
**Implementing an emulated UART
on STM32F4 microcontrollers**

Introduction

This application note describes how to implement an emulated universal asynchronous receiver transmitter (UART) on microcontrollers of STM32F4 Series.

Such an emulation is needed in applications that require more UARTs than the ones available on STM32F4 microcontrollers.

The emulated UART is full-duplex, supports up to 9 data bits and baud rates up to 115200 bps. It also offers high flexibility, as any I/O pin can be used as TX or RX line. In addition, this UART emulation uses DMA to minimize CPU usage.

This application note provides a basic example of communication between a hardware and a software UART, as well as a summary of CPU load and firmware footprint.

The STSW-STM32156 firmware package associated with this document contains the source code of the UART emulator with all firmware modules required to run the example.

Contents

- 1 **UART emulator description** **6****
- 1.1 Main features 6
- 1.2 UART emulator block diagram 7
- 1.3 UART emulator principle 8
 - 1.3.1 Data transmission 8
 - 1.3.2 Single-frame transmission 8
 - 1.3.3 Multiple-frame transmission 8
- 1.4 Data reception 9
 - 1.4.1 Single-frame reception 9
 - 1.4.2 Multiple-frame reception 9
- 1.5 Baudrate 10

- 2 **Software description** **11****
- 2.1 Implementation structure 11
- 2.2 Transmission 12
 - 2.2.1 Frame transmission 12
 - 2.2.2 Transmission routine 13
- 2.3 Reception 14
 - 2.3.1 Frame reception 14
 - 2.3.2 Reception routine 15
- 2.4 UART emulator peripherals and main functions 16
 - 2.4.1 Peripheral settings 16
 - 2.4.2 Initialization and configuration function 16
 - 2.4.3 UART main functions 17

- 3 **Example** **18****
- 3.1 Hardware requirements 18
- 3.2 Software settings 19
- 3.3 Running the example 19
- 3.4 Frame waveforms 20

- 4 **UART emulator CPU load and footprint** **21****
- 4.1 CPU load 21

4.2 UART emulator memory footprint 21

5 Conclusion 22

6 Revision history 23

List of tables

Table 1.	Transmission/reception functions	17
Table 2.	Callback functions	17
Table 3.	UART emulator and UART hardware connection	18
Table 4.	UART CPU load	21
Table 5.	UART memory footprint	21
Table 6.	Document revision history	23

List of figures

Figure 1.	UART emulator block diagram	7
Figure 2.	9-bit data transmission waveforms	8
Figure 3.	9-bit data reception waveforms.	9
Figure 4.	UART emulator application level view	11
Figure 5.	Frame transmission routine flowchart.	12
Figure 6.	Transmission routine flowchart	13
Figure 7.	Frame reception routine flowchart	14
Figure 8.	Reception routine flowchart	15
Figure 9.	UART emulator and UART HW connection	18
Figure 10.	Example MDK-ARM™ workspaces	19
Figure 11.	UART emulator frame with no parity.	20
Figure 12.	UART emulator frame with odd parity.	20

1 UART emulator description

The principle consists in emulating the UART protocol via GPIO, EXTI, timer and DMA peripherals.

1.1 Main features

The main features of the UART emulator are the following:

- Full-duplex asynchronous communications up to 115200 bps
- Programmable data word length: from 5 to 9 bits
- Flexible GPIO usage: all GPIOs can be configured as UART_TX/RX
- Configurable number of stop bits: 1 or 2 stop bits
- Reception:
 - Frames are received at once
 - To receive new frames the UART has to be reinitialized
- Parity control
 - Transmission of parity bit
 - Parity check of received data frame
- Transfer detection flags
 - Receive complete
 - Transmit complete
- Error detection flags
 - Frame error
 - Parity error

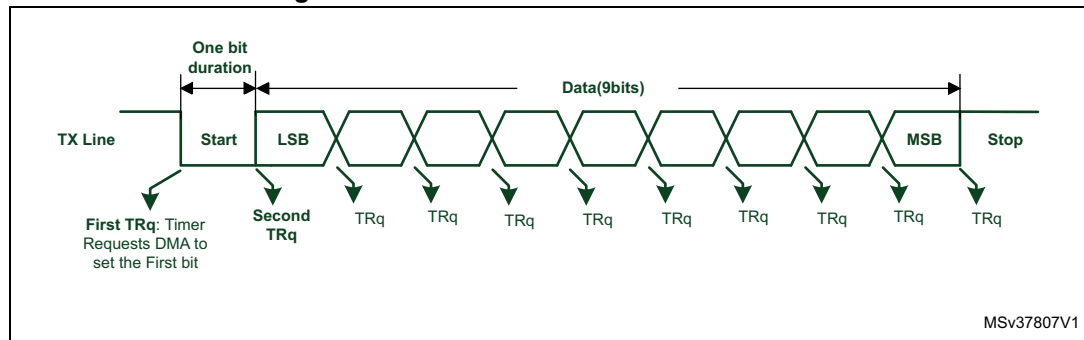
1.3 UART emulator principle

1.3.1 Data transmission

The transmitter can send 5-, 6-, 7-, 8- or 9-bit data frames, depending on user data length configuration.

