
    if(*pbuffer1 != *pbuffer2)
    {
      return 0;
    }
    pbuffer1++;
    pbuffer2++;
  }
  return 1;
}
/* main function */
int main(void)
{
  /* configure system clock and initialize resources on AT-START board */
  system_clock_config();
  at32_board_init();
  /* configure interrupt, clock, GPIO, USART and DMA initialization parameters */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  usart_configuration();
  dma_configuration();
  /* wait for the end of DMA transmission; transfer complete flag is set by the respective interrupt function
*/
  while((usart2_tx_dma_status == 0) || (usart2_rx_dma_status == 0) ||
   (usart3_tx_dma_status == 0) || (usart3_rx_dma_status == 0));
  while(1)
  {
    /* compare transmit and receive data buffers */
    if(buffer_compare(usart2_tx_buffer, usart3_rx_buffer, USART2_TX_BUFFER_SIZE) && \
       buffer_compare(usart3_tx_buffer, usart2_rx_buffer, USART3_TX_BUFFER_SIZE))
    {
      at32_led_toggle(LED2);
      at32_led_toggle(LED3);
      at32_led_toggle(LED4);
      delay_sec(1);
```

```
        }
      }
    }
```

■ Interrupt handler function code

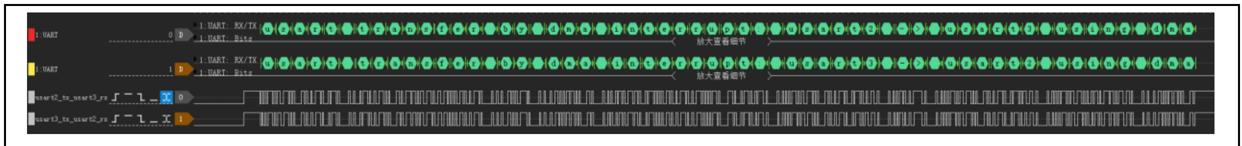
```
  /* UART2_TX transfer complete interrupt handler function */
  void DMA1_Channel1_IRQHandler(void)
  {
    if(dma_flag_get(DMA1_FDT1_FLAG))
    /* confirm the USART2_TX transfer complete; set usart2_tx_dma_status flag, clear interrupt flag and
disable interrupt channel */
    {
      usart2_tx_dma_status = 1;
      dma_flag_clear(DMA1_FDT1_FLAG);
      dma_channel_enable(DMA1_CHANNEL1, FALSE);
    }
  }
  /* UART2_RX transfer complete interrupt handler function */
  void DMA1_Channel2_IRQHandler(void)
  {
    if(dma_flag_get(DMA1_FDT2_FLAG))
    /* confirm the USART2_RX transfer complete; set usart2_rx_dma_status flag, clear interrupt flag and
disable interrupt channel */
    {
      usart2_rx_dma_status = 1;
      dma_flag_clear(DMA1_FDT2_FLAG);
      dma_channel_enable(DMA1_CHANNEL2, FALSE);
    }
  }
  /* UART3_TX transfer complete interrupt handler function */
  void DMA1_Channel3_IRQHandler(void)
  {
    if(dma_flag_get(DMA1_FDT3_FLAG))
    /* confirm the USART3_TX transfer complete; set usart3_tx_dma_status flag, clear interrupt flag and
disable interrupt channel */
    {
      usart3_tx_dma_status = 1;
      dma_flag_clear(DMA1_FDT3_FLAG);
      dma_channel_enable(DMA1_CHANNEL3, FALSE);
```

