AN14458

How to reach LPUART maximum band rate of 24 Mbit/s on MCX A series

Rev. 1.0 — 17 October 2024 Application note

Document information

Information	Content
Keywords	AN14458, MCX A series, LPUART, 24 Mbit/s, Trim Clock
Abstract	This application note describes how to reach LPUART maximum band rate of 24 Mbit/s on MCX A series and test on FRDM-MCXA153 board.

1 Introduction

The MCX A series microcontrollers powered by the Arm Cortex-M33 are general-purpose MCUs designed to address a wide range of applications with scalable device options, low power, and intelligent peripherals. They offer the Low-Power Universal Asynchronous Receiver/Transmitter (LPUART). It provides an asynchronous serial bus with commander and responder operations. MCXA152/3 supports 3 instances of LPUART and MCXA154/5/6 supports 5 instances of LPUART. Usually, the microcontroller communicates with the host computer through UART and prints debug logs. In this use case, the most common baud rates are 9600 bit/s and 115200 bit/s, it is an easy task for a microcontroller. However, in some use cases, a high baud rate LPUART is needed, and for MCX A series, the maximum baud rate of 24 Mbit/s is supported. This application note introduces the LPUART module in MCX A series and describes how to reach to LPUART maximum baud rate of 24 Mbit/s in detail:

- How to set up the LPUART to get the 24Mbit/s baud rate
- How to implement LPUART send, receive and check
- How to calculate the transmission error data rate
- How to output the FRO_HF clock
- How to trim the FRO_HF clock manually
- How to automatically trim the FRO_HF clock using an external crystal

2 LPUART overview

LPUART provides an asynchronous serial bus with commander and responder operations. It can reach a 24 Mbit/s maximum transfer rate. The data is based on characterization but not covered by the test limits in production.

2.1 Features

- Full-duplex, standard NRZ format
- Programmable baud rates (13-bit modulo divider) with a configurable oversampling ratio (OSR) from 4× to 32×, up to 24 Mbit/s.
- Asynchronous operation of transmit and receive baud rates regarding the bus clock:
  - The baud rate can be configured independently of the bus clock frequency.
  - Operation in Low-Power modes is supported.
- Support interrupt, DMA, or polled operations:
  - Transmit data empty and transmission complete
  - Receive data full
  - Receive overrun error, parity error, framing error, and noise error
  - Idle receiver detect
  - Active edge on receive pin
  - Break detect supporting LIN
  - Receive data match
- Hardware parity generation and checking
- Programmable 7-bit, 8-bit, 9-bit, or 10-bit data bits
- Programmable 1-bit or 2-bit stop bits
- Support for three receiver wake-up methods:
  - Idle line wake-up
  - Address mark wake-up

  - Receive data match
- Automatic address matching to reduce ISR overhead
  - Address mark matching
  - Idle line address matching
  - Address match start, address match end
- Optional 13-bit and 11-bit break character generation
- Configurable idle length detection supporting 1, 2, 4, 8, 16, 32, 64, or 128 idle characters
- Selectable transmitter output and receiver input polarity
- Hardware flow control support for request to send (RTS) and clear to send (CTS) signals
- Selectable IrDA 1.4 return-to-zero-inverted (RZI) format with a programmable pulse width
- Independent FIFO structure for transmit and receive functions:
  - Separate configurable watermarks for receive and transmit requests
  - An option for the receiver to assert the request after a configurable number of idle characters, if receive FIFO is not empty.

2.2 Function

Figure 1 shows the transmitter portion of LPUART and Figure 2 shows the receiver portion of LPUART.