Figure 2 shows the waveforms corresponding to a 9-bit transmit emulation.

Figure 2. 9-bit data transmission waveforms



1.3.2 Single-frame transmission

The transmission sequence includes the following steps:

1. The CPU formats the frame to be sent to the memory.
2. The timer sends to the DMA a request to set GPIO to 0 (start bit).
3. The timer starts counting the start bit duration according to the defined baudrate.
4. When the bit duration has elapsed, the timer sends to the DMA a request to set the next bit from memory.

Once the frame transmission is complete, the TC flag (UART transmission complete) is set.

1.3.3 Multiple-frame transmission

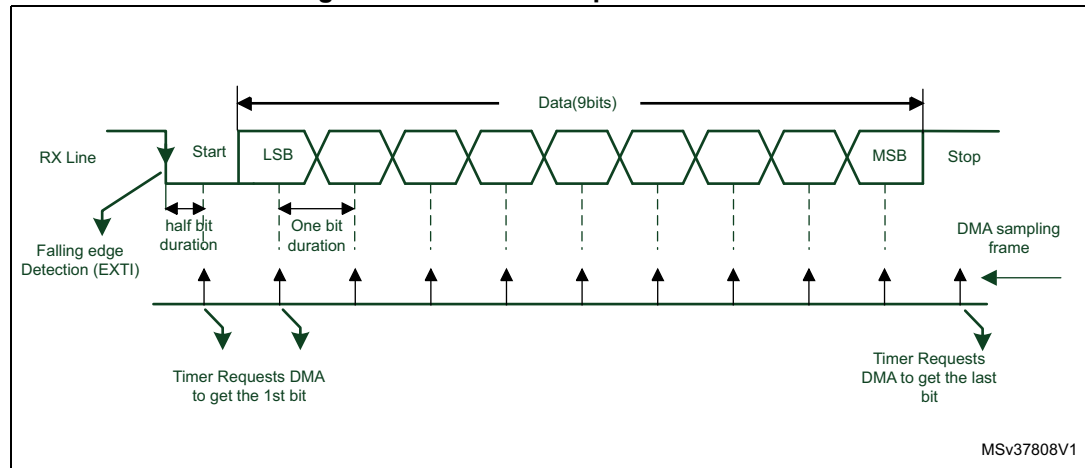
Multiple-frame transmission is based on two buffers. When the DMA transfers the first frame into the first buffer, the CPU formats the second frame in the second buffer. After each frame transfer, the DMA TC flag must be cleared and the DMA reconfigured with the new source address (the one of the first or of the second buffer). This operation is repeated until all frames are transmitted. When this is done, the TC flag (UART transfer complete flag) is set.

1.4 Data reception

The reception is launched when the start bit is detected via an external interrupt on UART RX line.

Figure 3 shows the waveforms corresponding to a 9-bit receive emulation.

Figure 3. 9-bit data reception waveforms



1.4.1 Single-frame reception

Reception starts when the start bit is detected. The reception sequence includes the following steps:

1. The CPU checks if the UART is ready and the RX buffer empty.
2. The timer sends a request to the DMA to start data transfer after a half bit duration.
3. The timer starts counting the bit duration according to the defined baudrate and then sends to the DMA a request to transfer the next bit. This step is repeated until the end of the frame.

Once the frame reception is complete, the RC flag (UART reception complete) is set.

1.4.2 Multiple-frame reception

When the falling edge of the start bit is detected, the reception goes on until all frames are received and the RC flag is set.

When the DMA transfer of each frame is complete, the CPU formats data and stores them in SRAM.

Note: During DMA transfer, the CPU is free for other tasks.

The application must assign the highest priority to the external interrupt for bit start detection.

1.5 Baudrate

The UART emulator supports baudrates up to 115200 bps.

The baudrate generation is ensured by the timer using different parameters, such as clock division, prescaler and period (ARR value).

The calculation of timer period (bit duration) is based on the following formula:

$$\text{timer_period} = \text{timer frequency} / \text{UART baudrate}$$

Example

- Configuration
 - Timer prescaler = 0
 - Timer clock division = 0
 - AHB clock = system clock = 84 MHz
 - APB2 frequency = 21 MHz
 - UART baudrate = 9600 bps