```
    }
  }
  /* UART3_RX transfer complete interrupt handler function */
  void DMA1_Channel4_IRQHandler(void)
  {
    if(dma_flag_get(DMA1_FDT4_FLAG))
    /* confirm USART3_RX transfer complete; set usart3_rx_dma_status flag, clear interrupt flag and disable interrupt channel */
    {
      usart3_rx_dma_status = 1;
      dma_flag_clear(DMA1_FDT4_FLAG);
      dma_channel_enable(DMA1_CHANNEL4, FALSE);
    }
  }
```

3.11.4 Test result

LEDs on the AT-START board blink, indicating normal data transmission and reception. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 21. transfer_by_dma_interrupt waveform captured by LA



3.12 Example 12: Communication by DMA polling

3.12.1 Function overview

Users can use DMA to achieve continuous high-speed transmission for USART, and DMA requests for RX buffer and TX buffer are generated separately.

Similar to example 11, set the DMATEN bit in the USART_CTRL3 register to enable DMA transmitter. The TDBE bit is set when the transmit data register is empty, and data is transmitted from the specified SRAM area to the USART_DT register. Set the DMAREN bit in the USART_CTRL3 register to enable DMA receiver. The RDBF bit is set each time the USART receives a byte, and data is transmitted from the USART_DT register to the specified SRAM area.

In this example, USART2 and USART3 are configured as two-wire unidirectional full-duplex mode and use DMA for data transmission. Different from example 11, this example does not use the DMA interrupt, and the DMA transfer status is obtained by polling in the main function. Similarly, the main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on the AT-START blink, indicating successful communication.

Use a Dupont cable to connect USART2_TX(PA2) and USART3_RX(PB11), USART2_RX(PA3) and USART3_TX(PB10).

3.12.2 Resources

1) Hardware

AT32F403A AT-START BOARD

2) Software

\project\at_start_f403a\examples\usart\transfer_by_dma_polling\mdk_v5

3.12.3 Software design

1) Configuration process

- Configure clocks, corresponding USART GPIOs and initialization parameters
- Configure DMA parameters
- Check the flag and compare transmit and receive data buffers

2) Code

- USART configuration code

```
/* configure USART clock, GPIOs and initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its peripheral GPIO clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 clock and its peripheral GPIO clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  gpio_default_para_init(&gpio_init_struct);
  /* configure USART2 TX(PA2) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure USART3 TX(PB10) pin as multiplexed push-pull output */
  gpio_init_struct.gpio_pins = GPIO_PINS_10;
  gpio_init(GPIOB, &gpio_init_struct);
  /* configure USART2 RX(PA3) pin as pull-up input */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
```

```
    gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
    gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
    gpio_init_struct.gpio_pins = GPIO_PINS_3;
    gpio_init_struct.gpio_pull = GPIO_PULL_UP;
    gpio_init(GPIOA, &gpio_init_struct);
    /* configure USART3 RX(PB11) pin as pull-up input */
    gpio_init_struct.gpio_pins = GPIO_PINS_11;
    gpio_init(GPIOB, &gpio_init_struct);

    /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
  check */
    usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART2 transmitter and receiver */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    /* enable DMATEN/DMAREN */
    usart_dma_transmitter_enable(USART2, TRUE);
    usart_dma_receiver_enable(USART2, TRUE);
    /* enable USART2 */
    usart_enable(USART2, TRUE);

    /* configure USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART3 transmitter and receiver */
    usart_transmitter_enable(USART3, TRUE);
    usart_receiver_enable(USART3, TRUE);
    /* enable DMATEN/DMAREN */
    usart_dma_transmitter_enable(USART3, TRUE);
    usart_dma_receiver_enable(USART3, TRUE);
    /* enable USART3 */
    usart_enable(USART3, TRUE);
  }
```

- USART configuration code

```
  /* configure DMA clock, channel and initialization parameters */
  void dma_configuration(void)
  {
    dma_init_type dma_init_struct;
```

```
/* enable DMA1 clock */
crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
/* configure DMA1 CHANNEL1 for USART2_TX */
dma_reset(DMA1_CHANNEL1); /* reset DMA1 channel1 to default configuration */
dma_default_para_init(&dma_init_struct);
dma_init_struct.buffer_size = USART2_TX_BUFFER_SIZE; /* set DMA buffer size*/
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /* data transfer direction: memory-to-peripheral */
dma_init_struct.memory_base_addr = (uint32_t)usart2_tx_buffer; /* memory address */
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_ BYTE; /*data width: in BYTE */
dma_init_struct.memory_inc_enable = TRUE; /* memory address increment: incremented by 1 after one data transmission */
dma_init_struct.peripheral_base_addr = (uint32_t)&USART2->dt; /* peripheral address USART2_DT register */
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE; /*data width*/
dma_init_struct.peripheral_inc_enable = FALSE; /* peripheral address increment disabled, always USART2_DT*/
dma_init_struct.priority = DMA_PRIORITY_MEDIUM; /*DMA priority: medium */
dma_init_struct.loop_mode_enable = FALSE; /* loop mode disabled */
dma_init(DMA1_CHANNEL1, &dma_init_struct); /* configure DMA1 channel1 as mentioned above*/
/* configure DMA1 CHANNEL1 flexible map to USART2_TX*/
dma_flexible_config(DMA1, FLEX_CHANNEL1, DMA_FLEXIBLE_UART2_TX);