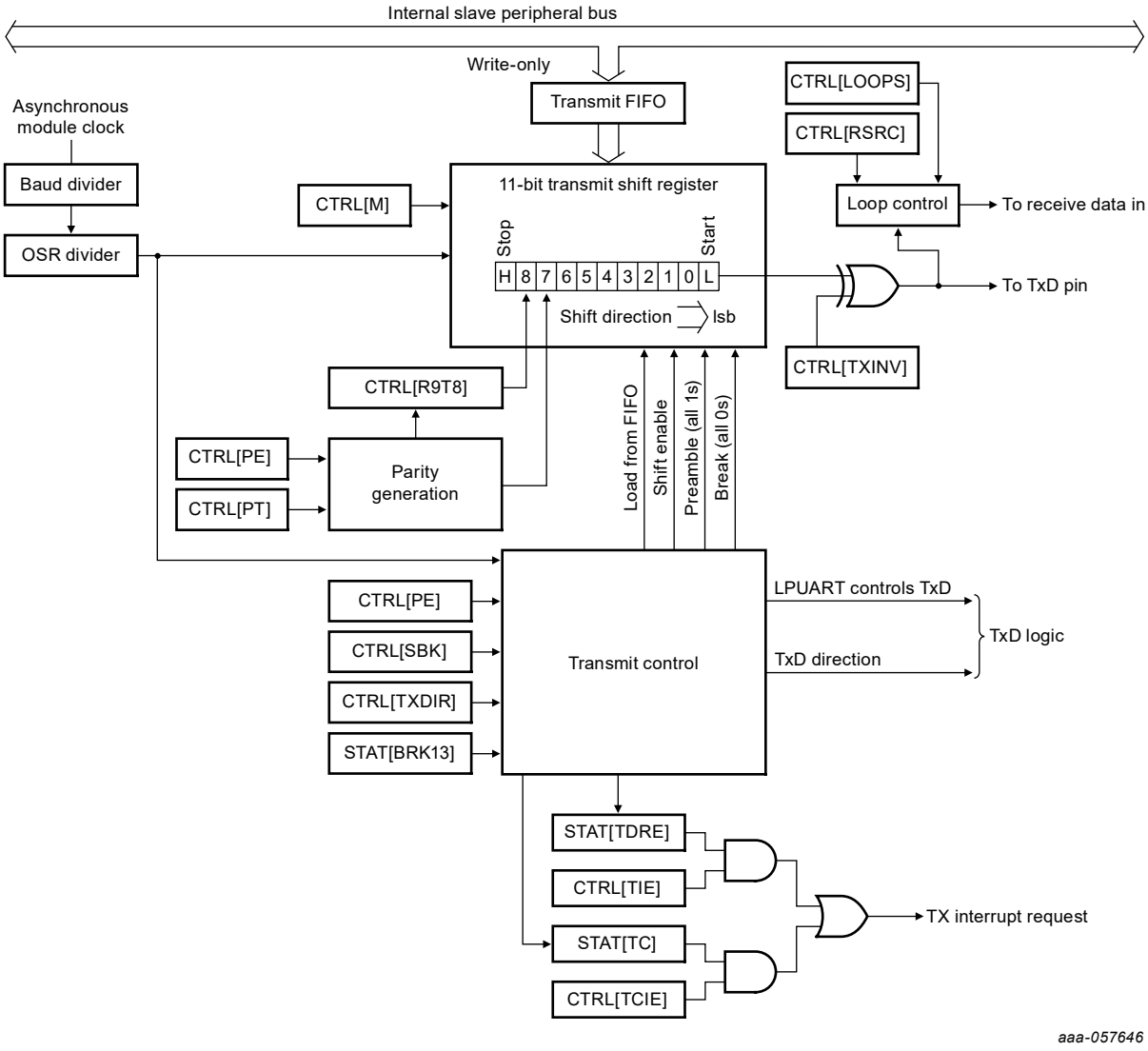

Figure 1. Transmitter block diagram

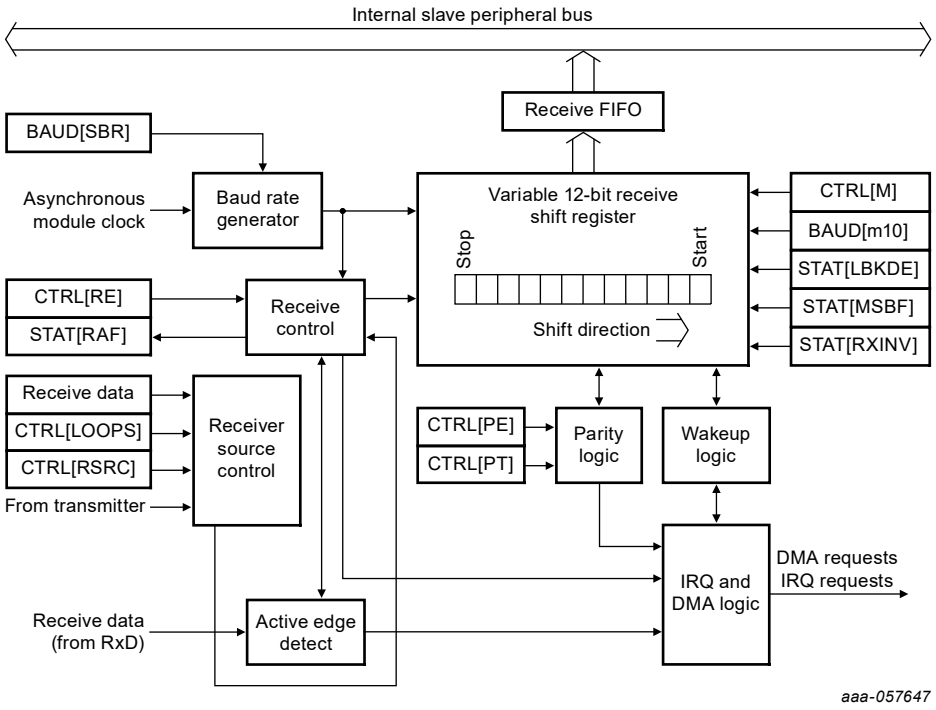

Figure 2. Receiver block diagram

2.2.1 Baud rate generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and transmitter. The value ranging from 1 to 8191 and written to BAUD[SBR] determines the baud clock divisor for the asynchronous LPUART baud clock. The baud rate clock drives the receiver, while a bit clock generated from the baud rate clock divided by the OSR drives the transmitter. Depending on the OSR, the receiver has an acquisition rate of 4 to 32 samples per bit time.