- Result

$$\text{APB2 prescaler} = 4$$

as a result

$$\text{timer frequency} = (84 / 4) \times 2 = 42 \text{ MHz}$$

and

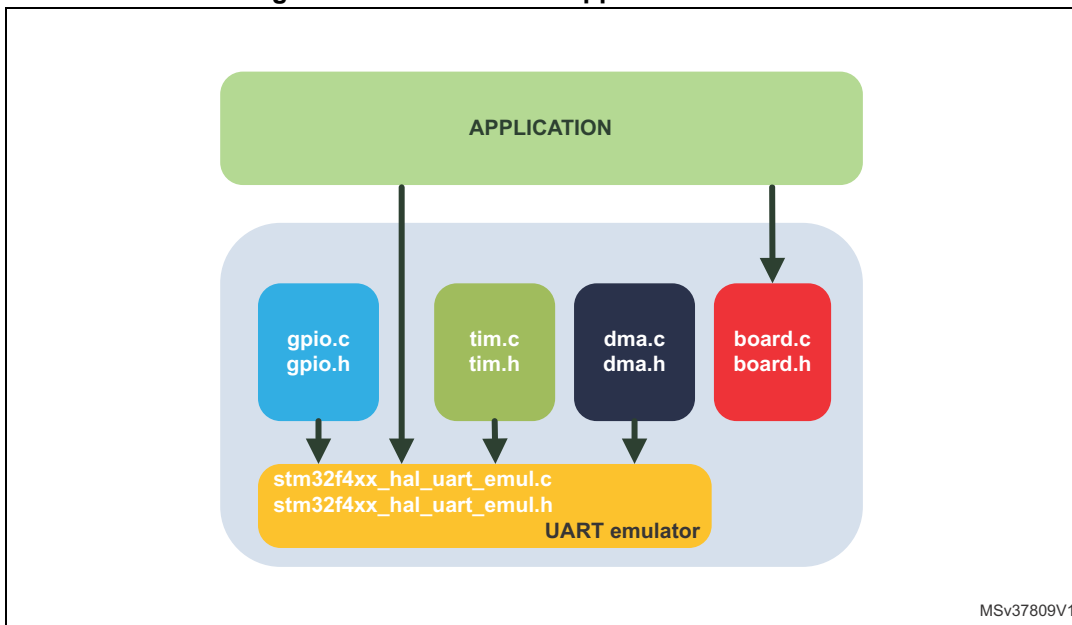
$$\text{timer period} = 42000000 / 9600$$

2 Software description

2.1 Implementation structure

Figure 4 shows the UART emulator software structure starting from the application level.

Figure 4. UART emulator application level view



2.2 Transmission

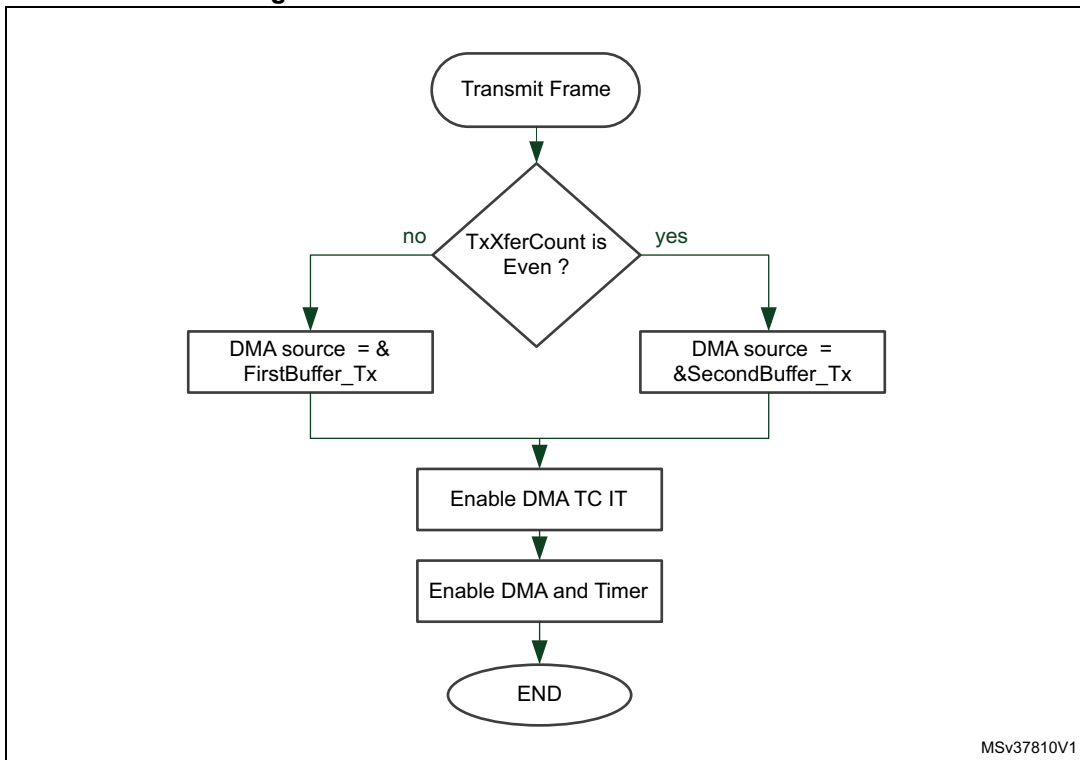
2.2.1 Frame transmission

Figure 5 describes the implementation of the function that sends a given number of bytes using DMA and timer.

TxXferCount is a counter incremented after the completion each DMA transfer.

FirstBuffer_Tx and **SecondBuffer_Tx**: the DMA source address switches between two buffers addresses. The first buffer is used by DMA as source data, while the second buffer is used by the CPU to format data.