/* configure DMA1 CHANNEL2 for USART2_RX */
dma_reset(DMA1_CHANNEL2);
dma_default_para_init(&dma_init_struct);
dma_init_struct.buffer_size = USART3_TX_BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_base_addr = (uint32_t)usart2_rx_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)&USART2->dt;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA1_CHANNEL2, &dma_init_struct);
/* configure DMA1 CHANNEL2 flexible map to USART2_RX*/
```

```
dma_flexible_config(DMA1, FLEX_CHANNEL2, DMA_FLEXIBLE_UART2_RX);

/* configure DMA1 CHANNEL3 for USART3_TX */
dma_reset(DMA1_CHANNEL3);
dma_default_para_init(&dma_init_struct);
dma_init_struct.buffer_size = USART3_TX_BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_base_addr = (uint32_t)usart3_tx_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)&USART3->dt;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA1_CHANNEL3, &dma_init_struct);
/* configure DMA1 CHANNEL3 flexible map to USART3_TX*/
dma_flexible_config(DMA1, FLEX_CHANNEL3, DMA_FLEXIBLE_UART3_TX);

/* configure DMA1 CHANNEL4 for USART3_RX */
dma_reset(DMA1_CHANNEL4);
dma_default_para_init(&dma_init_struct);
dma_init_struct.buffer_size = USART2_TX_BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_base_addr = (uint32_t)usart3_rx_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)&USART3->dt;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA1_CHANNEL4, &dma_init_struct);
/* configure DMA1 CHANNEL4 flexible map to USART3_RX*/
dma_flexible_config(DMA1, FLEX_CHANNEL4, DMA_FLEXIBLE_UART3_RX);
/* enable DMA1 CHANNELx */
dma_channel_enable(DMA1_CHANNEL2, TRUE); /* usart2 rx begin dma receiving */
dma_channel_enable(DMA1_CHANNEL4, TRUE); /* usart3 rx begin dma receiving */
```

```
    dma_channel_enable(DMA1_CHANNEL1, TRUE); /* usart2 tx begin dma transmitting */
    dma_channel_enable(DMA1_CHANNEL3, TRUE); /* usart3 tx begin dma transmitting */
}
```

- Main function code

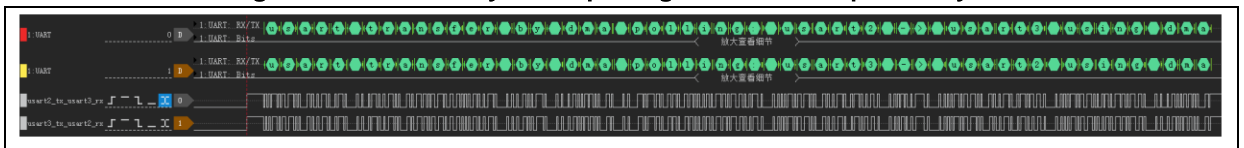
```
/* define buffer */
#define COUNTOF(a)                       (sizeof(a) / sizeof(*(a)))
#define USART2_TX_BUFFER_SIZE            (COUNTOF(usart2_tx_buffer) - 1)
#define USART3_TX_BUFFER_SIZE            (COUNTOF(usart3_tx_buffer) - 1)
uint8_t usart2_tx_buffer[] = "usart transfer by dma polling: usart2 -> usart3 using dma";
uint8_t usart3_tx_buffer[] = "usart transfer by dma polling: usart3 -> usart2 using dma";
uint8_t usart2_rx_buffer[USART3_TX_BUFFER_SIZE];
uint8_t usart3_rx_buffer[USART2_TX_BUFFER_SIZE];
/*buffer compare function */
uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
{
    while(buffer_length--)
    {
        if(*pbuffer1 != *pbuffer2)
        {
            return 0;
        }
        pbuffer1++;
        pbuffer2++;
    }
    return 1;
}
/* main function */
int main(void)
{
    /* configure system clock and initialize resources on AT-START board */
    system_clock_config();
    at32_board_init();
    /* configure clock, GPIO, USART and DMA initialization parameters */
    usart_configuration();
    dma_configuration();
    /* wait for the end of DMA transfer; transfer complete flag is obtained by the USART_STS register */
    while((dma_flag_get(DMA1_FDT1_FLAG) == RESET) || \
```

