AN14319

FlexIO Emulating UART with IRDA

Rev. 1.0 — 15 July 2024 Application note

Document information

Information	Content
Keywords	AN14319, MCXC444, MCXC242, FlexIO, UART, IRDA
Abstract	This application note describes FlexIO emulating UART with IRDA.

1 Introduction

This application note introduces how to use the universal peripheral module FlexIO for emulating the UART bus with IRDA. The FlexIO peripheral, initially introduced on the MCXC242 and MCXC444 family, is a highly configurable module capable of emulating a wide range of different communication protocols. These communication protocols include UART, I2C, SPI, I2S, and so on.

The standalone peripheral module FlexIO is not intended to replace the UART peripheral, but to serve as an additional peripheral module of the MCU. The important feature of this peripheral is that it enables the user to build their own peripheral directly in the MCU.

This use case for the UART module creates a simple software driver based on the independent receiver and transceiver. For this demonstration, the Freescale Tower System has been used. The maximum tested baud rate of the emulated UART bus is 115,200 baud.

This use case uses the FlexIO UART driver to create an IRDA protocol, with UART supporting IRDA. It uses two more timers apart from the FlexIO UART driver to encode and decode the UART signal into the IRDA waveform.

2 Features

The main features of the FlexIO peripheral module are as follows:

- FlexIO means flexible input and output peripheral
- Highly configurable module with a wide range of functionality
- Allows emulation of standard communication interfaces
- Supports a wide range of protocols and peripherals including:
  - UART
  - I2C
  - SPI
  - I2S
- Creates an interlink between the GPIO method of software emulation and the exact hardware peripheral module

3 Hardware and software requirements

This document describes the example application based on the NXP FRDM-MCXC242 board. The basic concept can be easily implemented on the customized hardware as well.

The example shows the UART communication using the FlexIO module with the following configuration:

- Data: 8 bits
- Stop: 1 bit
- Parity: None
- Hardware flow control: No

Figure 1 shows the UART data frame emulated by FlexIO.

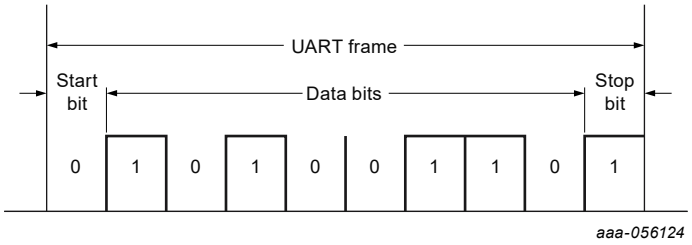

Figure 1. UART 8-bit data frame

4 UART overview

A universal asynchronous receiver/transmitter is a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used with communication standards such as EIA, RS-232, RS-422, or RS-485. The universal designation indicates that the data format and transmission speeds are configurable. A driver circuit external to the UART handles the electric signaling levels and methods, such as differential signaling. A UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port.

Transmitting and receiving UARTs must be set for the same bit speed, character length, parity, and stop bits for proper operation. The receiving UART can detect some mismatched settings and set a "framing error" flag bit for the host system. In exceptional cases, the receiving UART produces an erratic stream of mutilated characters and transfers them to the host system.

Typical serial ports used with personal computers connected to modems use the following configuration:

- Data: 8 bits
- Parity: None
- Stop: 1 bit

For this configuration, the number of ASCII characters per second equals the bit rate divided by 10.

5 UART emulation by FlexIO module

This section explains UART emulation by the FlexIO module.

- UART bus can be supported by using two timers, two shifters, and two pins:
  - The transmitter is supported by using one timer, one shifter, and one pin.
  - The receiver is supported by using one timer, one shifter, and one pin.
    Both transmitter and receiver parts can be used independently.
- The FlexIO peripheral automatically handles the start and stop bit insertion.
- The maximum baud rate of the emulated peripheral is 115,200 baud.
- The software implementation allows use of UART in interrupt or polling modes.
- Break and idle characters require software intervention and are not implemented in the example application.
- Configurable bit order (bit swapped buffer MSB first), and multiple transfers can be supported using the DMA controller.
- FlexIO module does not allow automatic insertion of parity bits.

Figure 2 shows the internal connection of FlexIO emulated UART.