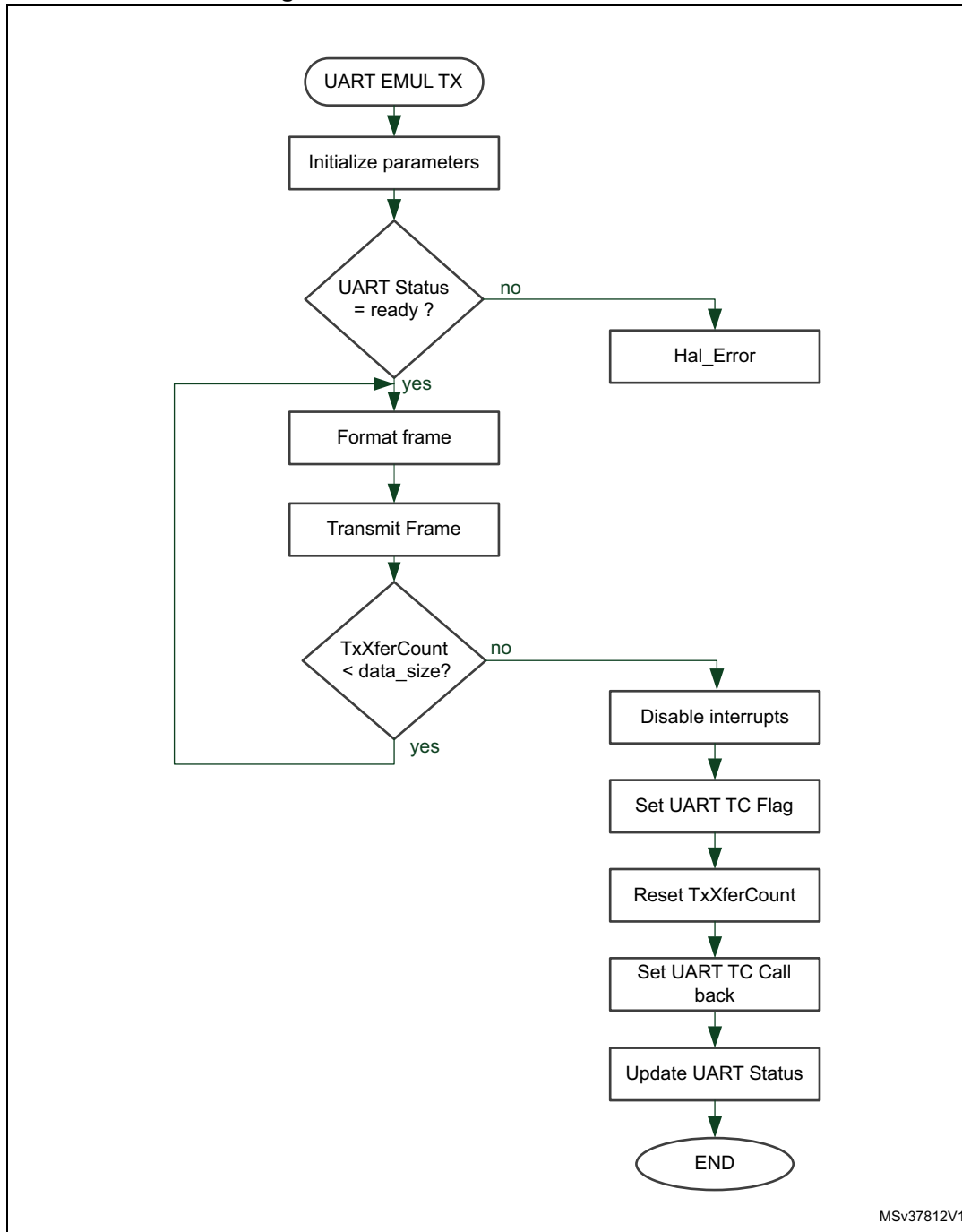
Figure 5. Frame transmission routine flowchart



2.2.2 Transmission routine

Figure 6 gives an overview of the UART emulator transmission routine.