```
                (dma_flag_get(DMA1_FDT2_FLAG) == RESET) || \
                (dma_flag_get(DMA1_FDT3_FLAG) == RESET) || \
                (dma_flag_get(DMA1_FDT4_FLAG) == RESET));
    while(1)
    {
      /* compare transmit and receive data buffers */
      if(buffer_compare(usart2_tx_buffer, usart3_rx_buffer, USART2_TX_BUFFER_SIZE) && \
         buffer_compare(usart3_tx_buffer, usart2_rx_buffer, USART3_TX_BUFFER_SIZE))
      {
        at32_led_toggle(LED2);
        at32_led_toggle(LED3);
        at32_led_toggle(LED4);
        delay_sec(1);
      }
    }
  }
```

3.12.4 Test result

LEDs on the AT-START board blink, indicating normal data transmission and reception. Users can also use a logic analyzer to capture waveforms of the transmit and receive data for further verification and analysis.

Figure 22. transfer_by_dma_polling waveform captured by LA



3.13 Example 13: Tx/Rx swap

3.13.1 Function overview

AT32F421/435/437 also support Tx/Rx SWAP. When the TRPSWAP bit (USART_CTRL2[15]) is set, Tx/Rx pin can be swapped.

Configuration of this example is similar to example 6, except for the SWAP configuration. Both USART2 and USART3 are configured as two-wire unidirectional full-duplex mode. In the main function, check the corresponding flag and then perform data transmission and reception. The main function executes parity check for the transmit and receive data. If there is no parity error, three LEDs on AT-START board will blink to indicate successful communication.

Use a Dupont cable to connect swapped USART2_TX(PA3) and USART3_RX(PB11), swapped USART2_RX(PA2) and USART3_TX(PB10).

3.13.2 Resources

1) Hardware

   AT32F435 AT-START BOARD

2) Software

   \project\at_start_f435\examples\usart\tx_rx_swap\mdk_v5

3.13.3 Software design

1) Configuration process

   - Configure clocks, corresponding USART GPIOs and initialization parameters (including SWAP)
   - The main function transmits and receives data, and compares receive data and transmit data

2) Code

   - USART configuration code

```
/* configure USART clock, GPIO, baud rate and other initialization parameters */
void usart_configuration(void)
{
  gpio_init_type gpio_init_struct;
  /* enable USART2 clock and its peripheral GPIO clock */
  crm_periph_clock_enable(CRM_USART2_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* enable USART3 clock and its peripheral GPIO clock */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

  /* configure USART2 PA2/PA3 pins as multiplexed push-pull output */
  gpio_default_para_init(&gpio_init_struct);
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type   = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_pins = GPIO_PINS_2 | GPIO_PINS_3;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(GPIOA, &gpio_init_struct);
  /* configure PA2 as MUX7  (USART2_TX)*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE2, GPIO_MUX_7);
  /* configure PA3 as MUX7  (USART2_RX) */
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE3, GPIO_MUX_7);

  /* configure USART3 PB10/PB11 pins as multiplexed push-pull output */
```

```
    gpio_init_struct.gpio_pins = GPIO_PINS_10 | GPIO_PINS_11;
    gpio_init(GPIOB, &gpio_init_struct);
    /* configure PB10 as MUX7 (USART3_TX) */
    gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE10, GPIO_MUX_7);
    /* configure PB11 as MUX7 (USART3_RX) */
    gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE11, GPIO_MUX_7);
    /* configure USART2 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART2, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART2 transmitter and receiver */
    usart_transmitter_enable(USART2, TRUE);
    usart_receiver_enable(USART2, TRUE);
    /* enable USART2 Tx/Rx swap */
    usart_transmit_receive_pin_swap(USART2, TRUE);
    /* enable USART */
    usart_enable(USART2, TRUE);
    /* configure USART3 parameters, and set baud rate=115200, 8-bit data bit, 1-bit stop bit, without parity
check */
    usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
    /* enable USART3 transmitter and receiver */
    usart_transmitter_enable(USART3, TRUE);
    usart_receiver_enable(USART3, TRUE);
    /* enable USART */
    usart_enable(USART3, TRUE);
}
```

- Main function code