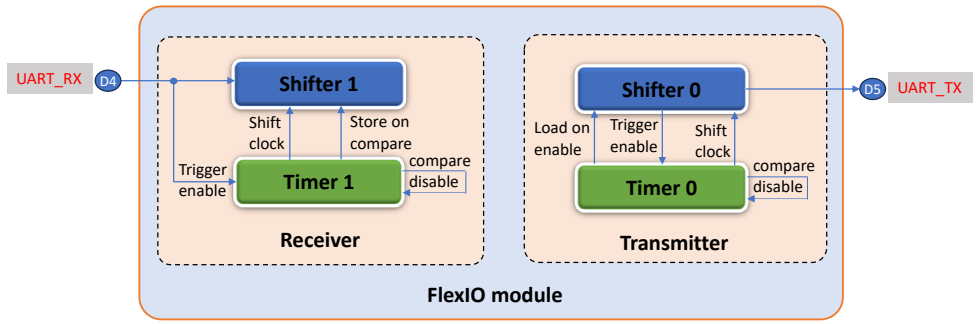

Figure 2. FlexIO emulated UART

5.1 Transmitter

Transmitting procedure includes the following steps:

1. Set the shifter to Transmit mode
2. Shift loaded data from the shifter buffer
3. Shift the data to the pin output
4. Start and stop bits are automatically loaded before or after data
5. Use the timer status flag to send the next data frame

Figure 3 shows the principle of UART transmitter emulation. The timer status flag is checked in the polling mode and the module generates an interrupt when the interrupt setting is enabled.

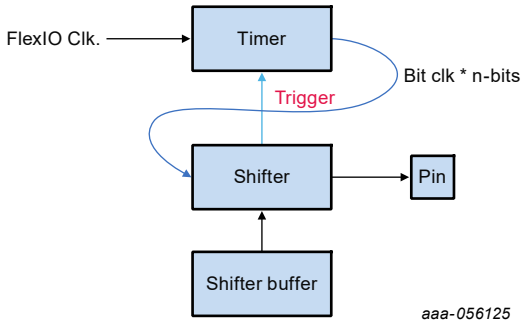

Figure 3. Block diagram of the UART transmitter on FlexIO module

5.2 Receiver

Receiving procedure includes the following steps:

1. Set the shifter to Receiver mode
2. The data is shifted in when the store event is signaled
3. The status flag indicates when data can be read (generate interrupt)
4. Wait for the shifter status flag in Polling mode
5. Store into the shifter buffer
6. Reading bit swaps the shifter buffer (without any logical operation)

Figure 4 shows the principle of UART receiver emulation. The shifter status flag is checked in the Polling mode and the module generates an interrupt when the interrupt setting is enabled.

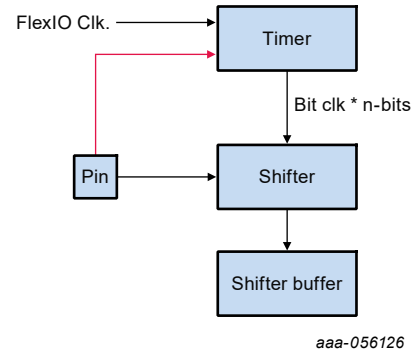

Figure 4. Block diagram of the UART receiver on FlexIO module

6 UART with IRDA encoding and decoding by FlexIO module

UART data is in NRZ format. To encode this data into the IRDA protocol, one FlexIO timer in dual 8-bit counter PWM mode is required to modulate the NRZ data. To receive the IRDA signal, one FlexIO timer in dual 8-bit counter baud/bit mode is required to decode the IRDA signal into NRZ format.

Figure 5 shows this use case diagram.

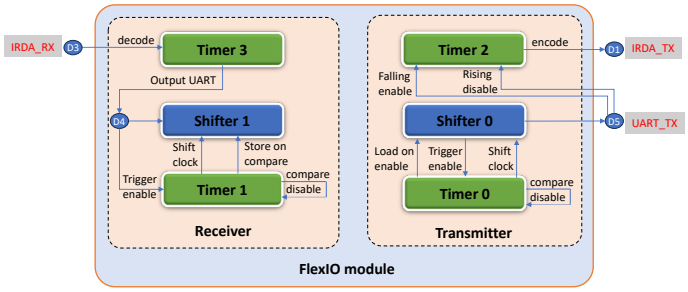

Figure 5. FlexIO UART with IRDA encode/decode

Figure 6 shows the waveform.