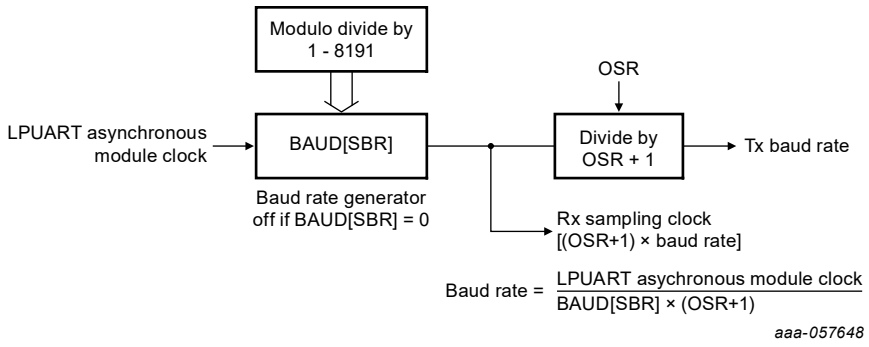

Figure 3. Baud rate generation

To reach the 24 Mbit/s maximum baud rate, set FRO_HF 96 MHz as the LPUART asynchronous module clock, BAUD[SBR] set to 1, OSR + 1 set to 4, and the baud rate will be 96 MHz / 4 = 24 Mbit/s.

2.2.2 Baud rate tolerance

A transmitting device may operate at a baud rate below or above that of the receiver.

Accumulated bit time misalignment can cause one of the three stop bit data samples to fall outside the actual stop bit. A noise error occurs if the three samples are not all the same logical values. A framing error occurs if the receiver clock is misaligned in such a way that the majority of the three stop bit samples are a logic zero.

As the receiver samples an incoming frame, it may re-synchronize the oversampling clock on any valid falling edge within the frame. Resynchronization within frames corrects a misalignment between transmitter bit times and receiver bit times.

In general, increasing the number of samples per bit increases the baud rate tolerance and decreasing the number of samples per bit reduces the baud rate tolerance.

Note: *Since LPUART implements triple voting on consecutive receive data samples, increasing the number of samples per bit moves those samples closer together that would reduce the width of noise that can be filtered by the triple voting logic.*

2.2.3 Calculating baud rate tolerance

Using the following definitions:

- SAM is the number of sample points per bit (valid range from 8 to 32; equal to (OSR + 1) × (BOTHEDGE + 1)).
- BIT is the number of bits in a character including start, data, and stop bits (valid range from 9 to 13).

The ideal baud rate tolerance can be calculated as follows:

- Slow data rate tolerance = ((SAM ÷ 2) -1) ÷ ((SAM × BIT) - (SAM ÷ 2) + 2)
- Fast data rate tolerance = ((SAM ÷ 2) - 2) ÷ (SAM × BIT)

In this example, OSR + 1 = 4, BOTHEDGE = 1, SAM = 4*2 = 8, BIT = 10 (1 start bit, 8-bit data, 1 stop bit):

- Slow data rate tolerance = (4 - 1) ÷ (80 - 4 + 2) = 3 ÷ 78 = 3.85 %
- Fast data rate tolerance = (4 - 2) ÷ 80 = 2 ÷ 80 = 2.5 %

3 Software design

The software is based on FRDM-MCXA153 SDK 2.16.0 `driver_examples/lpuart/lpuart_edma_transfer` that is an LPUART transfer example using eDMA. The user must be familiar with this example and the related hardware.

This software uses one FRDM-MCXA153 as the Sender, uses another FRDM-MCXA153 as the Receiver, the Sender sends the data to the Receiver using LPUART @24Mbps baud rate, and then receives the data replied by the Receiver and calculates the error data rate.

3.1 Clock generation

To set the LPUART baud rate to 24 Mbit/s, use FRO_HF 96 MHz as the LPUART asynchronous module clock, and divided it to 24 MHz to reach the 24 Mbit/s LPUART baud rate, refer to Section 2.2.1 for more detail.