```
/* define buffer */
#define COUNTOF(a)                         (sizeof(a) / sizeof(*(a)))
#define USART2_TX_BUFFER_SIZE                 (COUNTOF(usart2_tx_buffer) - 1)
#define USART3_TX_BUFFER_SIZE                 (COUNTOF(usart3_tx_buffer) - 1)
uint8_t usart2_tx_buffer[] = "usart transfer by polling: usart2 -> usart3 using polling";
uint8_t usart3_tx_buffer[] = "usart transfer by polling: usart3 -> usart2 using polling";
uint8_t usart2_rx_buffer[USART3_TX_BUFFER_SIZE];
uint8_t usart3_rx_buffer[USART2_TX_BUFFER_SIZE];
uint8_t usart2_tx_counter = 0x00;
uint8_t usart3_tx_counter = 0x00;
uint8_t usart2_rx_counter = 0x00;
uint8_t usart3_rx_counter = 0x00;
```

```
uint8_t usart2_tx_buffer_size = USART2_TX_BUFFER_SIZE;
uint8_t usart3_tx_buffer_size = USART3_TX_BUFFER_SIZE;
/*buffer compare function */
uint8_t buffer_compare(uint8_t* pbuffer1, uint8_t* pbuffer2, uint16_t buffer_length)
{
  while(buffer_length--)
  {
    if(*pbuffer1 != *pbuffer2)
    {
      return 0;
    }
    pbuffer1++;
    pbuffer2++;
  }
  return 1;
}
/* main function */
int main(void)
{
  /* configure system clock and initialize resources on AT-START board */
  system_clock_config();
  at32_board_init();
  /* configure USART clock, GPIO, baud rate and other initialization parameters */
  usart_configuration();
  /* wait for the end of data transfer from USART2_TX to USART3_RX */
  while(usart3_rx_counter < usart2_tx_buffer_size)
  {
    /* wait until the USART2 TDBE bit is set, and transmit data */
    while(usart_flag_get(USART2, USART_TDBE_FLAG) == RESET);
    usart_data_transmit(USART2, usart2_tx_buffer[usart2_tx_counter++]);
    /* wait until the USART3 RDB bit is set, and receive data */
    while(usart_flag_get(USART3, USART_RDBF_FLAG) == RESET);
    usart3_rx_buffer[usart3_rx_counter++] = usart_data_receive(USART3);
  }
  /* wait for the end of data transfer from USART3_TX to USART2_RX */
  while(usart2_rx_counter < usart3_tx_buffer_size)
  {
    /* wait until the USART3 TDBE bit is set, and transmit data */
```

```
        while(usart_flag_get(USART3, USART_TDBE_FLAG) == RESET);
        usart_data_transmit(USART3, usart3_tx_buffer[usart3_tx_counter++]);
        /* wait until the USART2 RDBF bit is set, and receive data */
        while(usart_flag_get(USART2, USART_RDBF_FLAG) == RESET);
        usart2_rx_buffer[usart2_rx_counter++] = usart_data_receive(USART2);
    }
    while(1)
    {
        /* compare transmit and receive data buffers */
        if(buffer_compare(usart2_tx_buffer, usart3_rx_buffer, USART2_TX_BUFFER_SIZE) && \
           buffer_compare(usart3_tx_buffer, usart2_rx_buffer, USART3_TX_BUFFER_SIZE))
        {
            at32_led_toggle(LED2);
            at32_led_toggle(LED3);
            at32_led_toggle(LED4);
            delay_sec(1);
        }
    }
}
```

3.13.4 Test result

LEDs on the AT-START board blink, indicating normal data transmission and reception.

4 Revision history

Table 5. Document revision history

Date	Version	Revision note
2022.02.10	2.0.0	Initial release
2022.04.15	2.0.1	Added example 9.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.

© 2022 ARTERY Technology – All Rights Reserved