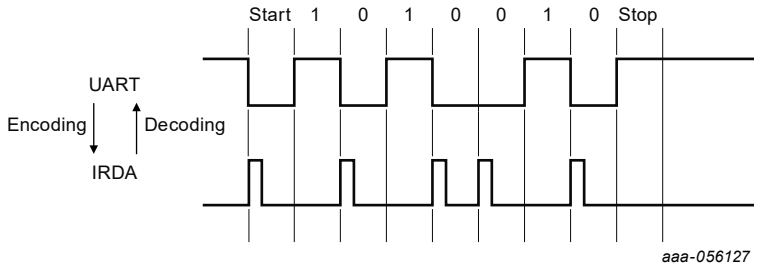

Figure 6. FlexIO UART and IRDA waveform

6.1 IRDA encoding timer configuration

The encoding timer is configured to trigger by the UART NRZ data falling edge, which represents the initial edge for the start signal.

The following lists the details of the timer configuration:

- Timer Control Register (FLEXIO_TIMCTLn):
  - TRGSEL: Select FlexIO UART TX output pin.

  - TRGPOL: Select trigger polarity active low.
  - TRGSRC: Select the trigger source internal.
  - PINCFG: Select timer pin output.
  - PINSEL: Select a timer pin to use. Select a pin, which is not used by FlexIO UART TX.
  - PINPOL: Select timer output polarity. It can be high true or low true depending on the external IRDA device used.
  - TIMOD: Select timer running mode, dual 8-bit counter PWM high mode.
- Timer Configuration Register (FLEXIO_TIMCFGn):
  - TIMOUT: Select timer output one, not affected by timer reset
  - TIMDEC: Select a timer decrease on the FlexIO clock
  - TIMRST: Select timer never reset
  - TIMDIS: Select timer disabled on trigger falling edge
  - TIMENA: Select timer enabled on trigger rising edge
  - TSTOP: Disable
  - TSTART: Disable
- Timer Compare Register (FLEXIO_TIMCMPn):
  - CMP: Timer value to set.
    In dual 8-bit counter PWM mode:
    - The lower 8 bits configure the high period output = (CMP[7:0] + 1).
    - The upper 8 bits configure the low period output = (CMP[15:8] + 1).

    For IRDA, the high period must be 3/16 UART period time.

6.2 IRDA decoding time configuration

The decoding timer is configured to trigger by the IRDA data rising edge, which represents the initial edge for the start signal.

The following lists the details of the timer configuration:

- Timer Control Register (FLEXIO_TIMCTLn):
  - TRGSEL: Select FlexIO pin input.
  - TRGPOL: Select trigger polarity active high, the IRDA receiver output signal polarity.
  - TRGSRC: Select trigger source internal depending on the TRGSEL configuration.
  - PINCFG: Select timer pin output enable.
  - PINSEL: Select a timer pin to use.
  - PINPOL: Select timer output polarity; active low.
  - TIMOD: Select timer running mode; single 16-bit counter mode.
- Timer Configuration Register (FLEXIO_TIMCFGn):
  - TIMOUT: Select timer output logic one when enabled and on timer reset.
  - TIMDEC: Select a timer decrease on the FlexIO clock.
  - TIMRST: Select timer reset on timer trigger rising edge.
  - TIMDIS: Select timer disabled on timer compared.
  - TIMENA: Select timer enabled on trigger rising edge.
  - TSTOP: Select timer stop disabled.
  - TSTART: Select the timer start bit disabled.
- Timer Compare Register (FLEXIO_TIMCMPn):
  - CMP: Timer value to set.
    The time is running in dual 8-bit counter baud/bit mode.
    In 16-bit counter mode, the compare value can be used to generate the baud rate divider = (CMP[15:0] + 1) * 2.

7 Software implementation

This section describes the configuration of FlexIO for UART and IRDA. The MCUXpresso configuration tool generates the code. For better understanding, refer to the corresponding registers section in the *MCX C24X Sub-Family Reference Manual* (document MCXC242RM).