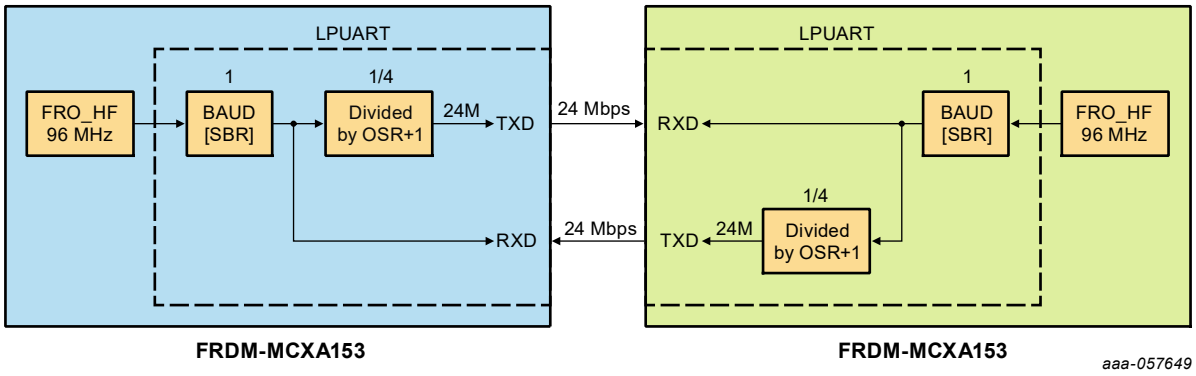

Figure 4. Clock generation

The core code for the clock generation is as follows:

```
CLOCK_SetupFROHFClocking(96000000U);   /* !< Enable FRO HF(96MHz) output */
/*!< Set up dividers */
CLOCK_SetClockDiv(kCLOCK_DivFRO_HF_DIV, 1U); /* !< Set FROHFDIV divider to value 1 */

/* attach 96 MHz clock to FLEXCOMM2, use LPUART2 to test the 24Mbps baud rate */
CLOCK_SetClockDiv(kCLOCK_DivLPUART2, 1u);
CLOCK_AttachClk(kFRO_HF_DIV_to_LPUART2);
RESET_PeripheralReset(kLPUART2_RST_SHIFT_RSTn);
```

3.2 Transfer process

To verify the reliability of the transmission, the following transfer process is used for test, where one MCXA153 is used as the Sender and another MCXA153 is used as the Receiver.

1. The Sender sends the start signal. The Receiver receives the start signal and records.
2. The Receiver replies to the start signal to the Sender, the Sender checks whether it is correct, if correct, connect success.
3. The Sender starts to send the test data, the Receiver receives them and records.
4. The Receiver replies the test data to the Sender. The Sender checks them and returns to step 3.

Note:

1. *start signal: 0x12, 0x21; test data: from 0x0000 to 0x07FF total 4096 bytes.*
2. *The Sender should be reset after the Receiver.*
3. *Other MCX A series like MCXA156 are also supported and there is no difference. Use MCXA153 as an example.*

And the user can understand the whole process more clearly through the figure below:

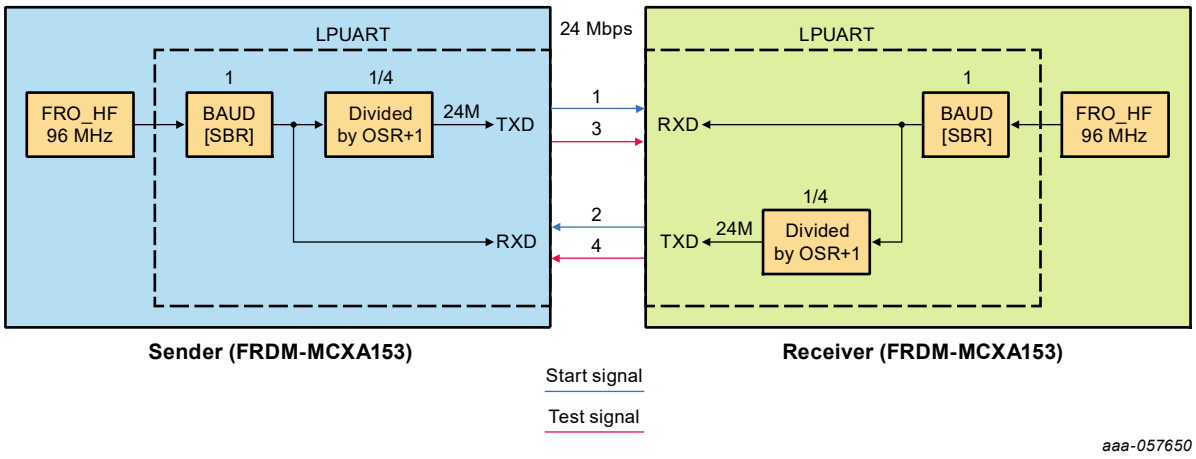

Figure 5. Transfer process

To send/receive the data through LPUART, refer to the below code for reference. The core code for the Sender to send, receive, and check the start signal is as follows:

```
uint8_t g_startBuffer[2] = {0x12, 0x21};  // Start signal
uint8_t g_rxBuffer[2]    = {0};           // RX buffer

/* LPUART eDMA handle define */
lpuart_edma_handle_t g_lpuartEdmaHandle;

/* LPUART transfer and receive structure define */
lpuart_transfer_t xfer;
lpuart_transfer_t sendXfer;
lpuart_transfer_t receiveXfer;

/* Send communication start signal */
xfer.data     = g_startBuffer;
xfer.dataSize = 2;
txOnGoing     = true;    // See SDK example for details
LPUART_SendEDMA(LPUART2, &g_lpuartEdmaHandle, &xfer);
PRINTF("FRDM-MCXA153 LPUART 24Mbps Demo, this is the sender\r\n");
PRINTF("Sending the communication start signal...\r\n");

/* Wait send finished */
while (txOnGoing)   // See SDK example for details
{
}
PRINTF("Send Done\r\n");

PRINTF("Receiving the communication start signal...\r\n");
/* Receive communication start signal */
receiveXfer.data = g_rxBuffer;
receiveXfer.dataSize = 2;
rxOnGoing = true;
LPUART_ReceiveEDMA(LPUART2, &g_lpuartEdmaHandle, &receiveXfer);
while(rxOnGoing)   // See SDK example for details
{
}
PRINTF("Receive Done\r\n");

PRINTF("Received the connect success signal\r\n");
if((receiveXfer.data[0] == g_startBuffer[0]) && (receiveXfer.data[1] ==
 g_startBuffer[1]))
{
  connectSuccess = true;
  PRINTF("Connect Success\r\n");
}
```

```
else
{
  PRINTF("Connect Fail\r\n");
}
```