Figure 6. Transmission routine flowchart



MSv37812V1

2.3 Reception

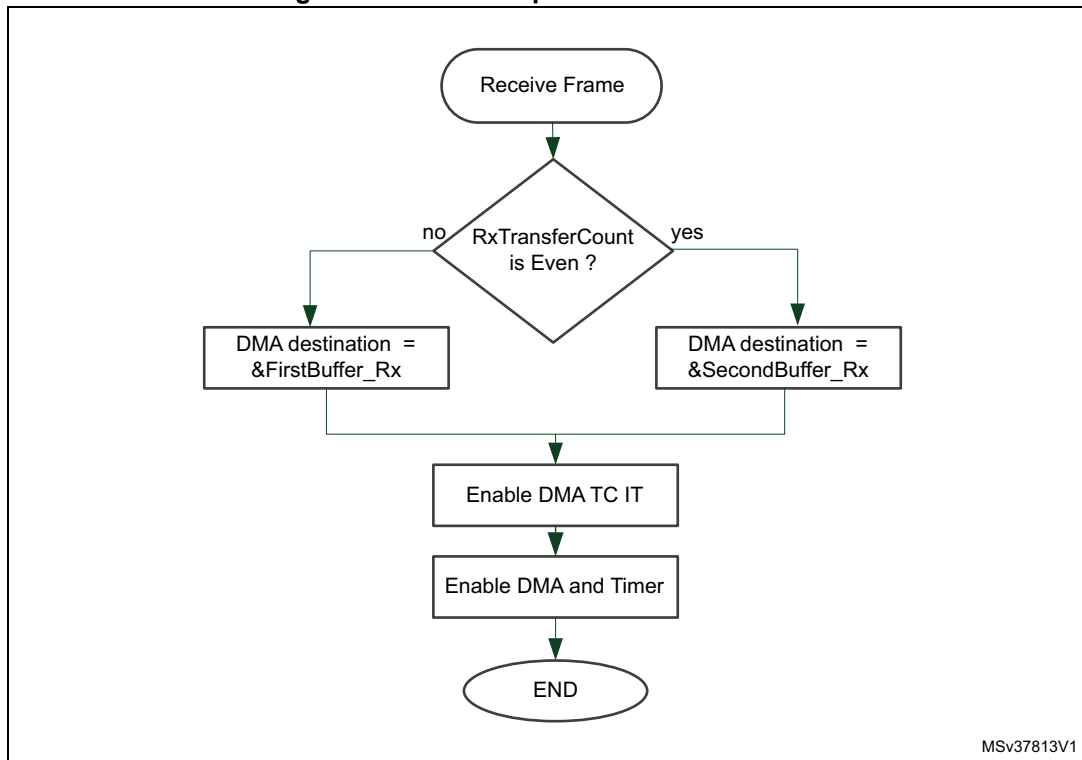
2.3.1 Frame reception

Figure 7 describes the implementation of the function that receives a given number of bytes via DMA and timer.

RxXferCount: is a counter that is incremented after the completion each DMA transfer.

FirstBuffer_Rx and **SecondBuffer_Rx**: the DMA destination address switches between two buffers addresses. The first buffer is used by DMA as data transfer destination, while the second buffer is used by the CPU to format data.

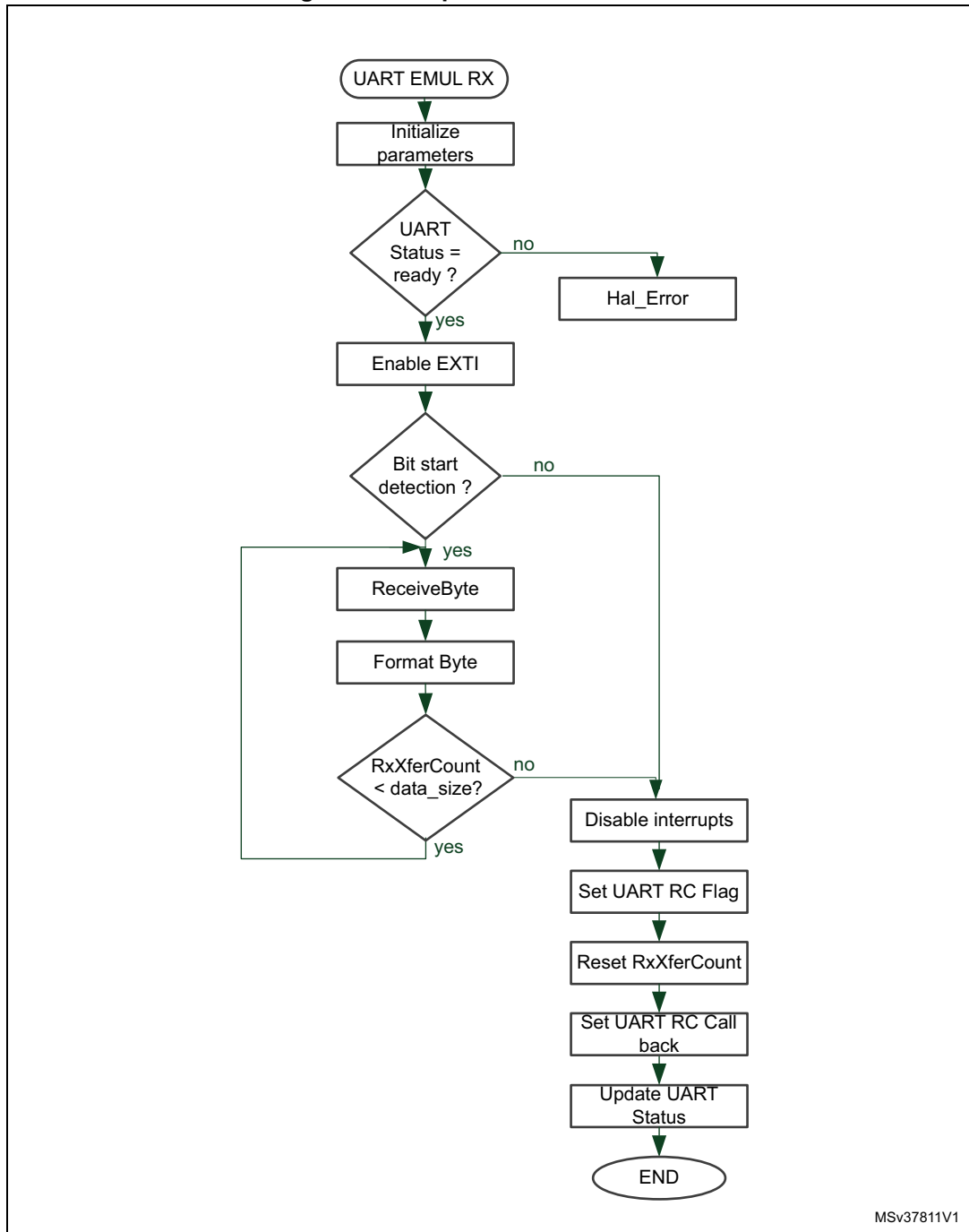
Figure 7. Frame reception routine flowchart



2.3.2 Reception routine

Figure 8 flowchart gives an overview of the UART emulator reception routine.