7.1 FlexIO UART driver initialization

The register configuration is as follows:

```
/* Definitions for FlexIO_UART_Init functional group */
/* FLEXIO_CTRL: DOZEN=1, DBGE=1, FASTACC=0, FLEXEN=1 */
#define FLEXIO_UART_INIT_CTRL_INIT 0xC0000001U
/* FLEXIO_SHIFTSIEN: SSIE=0 */
#define FLEXIO_UART_INIT_SHIFTSIEN_INIT 0x0U
/* FLEXIO_SHIFTEIEN: SEIE=0 */
#define FLEXIO_UART_INIT_SHIFTEIEN_INIT 0x0U
/* FLEXIO_TIMIEN: TEIE=0 */
#define FLEXIO_UART_INIT_TIMIEN_INIT 0x0U
/* FLEXIO_SHIFTSDEN: SSDE=0 */
#define FLEXIO_UART_INIT_SHIFTSDEN_INIT 0x0U
/* FLEXIO_SHIFTCTL0: TIMSEL=0, TIMPOL=0, PINCFG=3, PINSEL=5, PINPOL=0, SMOD=2 */
#define FLEXIO_UART_INIT_SHIFTCTL0_INIT 0x30502U
/* FLEXIO_SHIFTCTL1: TIMSEL=1, TIMPOL=1, PINSEL=4, PINPOL=0, SMOD=1 */
#define FLEXIO_UART_INIT_SHIFTCTL1_INIT 0x1800401U
/* FLEXIO_SHIFTCFG0: INSRC=0, SSTOP=3, SSTART=2 */
#define FLEXIO_UART_INIT_SHIFTCFG0_INIT 0x32U
/* FLEXIO_SHIFTCFG1: INSRC=0, SSTOP=3, SSTART=2 */
#define FLEXIO_UART_INIT_SHIFTCFG1_INIT 0x32U
/* FLEXIO_TIMCTL0: TRGSEL=1, TRGPOL=1, TRGSRC=1, PINCFG=0, PINSEL=5, PINPOL=0,
 TIMOD=1 */
#define FLEXIO_UART_INIT_TIMCTL0_INIT 0x1C00501U
/* FLEXIO_TIMCTL1: TRGSEL=8, TRGPOL=1, TRGSRC=1, PINCFG=0, PINSEL=0, PINPOL=0,
 TIMOD=1 */
#define FLEXIO_UART_INIT_TIMCTL1_INIT 0x8C00001U
/* FLEXIO_TIMCFG0: TIMOUT=0, TIMDEC=0, TIMRST=0, TIMDIS=2, TIMENA=2, TSTOP=2,
 TSTART=1 */
#define FLEXIO_UART_INIT_TIMCFG0_INIT 0x2222U
/* FLEXIO_TIMCFG1: TIMOUT=2, TIMDEC=0, TIMRST=6, TIMDIS=2, TIMENA=6, TSTOP=2,
 TSTART=1 */
#define FLEXIO_UART_INIT_TIMCFG1_INIT 0x2062622U
/* FLEXIO_TIMCMP0: CMP=3874 */
#define FLEXIO_UART_INIT_TIMCMP0_INIT 0xF22U
/* FLEXIO_TIMCMP1: CMP=3874 */
#define FLEXIO_UART_INIT_TIMCMP1_INIT 0xF22U
```