The core code for the Sender to send, receive, and check the test data is as follows:

```
uint16_t g_txBuffer16[2048] = {0};   // Send test data
uint16_t g_rxBuffer16[2048] = {0};   // Receive test data

/* Set values for the test data */
for(uint16_t i = 0; i < 2048U; i++)
{
  g_txBuffer16[i] = i;
}

PRINTF("Start to do test...\r\n");
sendXfer.txData16    = g_txBuffer16;
sendXfer.dataSize    = 2048U * 2;
receiveXfer.rxData16 = g_rxBuffer16;
receiveXfer.dataSize = 2048U * 2;

while (1)
{
  if(connectSuccess == true && uartCheckDone == false)
  {
     txOnGoing     = true;   // See SDK example for details
     LPUART_SendEDMA(LPUART2, &g_lpuartEdmaHandle, &sendXfer);
     /* Wait send finished */
     while (txOnGoing)    // See SDK example for details
     {
     }
     PRINTF("Send Done\r\n");

     rxBufferEmpty  = true;    // See SDK example for details
     rxOnGoing = true;         // See SDK example for details
     LPUART_ReceiveEDMA(LPUART2, &g_lpuartEdmaHandle, &receiveXfer);
     while(rxOnGoing)          // See SDK example for details
     {
     }
     PRINTF("Receive Done\r\n");

     UART_Check(); // Count Transfer number and Error data number, see section 3.3
  }
}
```

The core code for the Receiver to receive, check, and reply the start signal is as follows:

```
uint8_t g_startBuffer[2] = {0x12, 0x21};  // Start signal
uint8_t g_rxBuffer[2]    = {0};           // RX buffer

/* LPUART eDMA handle define */
lpuart_edma_handle_t g_lpuartEdmaHandle;

/* LPUART transfer and receive structure define */
lpuart_transfer_t xfer;
lpuart_transfer_t receiveXfer;

PRINTF("FRDM-MCXA153 LPUART 24Mbps Demo, this is the Receiver\r\n");
PRINTF("Receiving the communication start signal...\r\n");
/* Receive communication start signal */
receiveXfer.data = g_rxBuffer;
receiveXfer.dataSize = 2;
rxOnGoing = true;   // See SDK example for details
LPUART_ReceiveEDMA(LPUART2, &g_lpuartEdmaHandle, &receiveXfer);
```

```
while(rxOnGoing)    // Wait for start signal
{
}
PRINTF("Receive Done\r\n");

if((receiveXfer.data[0] == g_startBuffer[0]) && (receiveXfer.data[1] ==
 g_startBuffer[1]))
{
  PRINTF("Connect Success\r\n");
  connectSuccess = true;    // Global variable to indicate connect success
  rxBufferEmpty    = true;  // See SDK example for details
  /* Reply communication start signal */
  xfer.data        = g_startBuffer;
  xfer.dataSize = 2;
  txOnGoing     = true;    // See SDK example for details
  LPUART_SendEDMA(LPUART2, &g_lpuartEdmaHandle, &xfer);
  PRINTF("Sending the connect success signal...\r\n");
  /* Wait send finished */
  while (txOnGoing)    // See SDK example for details
  {
  }
  PRINTF("Send done\r\n");
}
else
{
  PRINTF("Connect Fail\r\n");
}
```

The core code for the Receiver to receive and reply the test data is as follows:

```
uint16_t g_txBuffer16[2048] = {0};  // Send test data
uint16_t g_rxBuffer16[2048] = {0};  // Receive test data

PRINTF("Start to do test...\r\n");
sendXfer.txData16    = g_txBuffer16;
sendXfer.dataSize    = 2048U * 2;
receiveXfer.rxData16 = g_rxBuffer16;
receiveXfer.dataSize = 2048U * 2;

while (1)
{
 if(connectSuccess == true) // Sender and Receiver connect success
 {
    rxBufferEmpty  = true;  // See SDK example for details
    rxOnGoing = true;       // See SDK example for details
    LPUART_ReceiveEDMA(LPUART2, &g_lpuartEdmaHandle, &receiveXfer);
    while(rxOnGoing)        // See SDK example for details
    {
    }
    PRINTF("Receive Done\r\n");

    memcpy(g_txBuffer16, g_rxBuffer16, BUFFER_LENGTH * 2);
    memset(g_rxBuffer16, 0, BUFFER_LENGTH * 2);

    txOnGoing     = true;   // See SDK example for details
    LPUART_SendEDMA(LPUART2, &g_lpuartEdmaHandle, &sendXfer);
    /* Wait send finished */
    while (txOnGoing)       // See SDK example for details
    {
    }
    PRINTF("Reply Done\r\n");
 }
}
```