Figure 8. Reception routine flowchart



2.4 UART emulator peripherals and main functions

2.4.1 Peripheral settings

This section describes the configuration of the peripherals used inside the emulator.

- **GPIO**
 - BSRR and IDR are used as destination and source registers for DMA transfers.
 - Two pins must be configured as input and output by the user application.
 - The input pin mode is configured as an EXTI line with falling edge detection.
- **DMA2**
 - Channel 6 and stream 1 are used for transmission.
 - Channel 6 and stream 2 are used for reception.
 - The transfer is performed by words.
 - DMA transfer complete interrupt is used at the end of frame transfers.
- **Timer 1**
 - Timer channel 1 is configured as capture compare for DMA transmit requests.
 - Timer channel 2 is configured as capture compare for DMA receive requests.
 - No clock division: CKD[1:0] = 00.
 - No prescaler: PSC[15:0] = 0.
- **SRAM**

Four SRAM buffers are used to format data:

 - uint16_t pFirstBuffer_Tx[12] and uint16_t pSecondBuffer_Tx[12] are the buffers for formatting data in transmission mode.
 - uint16_t pFirstBuffer_Rx[12], uint16_t pSecondBuffer_Rx[12] are the buffers for formatting data in reception mode.

2.4.2 Initialization and configuration function

The initialization of the UART is performed by *HAL_UART_Emul_Init* function, which allows the user to:

- setup the following UART parameters:
 - Baudrate
 - Frame length
 - Stop bit
 - Parity.
- enable clocks for all the peripherals used: Timer, DMA, GPIOs
- configure the DMA: channel, stream, mode, TC interrupt,...
- configure the Timer: channel, period,...

2.4.3 UART main functions

This section provides a set of functions ensuring UART transmission/reception emulation.

Table 1. Transmission/reception functions

Function	Parameters	Description
<i>HAL_UART_Emul_Transmit_DMA</i> (<i>UART_Emul_HandleTypeDef *huart, uint8_t *pData, uint16_t Size</i>)	<i>huart</i> : UART Emulator handle <i>pData</i> : Pointer to data buffer <i>Size</i> : Amount of data to be sent	Sends data
<i>HAL_StatusTypeDef HAL_UART_Emul_Receive_DMA</i> (<i>UART_Emul_HandleTypeDef *huart, uint8_t *pData, uint16_t Size</i>)	<i>huart</i> : UART Emulator handle <i>pData</i> : Pointer to data buffer <i>Size</i> : Amount of data to be received	Receives data

Callback functions are also available. They allow the user to implement his own code in the user file.

Table 2. Callback functions

Function	Parameters	Description
<i>__weak void HAL_UART_Emul_TxCpltCallback</i> (<i>UART_Emul_HandleTypeDef *huart</i>)	<i>huart</i> : UART Emulator handle	This function is called at the end of transmit process.
<i>__weak void HAL_UART_Emul_RxCpltCallback</i> (<i>UART_Emul_HandleTypeDef *huart</i>)	<i>huart</i> : UART Emulator handle	This function is called at the end of receive process.
<i>__weak void HAL_UART_Emul_ErrorCallback</i> (<i>UART_Emul_HandleTypeDef *huart</i>)	<i>huart</i> : UART Emulator handle	This function is called when a communication error is detected.

3 Example

The example provided in STSW-STM32156 illustrates the data exchange between the UART emulator and the hardware UART.

3.1 Hardware requirements

The hardware required to run this example is the following:

- Two Nucleo boards (NUCLEO-F401RE)
- Two Mini-USB cables to power the boards and to connect the Nucleo embedded ST-LINK for debugging and programming.

The connection between the two Nucleo boards through UART lines is detailed in [Figure 9](#) and [Table 3](#).