7.2 FlexIO IRDA driver initialization

The register configuration is as follows:

```
/* Definitions for FlexIO_UART_IRDA_Init functional group */
/* FLEXIO_CTRL: DOZEN=1, DBGE=1, FASTACC=0, FLEXEN=1 */
#define FLEXIO_UART_IRDA_INIT_CTRL_INIT 0xC0000001U
/* FLEXIO_SHIFTSIEN: SSIE=0 */
#define FLEXIO_UART_IRDA_INIT_SHIFTSIEN_INIT 0x0U
/* FLEXIO_SHIFTEIEN: SEIE=0 */
```

```
#define FLEXIO_UART_IRDA_INIT_SHIFTEIEN_INIT 0x0U
/* FLEXIO_TIMIEN: TEIE=0 */
#define FLEXIO_UART_IRDA_INIT_TIMIEN_INIT 0x0U
/* FLEXIO_SHIFTSDEN: SSDE=0 */
#define FLEXIO_UART_IRDA_INIT_SHIFTSDEN_INIT 0x0U
/* FLEXIO_SHIFTCTL0: TIMSEL=0, TIMPOL=0, PINCFG=3, PINSEL=5, PINPOL=0, SMOD=2 */
#define FLEXIO_UART_IRDA_INIT_SHIFTCTL0_INIT 0x30502U
/* FLEXIO_SHIFTCTL1: TIMSEL=1, TIMPOL=1, PINSEL=4, PINPOL=0, SMOD=1 */
#define FLEXIO_UART_IRDA_INIT_SHIFTCTL1_INIT 0x1800401U
/* FLEXIO_SHIFTCFG0: INSRC=0, SSTOP=3, SSTART=2 */
#define FLEXIO_UART_IRDA_INIT_SHIFTCFG0_INIT 0x32U
/* FLEXIO_SHIFTCFG1: INSRC=0, SSTOP=3, SSTART=2 */
#define FLEXIO_UART_IRDA_INIT_SHIFTCFG1_INIT 0x32U
/* FLEXIO_TIMCTL0: TRGSEL=1, TRGPOL=1, TRGSRC=1, PINCFG=0, PINSEL=5, PINPOL=0,
 TIMOD=1 */
#define FLEXIO_UART_IRDA_INIT_TIMCTL0_INIT 0x1C00501U
/* FLEXIO_TIMCTL1: TRGSEL=8, TRGPOL=1, TRGSRC=1, PINCFG=0, PINSEL=0, PINPOL=0,
 TIMOD=1 */
#define FLEXIO_UART_IRDA_INIT_TIMCTL1_INIT 0x8C00001U
/* FLEXIO_TIMCTL2: TRGSEL=10, TRGPOL=1, TRGSRC=1, PINCFG=3, PINSEL=1, PINPOL=0,
 TIMOD=2 */
#define FLEXIO_UART_IRDA_INIT_TIMCTL2_INIT 0xAC30102U
/* FLEXIO_TIMCTL3: TRGSEL=6, TRGPOL=0, TRGSRC=1, PINCFG=3, PINSEL=4, PINPOL=1,
 TIMOD=3 */
#define FLEXIO_UART_IRDA_INIT_TIMCTL3_INIT 0x6430483U
/* FLEXIO_TIMCFG0: TIMOUT=0, TIMDEC=0, TIMRST=0, TIMDIS=2, TIMENA=2, TSTOP=2,
 TSTART=1 */
#define FLEXIO_UART_IRDA_INIT_TIMCFG0_INIT 0x2222U
/* FLEXIO_TIMCFG1: TIMOUT=2, TIMDEC=0, TIMRST=6, TIMDIS=2, TIMENA=6, TSTOP=2,
 TSTART=1 */
#define FLEXIO_UART_IRDA_INIT_TIMCFG1_INIT 0x2062622U
/* FLEXIO_TIMCFG2: TIMOUT=0, TIMDEC=0, TIMRST=0, TIMDIS=6, TIMENA=6, TSTOP=0,
 TSTART=0 */
#define FLEXIO_UART_IRDA_INIT_TIMCFG2_INIT 0x6600U
/* FLEXIO_TIMCFG3: TIMOUT=2, TIMDEC=0, TIMRST=6, TIMDIS=2, TIMENA=6, TSTOP=0,
 TSTART=0 */
#define FLEXIO_UART_IRDA_INIT_TIMCFG3_INIT 0x2062600U
/* FLEXIO_TIMCMP0: CMP=3874 */
#define FLEXIO_UART_IRDA_INIT_TIMCMP0_INIT 0xF22U
/* FLEXIO_TIMCMP1: CMP=3874 */
#define FLEXIO_UART_IRDA_INIT_TIMCMP1_INIT 0xF22U
/* FLEXIO_TIMCMP2: CMP=14348 */
#define FLEXIO_UART_IRDA_INIT_TIMCMP2_INIT 0x380CU
/* FLEXIO_TIMCMP3: CMP=69 */
#define FLEXIO_UART_IRDA_INIT_TIMCMP3_INIT 0x45U
```

8 Demos

This section introduces the following two demos:

- FlexIO UART demo
- FlexIO UART with IRDA demo

The MCUXpresso configuration tool generates the code.