3.3 Error data rate calculate

To check the error data rate, follow the steps below:

1. Count the transfer number.
2. Compare the test data sent by the Sender and the test data reply by the Receiver.
3. Find the mismatch data and count the error data number.
4. Calculate the error data rate using the transfer number and the error data number. The core code for the error data rate calculate is as follows:

```
/**
 * @brief    UART_Check    UART print the transfer number and error data number
 * @param    NULL
 * @return   NULL
*/
void UART_Check()
{
  static uint32_t transferNum = 0;
  static uint32_t errorDataNum = 0;
  float  errorDataRate = 0;

  transferNum ++;              // transfer number +1 every time

  if (transferNum >= 1000)
  {
    uartCheckDone   = true;   // Stop the transfer after transfer number reach 1000
  }
  for(uint16_t i = 0; i < BUFFER_LENGTH; i++)  // Check whether the data is correct
  {
   if(g_rxBuffer16[i] != g_txBuffer16[i])
   {
    errorDataNum++;         // If not match, error data number +1
   }
 }
 errorDataRate = 1.0f * errorDataNum / transferNum; // Calculate the error data rate
 PRINTF("Transfer Num:%u Error Data Num:%u Error Data Rate:%f\r\n", transferNum,
 errorDataNum, errorDataRate);
}
```

3.4 Clock out

The user can use the clock output pin to output the clock signal and observe it. In this application note, P3_8 is used to output the clock signal. Use an oscilloscope or other device to capture the signal and calculate the clock error.

Note: *First, configure the P3_8 for the clock out function.*

The core code for clock out is as follows, a signal of about 96 MHz is captured after this setting:

```
/**
 * @brief    app_ClockOut    Clock out setting, to observe clock signal
 * @param    NULL
 * @return   NULL
*/
void app_ClockOut()
{
// Allow clock update
   SYSCON -> CLKUNLOCK             = SYSCON_CLKUNLOCK_UNLOCK(0);
// Choose FRO_HF_DIV as the source
   MRCC0  ->  MRCC_CLKOUT_CLKSEL   = MRCC_MRCC_CLKOUT_CLKSEL_MUX(1);
// Direct output
   MRCC0  ->  MRCC_CLKOUT_CLKDIV   = MRCC_MRCC_CLKOUT_CLKDIV_DIV(0);
// Freezes clock update
```

```
    SYSCON -> CLKUNLOCK             |= SYSCON_CLKUNLOCK_UNLOCK(1);
}
```

3.5 Trim the FRO_HF manually

In the MCX A series, the FRO_HF(FIRC) supports the trim function. It means the user can manually trim the clock to make it closer to the frequency needed. In this way, the user can simulate the effect of temperature and other factors on the clock. The trim register includes the coarse trim and fine trim function. For the coarse trim, the frequency change rate near the center is about 2.5% per bit. For the fine trim, the frequency change rate near the center is about 0.06% per bit. The core code for manual trim FRO_HF is as follows:

```
/**
 * @brief    app_FircTrim     Manual Trim FRO_HF clock
 * @param    type             TRIM_COARSE 2.5%, TRIM_FINE 0.06%
 *           value            FIRCTRIM + value * 2.5%(TRIM_COARSE) or 0.06%(TRIM_FINE)
 * @return   NULL
*/
void app_FircTrim(trim_type type,char value)
{
   uint32_t trim_value  = SCG0 -> FIRCTRIM;   // Record the initial trim value
   uint8_t  trim_coarse = (uint8_t)(trim_value >> 8);
   uint8_t  trim_fine   = (uint8_t)trim_value;
   PRINTF("%x\r\n", trim_value);
   /* UNLOCK the trim */
   SCG0 -> TRIM_LOCK = (SCG_TRIM_LOCK_TRIM_LOCK_KEY(0x5A5A)) |
 (SCG_TRIM_LOCK_TRIM_UNLOCK(1));
   if (type == TRIM_COARSE)  // Coarse trim mode, use coarse trim register
   {
    trim_coarse = trim_coarse + value;
    /* Change the trim register */
    SCG0 -> FIRCTRIM = (trim_value & 0xFFFF0000) | (trim_fine & 0xff) | ((trim_coarse &
 0x3f) << 8);
   }
   else if(type == TRIM_FINE) // Fine trim mode, use fine trim register
   {
    trim_fine = trim_fine + value;
    /* Change the trim register */
    SCG0 -> FIRCTRIM = (trim_value & 0xFFFF0000) | (trim_fine & 0xff)  | ((trim_coarse &
 0x3f) << 8);
   }
   trim_value  = SCG0 -> FIRCTRIM;
   PRINTF("%x\r\n", trim_value);  // Print the trim value
 }
```

3.6 Doing autotrimming using the external crystal

If a more accurate clock is needed, the autotrimming using an external crystal is supported. After doing the autotrimming, the accuracy of FRO_HF can achieve ±0.25 % (-40~125 ℃), while without autotrimming the accuracy is ±3 % (-40~125 ℃).

Note: *External crystal is needed here. In this example, an 8 MHz external crystal is connected to the MCU.*

The core code for autotrimming is as follows:

```
/**
 * @brief    app_AutoTrim   Run Auto-trimming to trim the clock using external crystal
 * @param    NULL
 * @return   NULL
*/
void app_AutoTrim()
{
  CLOCK_SetupExtClocking(8000000U);            // Enable the 8 MHz external crystal
```

```
  SCG0 -> FIRCTCFG |= SCG_FIRCTCFG_TRIMSRC(2); // Select the external crystal(SOSC) as
 the source
  SCG0 -> FIRCTCFG |= SCG_FIRCTCFG_TRIMDIV(7); // Divide the SOSC to 1 MHz
  SCG0 -> FIRCCSR  &= ~SCG_FIRCCSR_LK(1);      // Unlock the FIRCCSR register
  SCG0 -> FIRCCSR  |= SCG_FIRCCSR_FIRCTREN(1); // Enable auto trim
  SCG0 -> FIRCCSR  |= SCG_FIRCCSR_FIRCTRUP(1); // Enable update
  /* Until it returns 1, indicating FIRC is valid */
  while(!(SCG0 -> FIRCCSR & SCG_FIRCCSR_FIRCVLD_MASK));
  /* Until it returns 0, indicating no error */
  while(SCG0 -> FIRCCSR & SCG_FIRCCSR_FIRCERR_MASK);
  /* Until it returns 1, indicating FIRC auto trim locked to target frequency range */
  while(!(SCG0 -> FIRCCSR & SCG_FIRCCSR_TRIM_LOCK_MASK));
  SCG0 -> FIRCCSR  |= SCG_FIRCCSR_LK(1);        // Lock the FIRCCSR register
}
```

4 Test on board

To test the LPUART functions, the FRDM-MCXA153 board is used, as shown in Figure 6.

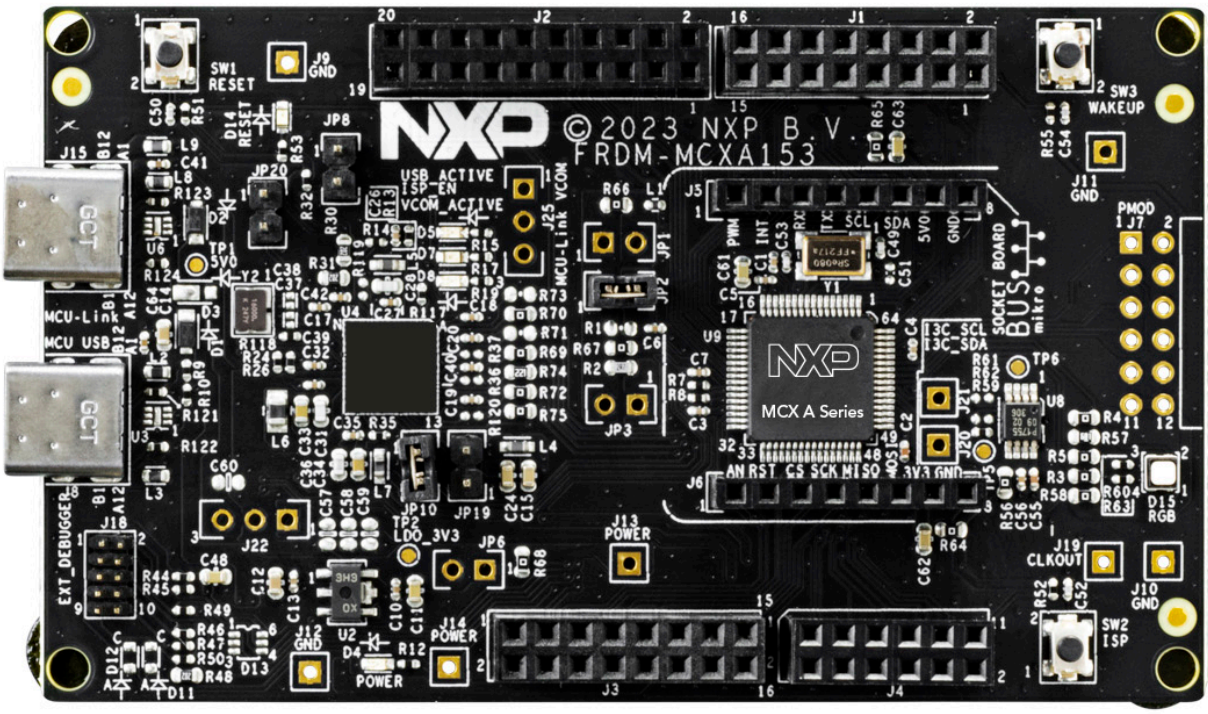

Figure 6. FRDM-MCXA153 board

4.1 Board connection

The FDRM-MCXA153 board ships with an onboard debugger. Use a USB-C cable to connect the board to the computer via J15 for downloading and debugging. In this test, 2x FRDM-MCXA153 boards are needed. The connection is shown below. The board connection is shown in Figure 7 .

- LPUART2_RXD (J1-2) ------ LPUART2_TXD (J1-4)
- LPUART2_TXD (J1-4) ------ LPUART2_RXD (J1-2)
- 5 V (J5-7) ------ 5 V (J5-7)
- GND (J5-8) ------ GND (J5-8)

Note: *The Sender must be reset after the Receiver is in the test.*

In case the two boards are powered independently, the 5 V connection is not needed.

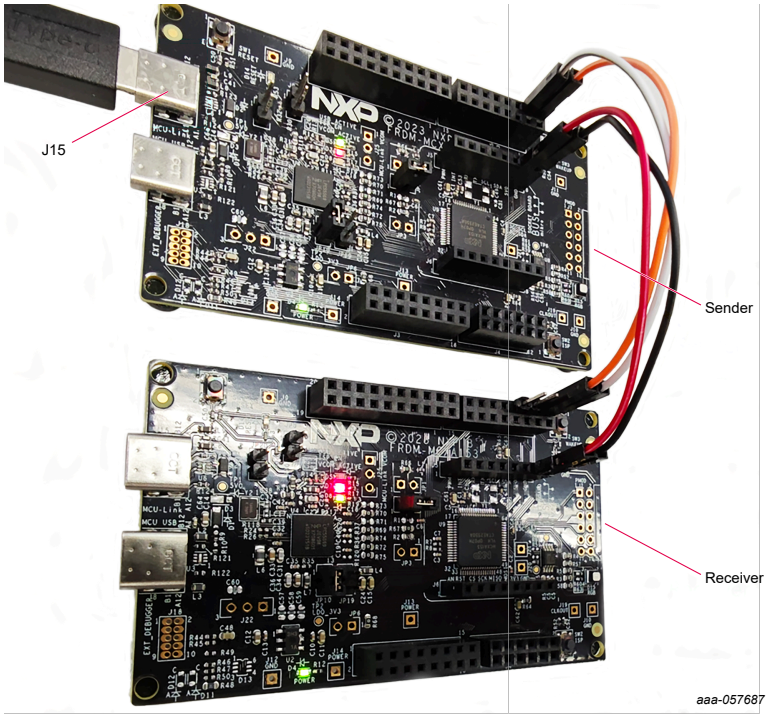

Figure 7. FRDM-MCXA153 board connection

4.2 UART debug terminal setup

To observe debug messages from the board, open a UART debug terminal. Set the terminal program to the appropriate COM port and use the setting "115200, 8, none, 1, none", as shown in Figure 8.

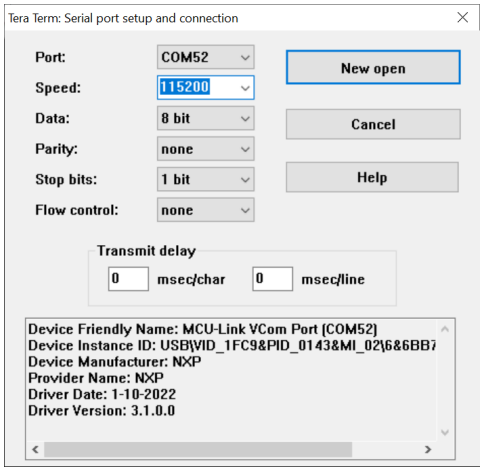

Figure 8. UART debug terminal setup

4.3 Test result

Press the RESET button on Receiver and Sender successively and the 24 Mbit/s LPUART communication will begin. The test results are printed to the UART debug terminal. The test result is shown in Figure 9. The Sender and Receiver finished 1000 cycles of test data transmission, and no error data was found.

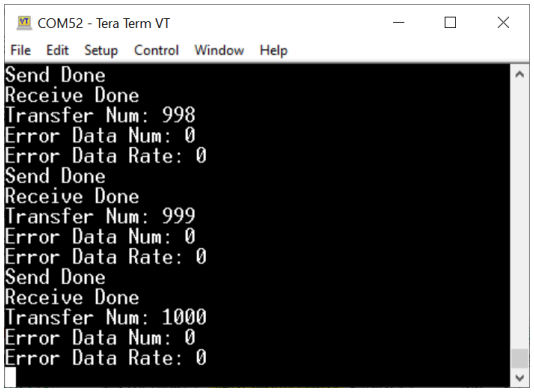

Figure 9. UART debug terminal setup

5 Conclusion

This application note provides information and code to help user reach LPUART maximum baud rate of 24 Mbps on MCX A series. The result shows that the communication is reliable.

6 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN14458 v.1.0	17 October 2024	Initial version

7 Note about the source code in the document

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 must be 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.

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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

Contents

1 Introduction 2
2 LPUART overview 2
2.1 Features 2
2.2 Function 3
2.2.1 Baud rate generation 5
2.2.2 Baud rate tolerance 5
2.2.3 Calculating baud rate tolerance 6
3 Software design 6
3.1 Clock generation 6
3.2 Transfer process 7
3.3 Error data rate calculate 11
3.4 Clock out 11
3.5 Trim the FRO_HF manually 12
3.6 Doing autotrimming using the external crystal 12
4 Test on board 13
4.1 Board connection 13
4.2 UART debug terminal setup 14
4.3 Test result 15
5 Conclusion 15
6 Revision history 15
7 Note about the source code in the document 15
Legal information 17