Figure 9. UART emulator and UART HW connection

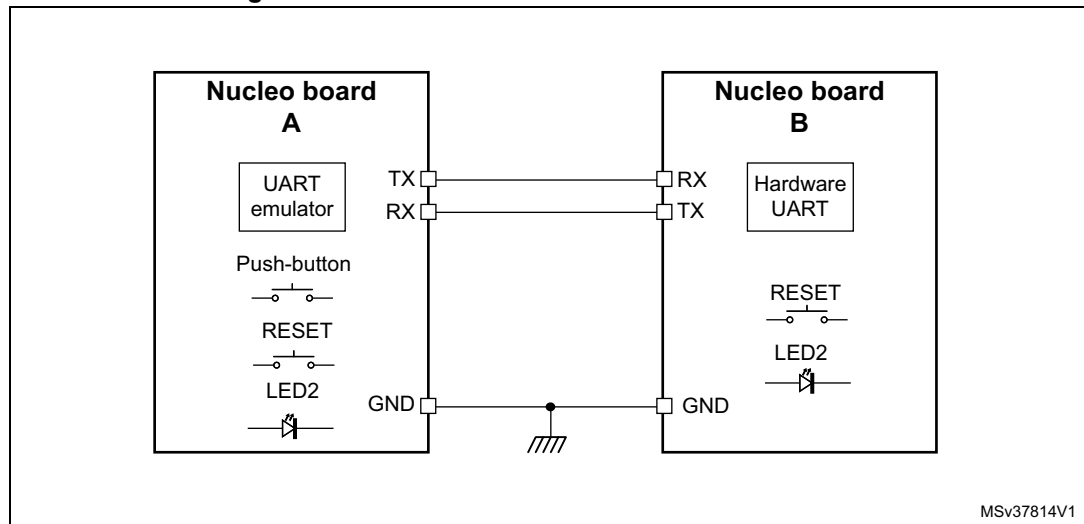


Table 3. UART emulator and UART hardware connection

Nucleo board A (UART emulator)	Nucleo board B (hardware UART)
UART TX (PC10)	UART RX (PA10)
UART RX (PC11)	UART TX (PA9)
GND	GND

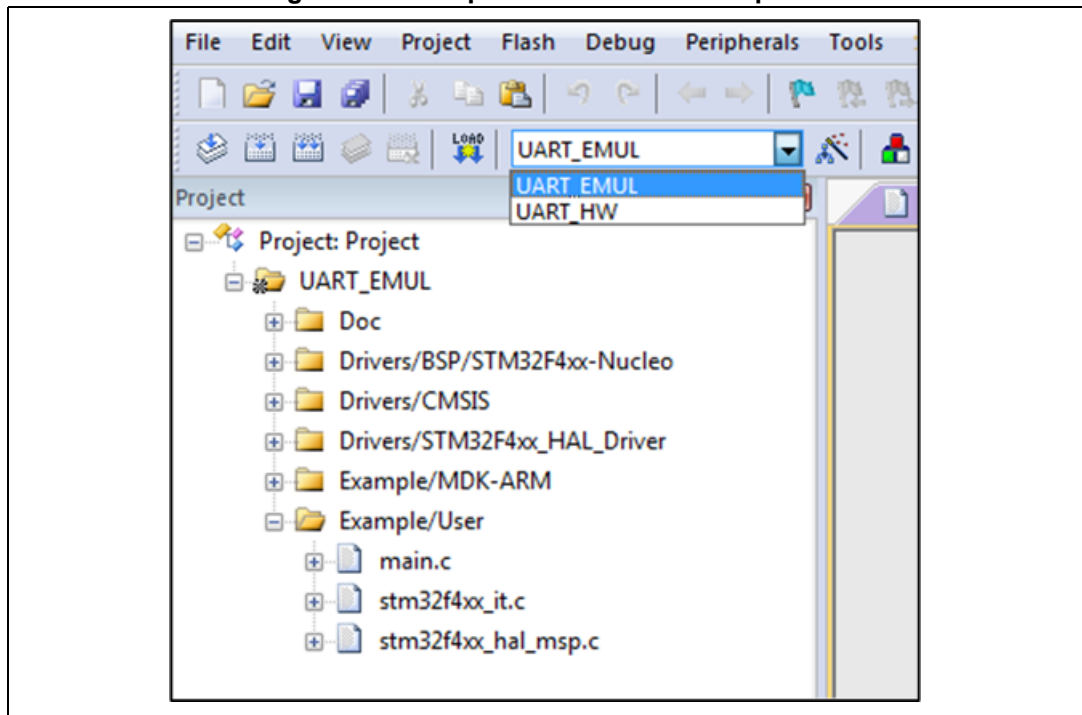
3.2 Software settings

The project example includes two workspaces: UART_EMUL and UART_HW (see [Figure 10](#)).

To make the program work, follow the steps described below:

1. Open your preferred toolchain (EWARM or MDK-ARM™).
2. Rebuild all files and load your image into target memory.
3. Run the example.

Figure 10. Example MDK-ARM™ workspaces



3.3 Running the example

To run the example follow the sequence below:

1. Power on the two boards.
2. Load the code in each board MCU.
3. Press the user button key on board A. The example then starts running and the UART emulator starts transmitting data.
4. The UART hardware receives the data and sends them back to UART emulator.
5. The data transmitted by the UART emulator is compared to received ones: if data do not match, the green LED (LED2) toggles continuously.