8.1 FlexIO UART demo

In this demo, a FlexIO simulated UART is connected to the PC through USB-serial. The board returns all characters that the PC sends to the board.

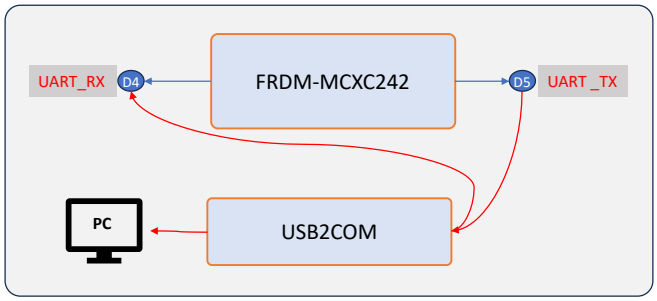

Figure 7. FlexIO UART demo

8.1.1 Software configuration

For software connection, enable FlexIO UART define as follows:

```
#define FLEXIO_UART//FLEXIO_UART_IRDA//
#ifdef FLEXIO_UART

 /* Initialize components */
    FlexIO_UART_InitPins();
    FlexIO_UART_Init();

    uartDev.flexioBase      = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex      = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex      = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0]   = 0U;
    uartDev.timerIndex[1]   = 1U;

#endif
```

8.1.2 Hardware connection

For hardware connection, perform the following steps:

1. Plug one USB Type-C cable into the J9 to power the FRDM-MCXC242 board.
2. Connect one USB to the UART board with UART pins:
   - J1-9, TX of USB2COM connected
   - J1-11, RX of USB2COM connected
   - J2-14, Ground of USB2COM connected
3. For a serial device, open a serial terminal on a PC with the following settings:
   - Baud rate = 115200
   - Data = 8 bits
   - Parity = None
   - Stop = 1 bit
   - Flow control = No
4. Download the program to the target board.
5. To run the demo, either press the reset button on the board or launch the debugger in the IDE.

8.1.3 Demo result

When the demo runs successfully, the log appears on the UART Terminal port connected to the USB2COM, as shown in Figure 8.

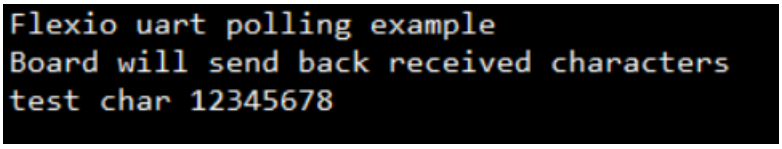

Figure 8. Terminal port

The board returns the characters sent from the PC UART tool.

Figure 9 shows the timing capture in the logic device.

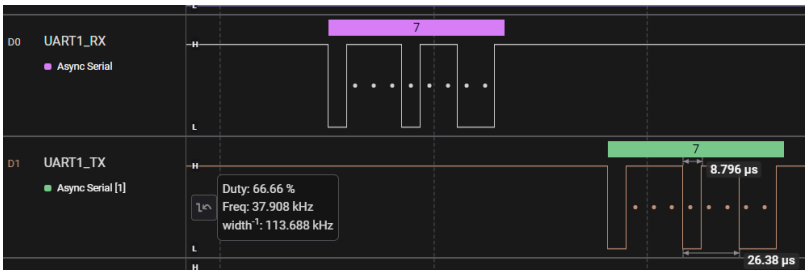

Figure 9. Timing capture

8.2 FlexIO UART with IRDA demo

In this demo, one FRDM-MCXC242 simulates the IRDA encoder and the other FRDM-MCXC242 stimulates the IRDA decoder. Connect IRDA_TX pin of one board with IRDA_RX pin of another board. For debugging, the data received by IRDA_RX is sent to USB2COM through pin UART_TX.

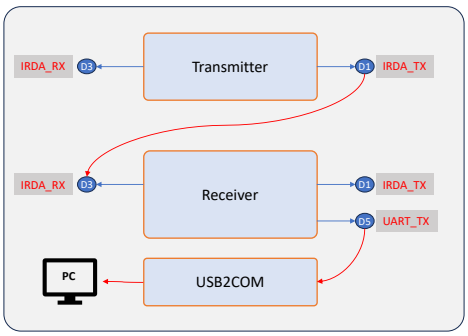

Figure 10. FlexIO UART with IRDA demo

8.2.1 Software configuration

For software connection, perform the following steps:

1. Enable FlexIO UART IRDA define as follows:

```
#define FLEXIO_UART_IRDA//FLEXIO_UART//
#ifdef FLEXIO_UART_IRDA

    /* Initialize components */
    FlexIO_UART_IRDA_InitPins();
    FlexIO_UART_IRDA_Init();
```

```
    uartDev.flexioBase      = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex      = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex      = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0]   = 0U;
    uartDev.timerIndex[1]   = 1U;

#endif
```

2. Code for transmitter is as follows:

```
#define TRANSMITTER//RECEIVER//
#ifdef  TRANSMITTER
    ch = 0x30;
    while (1)
    {
     ch++;
        FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
        if(ch == 0x5A)
        {
            ch = 0x30;
        }
        SDK_DelayAtLeastUs(10000, 48000000);
    }
#endif
```

3. Code for receiver is as follows:

```
#define RECEIVER//TRANSMITTER//
#ifdef  RECEIVER
    FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff) - 1);
    while (1)
    {
       FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
        FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
    }
#endif
```

8.2.2 Hardware connection

For hardware connection, perform the following steps:

1. Plug one USB Type-C cable into the J9 to power the FRDM-MCXC242 board.
2. Connect the pin IRDA_TX of the transmitter to the pin IRDA_RX of the receiver.
3. Connect one USB to the UART board with pin UART_TX of the receiver board:
   - J1-9, TX of USB2COM connected
   - J2-14, Ground of USB2COM connected
4. For a serial device, open a serial terminal on a PC with the following settings:
   - Baud rate = 115200
   - Data = 8 bits
   - Parity = None
   - Stop = 1 bit
   - Flow control = No
5. Download the program to the target board.
6. To run the demo, either press the reset button on the board or launch the debugger in the IDE.

8.2.3 Demo result

When the demo runs successfully, the log appears on the UART Terminal port connected to the USB2COM, as shown in Figure 11.

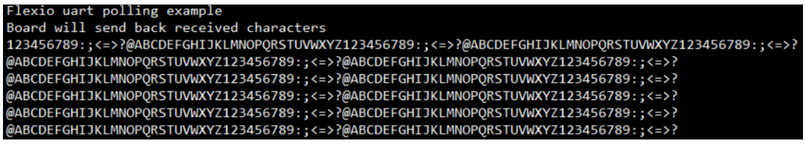

Figure 11. Terminal port

The characters sent by the transmitter appear on the UART Terminal port.

Figure 12 shows the timing of signals.

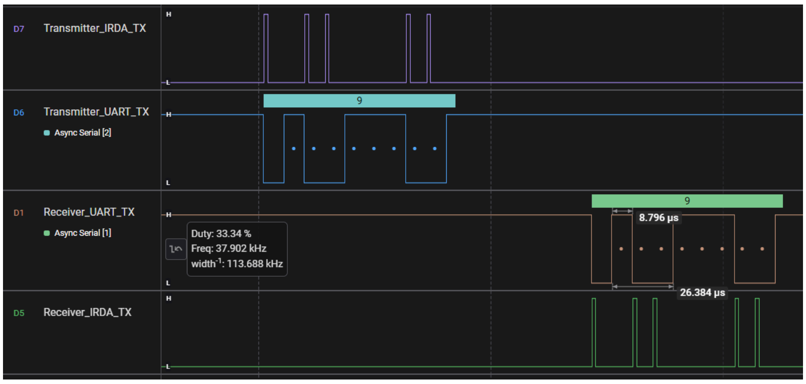

Figure 12. UART with IRDA signals timing

9 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN

ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Revision history

Table 1 summarizes revisions to this document.

Table 1. Revision history

Document ID	Release date	Description
AN14319 v.1.0	15 July 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Freescale — is a trademark of NXP B.V.

MCX — is a trademark of NXP B.V.

Tower — is a trademark of NXP B.V.

Contents