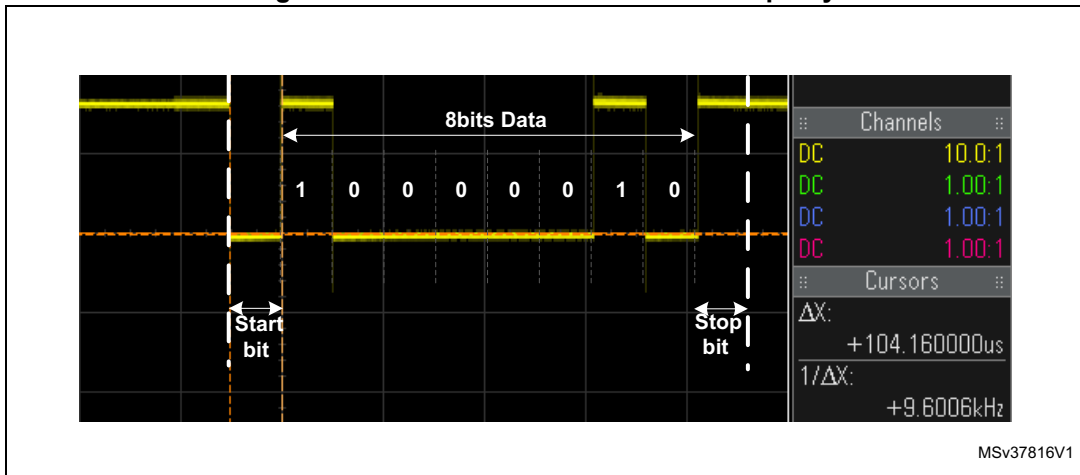
Note: For more details, refer to the *readme.txt* in the firmware package.

3.4 Frame waveforms

Figure 11 and Figure 12 show examples of configurations of 'A' character transfer:

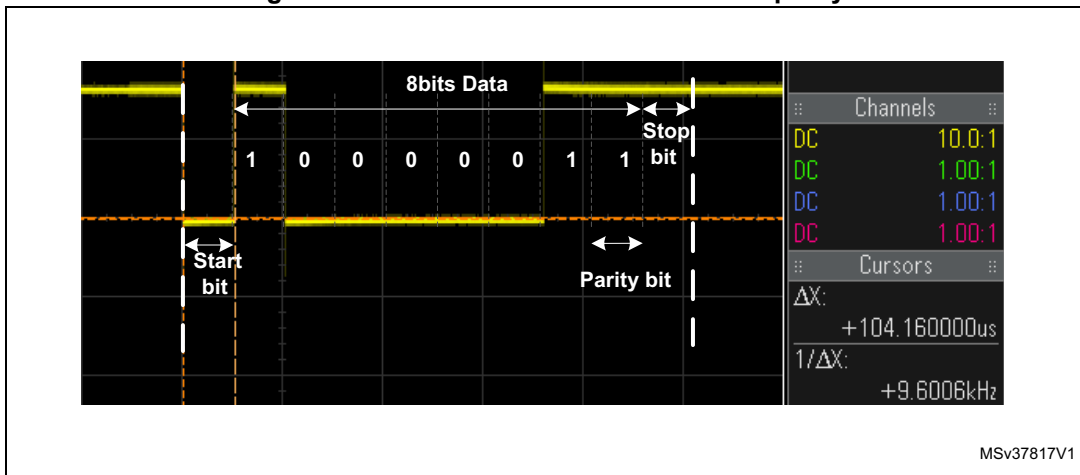
- First configuration (Figure 11):
 - Baudrate: 9600 baud
 - Word length: 8 bits
 - Number of stop bits: 1
 - Parity: none.

Figure 11. UART emulator frame with no parity



- Second configuration (Figure 12):
 - Baudrate: 9600 baud
 - Word length: 8 bits
 - Number of stop bits: 1
 - Parity: parity odd.

Figure 12. UART emulator frame with odd parity



4 UART emulator CPU load and footprint

The UART emulator uses the CPU for several tasks such as data formatting, DMA interrupt handling both transmission and reception, and EXTI interrupt handling for reception.

4.1 CPU load

The CPU load depends on whether the transmit or receive process is active. When UART full-duplex mode is active, the CPU load is increased. Refer to [Table 4](#) for an example.

The software settings used to obtain the results given in [Table 4](#) are the following:

- System clock: 84 MHz
- Toolchain: MDK-ARM™ V5.14, optimization level3 (-O3) for size
- Word length: 8 bits
- One stop bit
- No parity.

Table 4. UART CPU load

Baudrate (bps)	Transmission		Reception	
	Load CPU (%)	MIPS	Load CPU (%)	MIPS
4800	2.4	2	4	3
9600	5	4	8.8	7
115200	6	5	9	7.5

4.2 UART emulator memory footprint

[Table 5](#) gives an estimate of the code size required by the UART emulator software compiled with MDK-ARM™ V5.14, optimization level3 (-O3) for size.

Table 5. UART memory footprint

Flash memory footprint (bytes)	RAM footprint (bytes)
2872	400

5 Conclusion

This application note demonstrates that the product capability can be increased by adding an emulated UART.

This solution has many advantages, among them reduced CPU load by using DMA for data transfer. In addition, the user can configure any GPIO as UART transmitter or receiver.

6 Revision history

Table 6. Document revision history

Date	Revision	Changes
30-Mar-2015	1	Initial release.
04-Aug-2016	2	Updated document title. Updated <i>Introduction</i> and removed former <i>Table 1: Applicable products</i> . Updated <i>Section 1.1: Main features</i> and <i>Section 1.2: UART emulator block diagram</i> .

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved