AN12679

Implement High-Speed RS-485 Application with i.MX RT Based on NXP SDK

Rev. 0 — December 2019

Application Note

Contents

1 Introduction.....1

2 Hardware Kit and SDK.....1

3 Driver and software design.....3
3.1 UARTs features on i.MX RT... 3
3.2 DMA usage.....5
3.3 Software flow.....6

4 Demo setup.....8
4.1 Hardware EVK setup.....9
4.2 Use on-board OPENSDA UART as demonstration.....9
4.3 Software package of the project.....10
4.4 Test result.....11

5 Conclusion.....13

1 Introduction

RS-485 is a standard defining the electrical characteristics of drivers and receivers for use in serial communications systems. Electrical signaling is balanced, and multipoint systems are supported. Multiple receivers may be connected to such a network in a linear, multidrop bus. These characteristics make RS485 useful in industrial control systems and similar applications, especially in Smart Metering, Industrial & Factory Automation, Smart Building. RS-485 is often used with long-distance, high-speed, and noisy environments.

Normally, the UART interface on MCU could act as an RS485 controller with an external transceiver. It provides TTL logical signal and direction control signals; the RS485 transceiver provides voltage level translation, isolation, fault protection, etc. Usually, RS485 communication works in half-duplex mode.

This application note describes how to implement RS-485 communication with NXP i.MX RT series EVK, and how to design software based on NXP MCUXpresso SDK. For high-speed RS-485 bus, normally the baud rate is up to 5 Mbps; the interrupt and software polling mode is not practical in such a scenario. DMA and other mechanisms must be adopted based on SDK API.

2 Hardware Kit and SDK

MIMXRT1060 Evaluation Kit Board is a platform designed to showcase the commonly used features of the i.MX RT1060 Processor in a small, low-cost package. Although there is no RS-485 transceiver on the board, we could still demonstrate how to test the series communication with the EVK. Figure 1 is the block diagram of the evaluation kit.

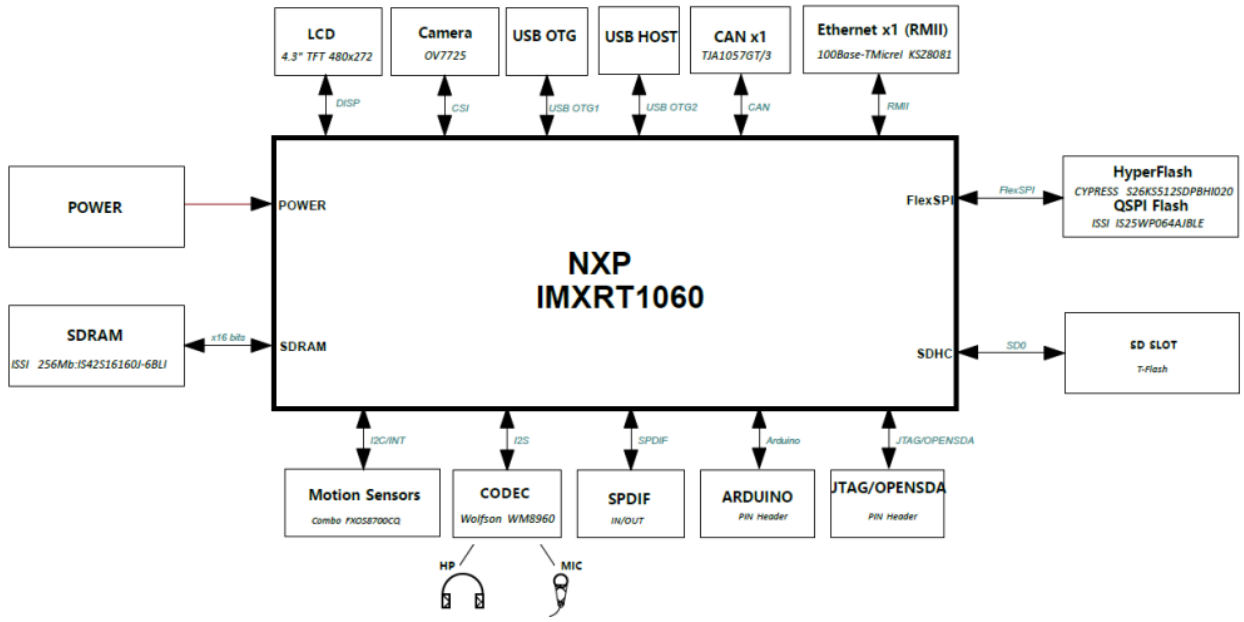

Figure 1. Block diagram

The OpenSDA circuit (CMSIS–DAP) is an open-standard serial and debugs adapter. It bridges serial and debug communications between a USB host and an embedded target processor. We use this OpenSDA interface as power supply, debug interfaces, serial-to-USB converters in this AN, without any external components. The LPUART1 signal is routed to OpenSDA debug USB (which is enumerated as USB CDC UART and CMSIS-DAP debug interface on PC), we could demonstrate half-duplex communication between PC USB and RT1060's UART.

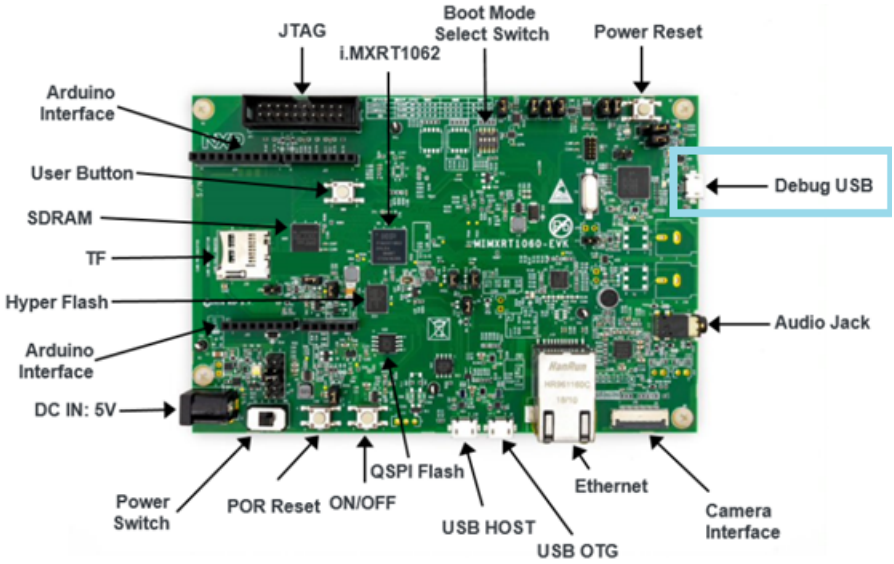

Figure 2. Overview of the MIMXRT1060 EVK board (Front side)

RS-485 circuit needs direction control for the external transceiver. In Figure 3, the receiver enable signal is asserted all the time. Another optional RS-485 connection is to connect RTS_B to both DE and RE_B, it is the commonly used connection. In this case, while driving RTS_B, the receiver of the transceiver is disabled. We could use any GPIO to replace the RTS signal, which could reduce hardware dependence. GPIO_AD_B0_15 could be multiplexed as LPUART1_RTS signal or GPIO signal GPIO1_IO15 to control the direction of the transceiver.

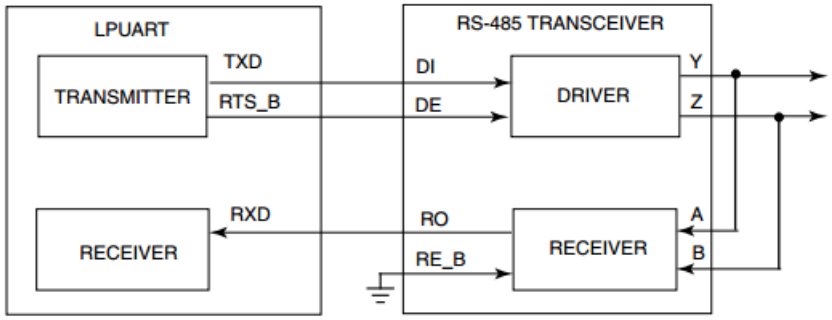

Figure 3. Transceiver driver enable using RTS_B

3 Driver and software design

3.1 UARTs features on i.MX RT

i.MX RT 1060 family has 8 powerful UARTs and they have many useful features. Below are some important features of LPUART on i.MX RT1060, but not including all. Details could be found in IMXRT1060RM, Chapter 48: Low-Power Universal Asynchronous Receiver/Transmitter (LPUART). In this AN, these features listed below is used to implement the high-speed UART communication:

- Full-duplex, standard non-return-to-zero (NRZ) format
- Support a sort of interrupts, DMA, or polling operation:
  - Transmit data register empty and transmission complete
  - Receive data register full
  - Receive overrun, parity error, framing error, and noise error
  - Idle receiver detection, idle type, and timeout are configurable
  - Active edge on receive pin
  - Break detect supporting LIN
  - Receive data match
- Configurable idle length detection supporting 1, 2, 4, 8, 16, 32, 64 or 128 idle characters
- Independent FIFO structure for transmit and receive. LPUART has 4 Entry FIFO, to buffer transmit and receive data. And there is one parameter, called Watermark, which could be:
  - Separated configurable watermark for receiving and transmitting requests
  - Option for the receiver to assert request after a configurable number of idle characters if receive FIFO is not empty
  - When num of buffered data in transmit FIFO <= Transmit Watermark, then trigger “transmit data register empty” interrupt/DMA;
  - When num of buffered data in receive FIFO >= Receive Watermark, then trigger “Receive data register full” interrupt/DMA;
- LPUART could enable transmission complete (TC)/idle receiver detection. In this case, RX idle could be recognized when UART receives the last data on the bus, then claim the bus idle status. We could use this signal to indicate the end of one receiving frame; while Transmission Complete (TC) signal indicates that the last data in UART TX register has been sent out on bus, which means the end of transmitting one frame.

- Error detect flags: users must handle the error flags properly, acknowledge them to inform that they have known the occurred errors, otherwise LPUART do not receive new data anymore.
- Hardware flow control support request-to-send (RTS) and clear-to-send (CTS) signals.

If an enabled event or defined error occurs, Interrupt or DMA action is triggered to execute pre-defined routines.

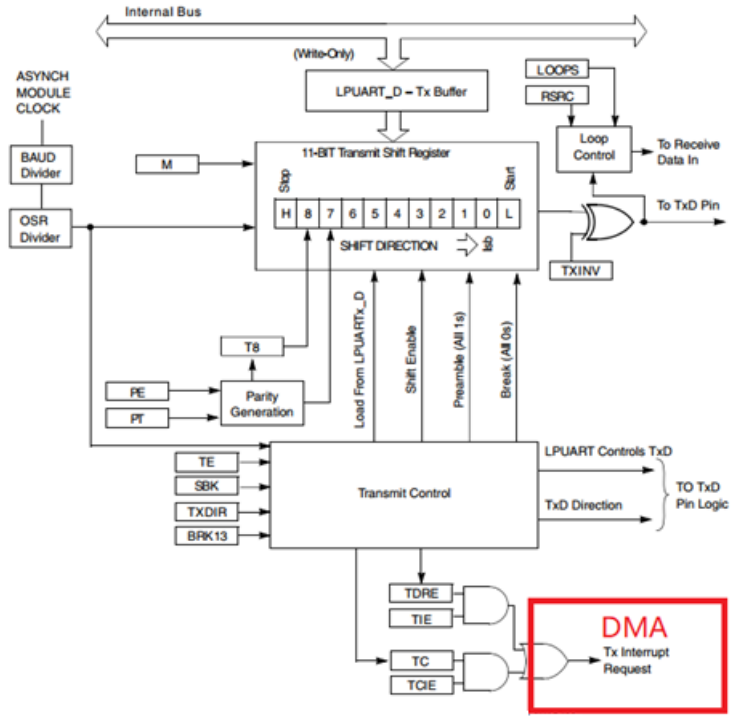

Figure 4. LPUART transmitter block diagram

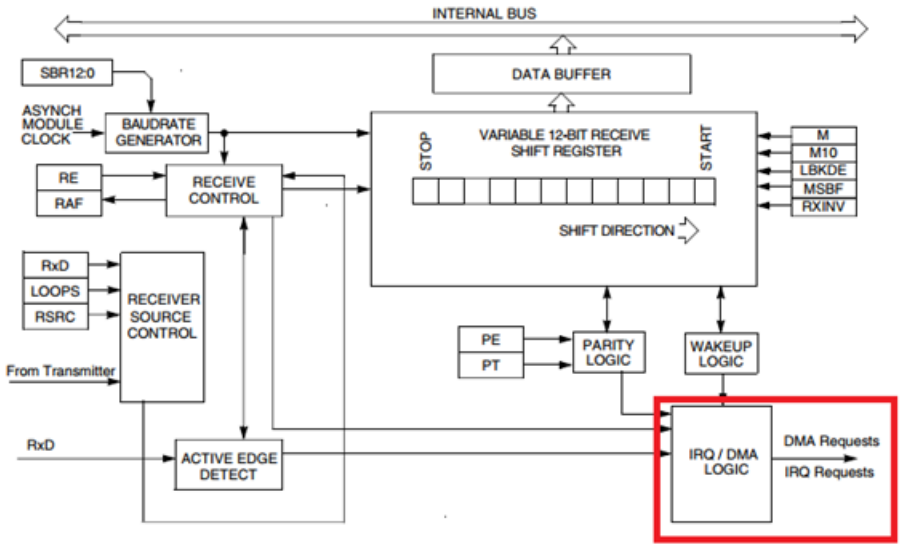

Figure 5. LPUART receiver block diagram

3.2 DMA usage

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data transfers with minimal intervention from a host processor. Below are some features used for UART data transfer, for more details refer to i.MX RT 1060 reference manual Chapter 3: Interrupts, DMA Events, and XBAR Assignments, Chapter 4: Direct Memory Access Multiplexer (DMAMUX), Chapter 5: Enhanced Direct Memory Access (eDMA).

- Basic data-movement operations: transmit data in RAM buffer to LPUART transmitter, or fetch receiver data to RAM buffer:
  - Source address and destination address calculations.
  - Local memory containing transfer control descriptors for each of the 32 channels.
  - All data movement via dual-address transfers: read from source, write to destination.
  - Programmable source and destination address and transfer size.
- Advanced data-movement operations: inner and outer data transfer could support complex operation.
- Channel activation via one of three methods:
  - Explicit software initiation could initiate one transfer by software to reduce MCU workload.
  - Initiation via a channel-to-channel linking mechanism for continuous transfers. Trigger linked transfer could reduce software's participation.
  - Peripheral-paced hardware requests, one per channel. Peripherals could request by themselves at any time, totally no MCU core need to be involved; it can work even if the MCU is sleeping.
- DMA Channels could be assigned with different priority and interrupt action when completion:
  - Fixed-priority and round-robin channel arbitration, user could design channel priority during design phase, decide the most important channel if there maybe conflict of DMA bus bandwidth.
  - The channel completion is reported via programmable interrupt requests.
  - One interrupt per channel, which can be asserted at completion of major iteration count.
  - Programmable error terminations per channel and logically summed together to form one error interrupt to the interrupt controller.

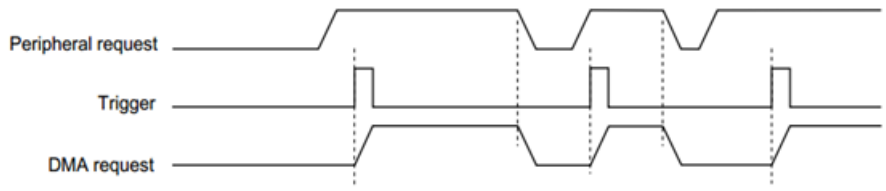

Figure 6. DMAMUX channel triggering

To run the demo project of this AN, we need to use LPUART TX and RX signals as DMA MUX triggering signals to trigger DMA operation. Peripheral request initiates one DMA operation with the triggering signals. When peripherals meet some predefined conditions, take LPUART as an example, if UART TX FIFO empty or RX FIFO full flags are set, it raises a peripheral request. Specific peripheral requests could be combined with DMA channels as DMA channel triggering source, so once a peripheral request is triggered, the DMA operation is implemented.

Channel	Module	Logic	Description
2	LPUART1	OR	UART TX FIFO DMA Request
		OR	UART TX FIFO Async DMA Request
3	LPUART1	OR	UART RX FIFO DMA Request
		OR	UART RX FIFO Async DMA Request
4	LPUART3	OR	UART TX FIFO DMA Request
		OR	UART TX FIFO Async DMA Request
5	LPUART3	OR	UART RX FIFO DMA Request
		OR	UART RX FIFO Async DMA Request
6	LPUART5	OR	UART TX FIFO DMA Request
		OR	UART TX FIFO Async DMA Request

Figure 7. DMA MUX mapping

3.3 Software flow

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for i.MXRT, Kinetis and LPC Microcontrollers. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering many use cases from basic peripheral use examples to full function applications.

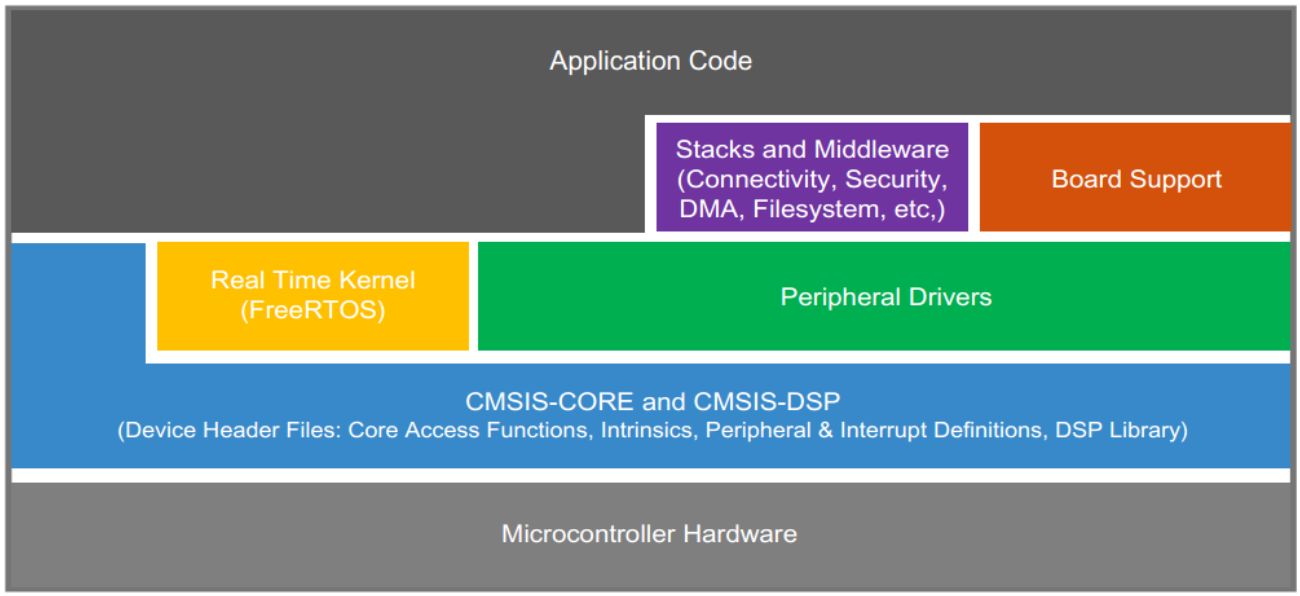

Figure 8. MCUXpresso SDK layers

Take LPUART as an example in SDK 2.6.2, there are 8 demos for LPUARTs, they contain three operation modes of polling/interrupt/DMA; they can work with/without Ring Buffer memory organize; they can use transfer API or not. Here are some demo projects in the SDK:

- edma_rb_transfer
- edma_transfer
- interrupt

- interrupt_rb_transfer
- interrupt_transfer
- interrupt_transfer_seven_bits
- polling
- polling_seven_bits

Description of the edma_transfer demo in SDK release v2.6.2:

1. LPUART edma_transfer demo uses DMA to receive and transmit data;
2. It enables transmit and receive FIFO in default, but watermarks are both set to 1, the effect is the same as disabling FIFO, so it means only one FIFO entry to receive and transmit data;
3. Only basic transmit and receive functions are implemented. The DMA is assigned to transfer fixed length (8 bytes) data, after the transfer is finished, DMA interrupt occurs.

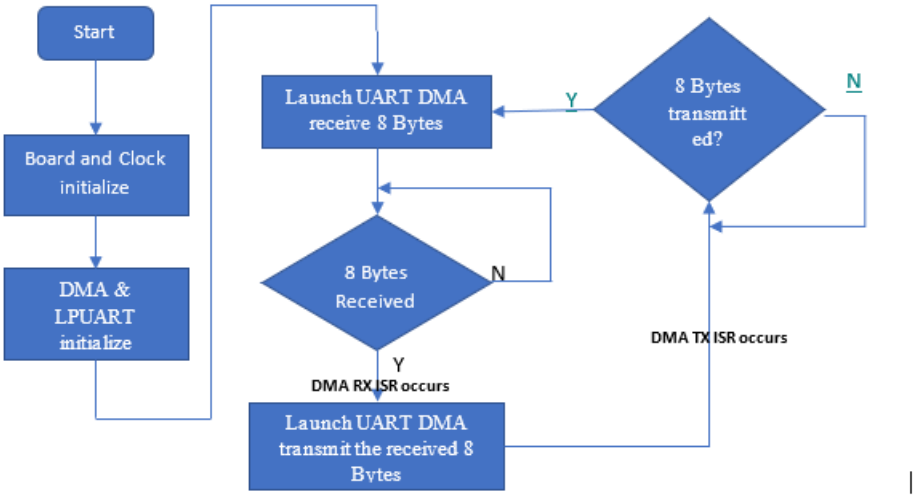

Figure 9. edma_transfer flow

If customers use this demo for RS-485 communication directly, they will have below limitations:

1. If error happens, as there is no error handler entry, the UART communication flow will be blocked;
2. The receiving DMA is assigned to receive 8 bytes, the frame length is fixed, which is not feasible for every case. If the received data length doesn't reach the pre-defined length, the receive DMA routine will not exit. Furthermore, in a noisy environment, the communication may lose some data bytes, which has similar risk as fixed frame length.
3. The demo code uses DMA to transmit complete interrupt to switch demo state machine from TX to RX. But for RS-485 communication, if customers switch external RS485 RX/TX transceiver in the ISR, receiver will lose data. Because DMA transmit complete flag only indicates the data have been transferred from RAM buffer to LPUART TX FIFO, it doesn't indicate all data are sent out on the bus.

To design the project in this AN which is suitable for a real RS-485 industry bus, we modified the DMA transfer demo in SDK, while we don't modify any original driver code (fsl_lpuart.c and fsl_lpuart_edma.c), to comply with existing SDK API.

Description of the new edma_transfer demo features:

1. On the existing LPUART_edma_transfer project, set DMA receiving buffer to 2KB (assume this size is large enough, no RS485 frame exceed 2KB size, so that DMA receiving full interrupt never occurs. If the buffer size exceeds 2KB during test, the preceding 2KB data will be discarded);
2. On MIMXRT1060-EVK board, use LPUART1 as RS-485 communication port, which is bridged to USB CDC interface and works on half duplex mode. Connect wire GPIO_AD_B0_15 as an external transceiver direction control signal;

3. After reset, the LPUART works as receiver, receive idle interrupt and error interrupt are enabled; of course, we provide an error handler in LPUART interrupt service routine;
4. After receiving one frame sent from PC console, MCU will enter RX idle interrupt service routine, then judge whether this data frame is for this node, make decision to receive another frame or echo data; then reset DMA receive state (initialize DMA state machine and pointer etc.)
5. If MCU needs to echo data, in the main loop, it switches RS485 transceiver to TX mode, organize an echo frame, send it out using Transmit (TX) DMA channel. In Transmit (TX) finish DMA interrupt service routine, enable transmit complete (TC) interrupt; in transmit complete (TC) interrupt service routine, switch RS-485 direction to receive again and reset DMA receive state machine, then restore to status (3).

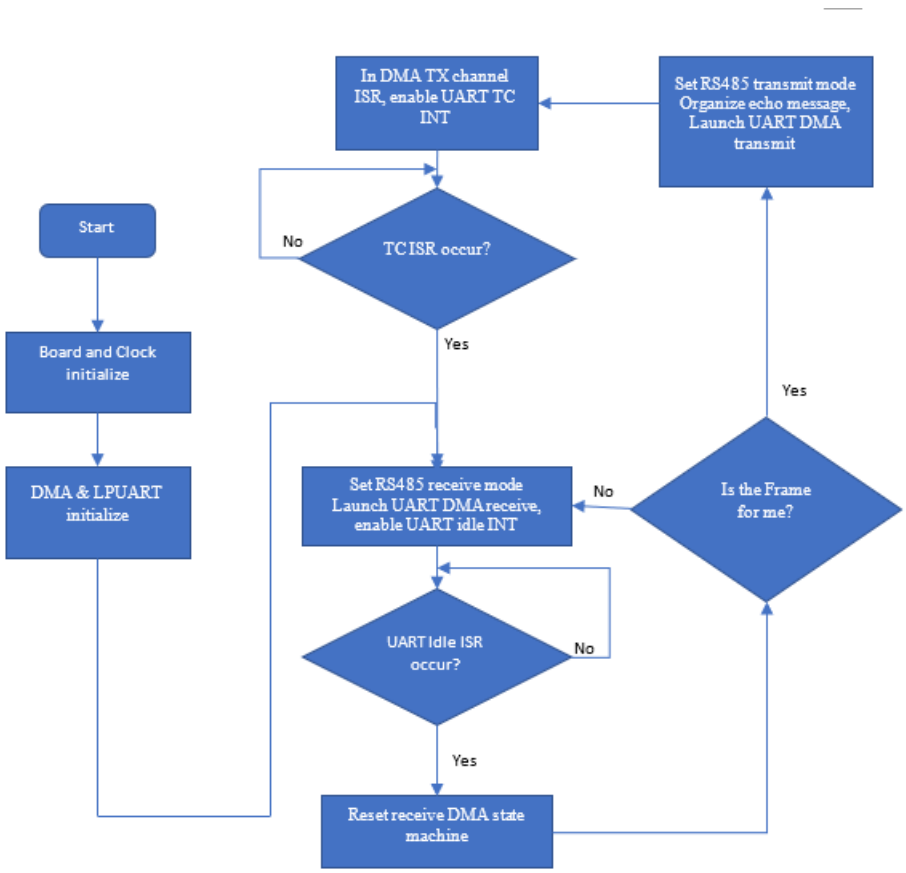

Figure 10. new edma_transfer project work flow

NOTE

1. In this new project, set TX FIFO watermark to 3 (maximum 4): when TX FIFO entry number is less than 3, the TX DMA channel is triggered to transfer data from RAM buffer to LPUART TX FIFO;
2. In a real application, UART error flag must be handled, which could bring MCU back to normal status, even working in a tough environment.
3. We keep original SDK driver unchanged, re-use all API in SDK.

4 Demo setup

4.1 Hardware EVK setup

We need:

- one MIMXRT1060-EVK
- one Micro USB cable
- PC with Serial Port Tool installed, such as TeraTerm
- Fly wire GPIO_AD_B0_15 on Pin4 of U12 as external transceiver direction control signal

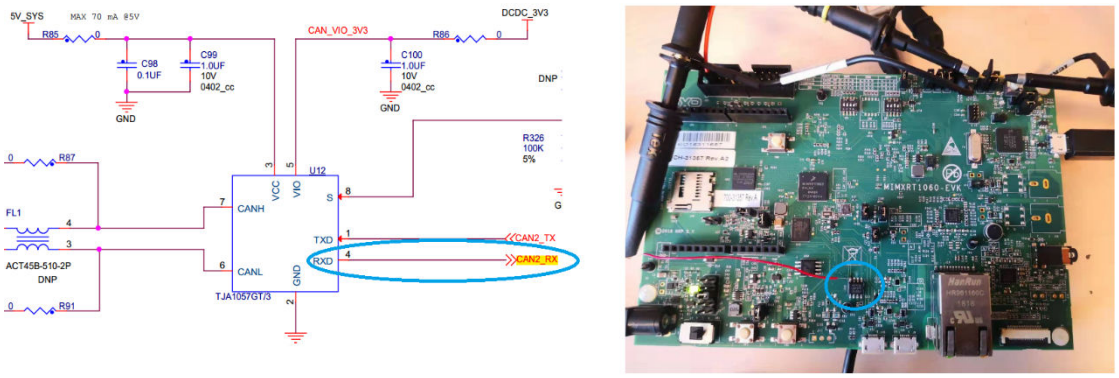

Figure 11. Direction control signal

4.2 Use on-board OPENSDA UART as demonstration

Check to confirm J45/J46 are connected. Connect J41 (USB port) with a PC, the board is emulated as CDC device and CMSIS-DAP interface on the PC. Compile and download test code onto iMXRT1060 EVK, then restart. Open TeraTerm with CDC UART, then type in one string, Teraterm receives the same string back. On the EVK board, no real RS485 transceiver exists, we set the baud rate as 115200 bps in this test. But in fact as this demo uses DMA and interrupt together, it could implement higher baud rate on field application.

NOTE

If you have an external RS485 transceiver on hand, unplug J45/J46 and connect them as UART_RXD/UART_TXD signal to transceiver, and connect direction control signal also. Then you could test real RS485 hardware.

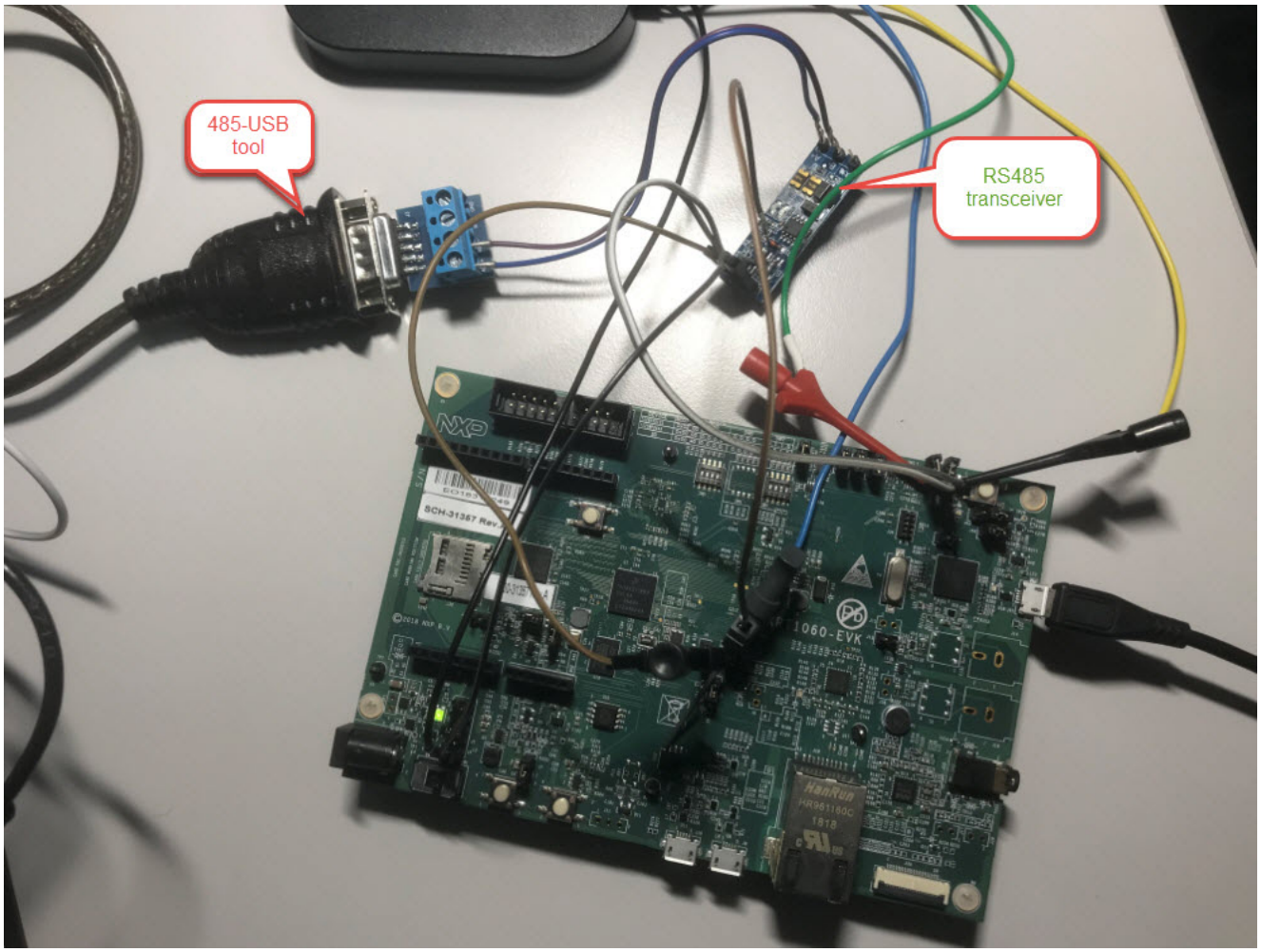

Figure 12. Use External RS485 transceiver

4.3 Software package of the project

The SW package could be unzipped to “SDK_2.6.2_EVK-MIMXRT1060\boards\evkmimxrt1060\driver_examples\lpuart”, customers can use IAR or Keil to rebuild and download it to EVK for test. There is one more Macro definition in the project build setting compared with original SDK project:

ENABLE_RTS_TRANSCEIVER=0. Means GPIO_AD_B0_15 will act as GPIO

ENABLE_RTS_TRANSCEIVER=1. Means GPIO_AD_B0_15 will act as LPUART1_RTS signal

Take IAR project as an example, customer can modify the ENABLE_RTS_TRANSCEIVER from project options.

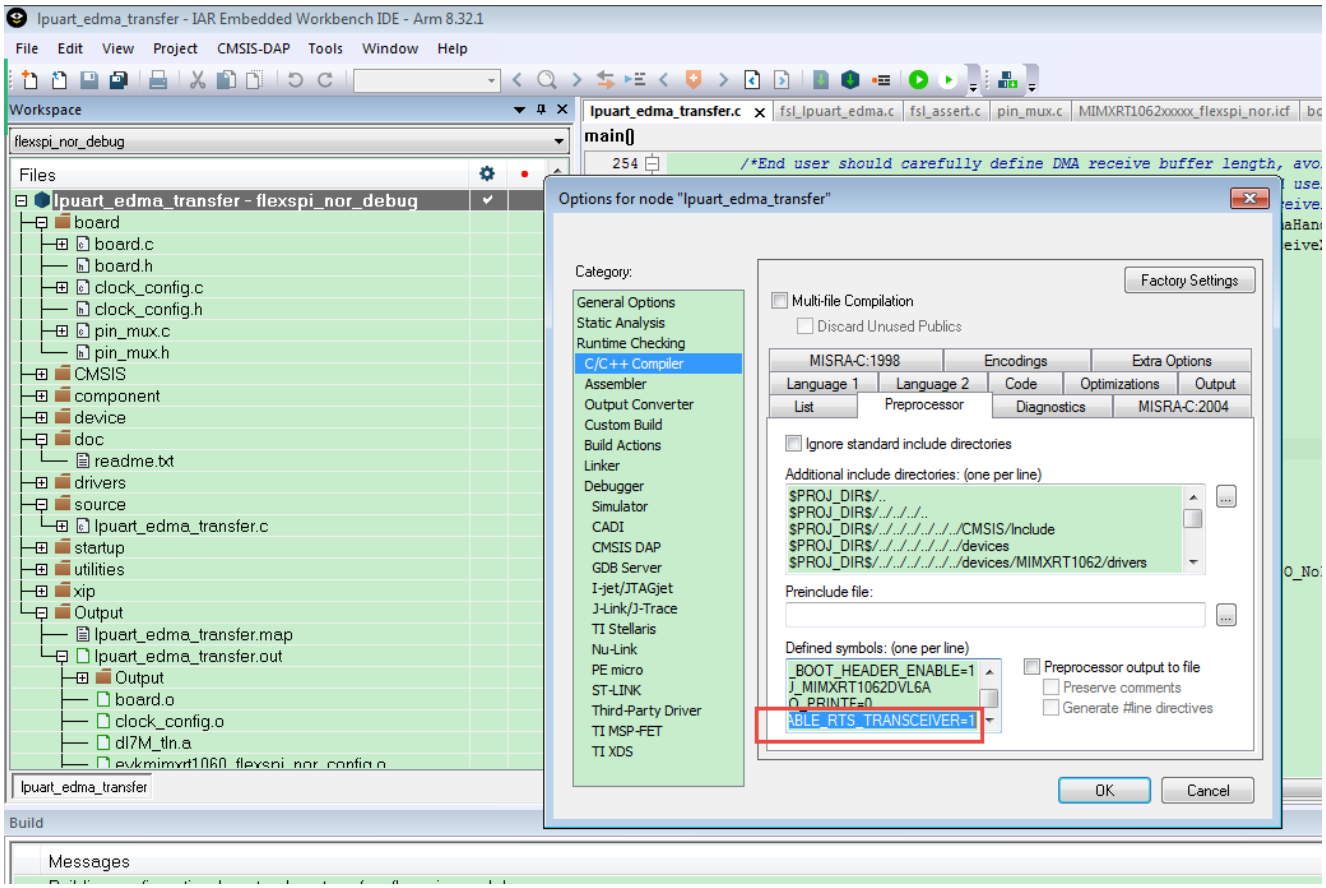

Figure 13. Macro Definition in software package

4.4 Test result

With the serial Port tool, the sent string could be echoed by RT1060 EVK, and by watching the UART RX/TX and direction control signals, we could know the timing better. Comparing with the two-direction control methods, using RTS controls have one-bit fixed time difference with Start bit or last Stop bit. Using GPIO control actually has similar performance, thanks to i.MX RT powerful performance.

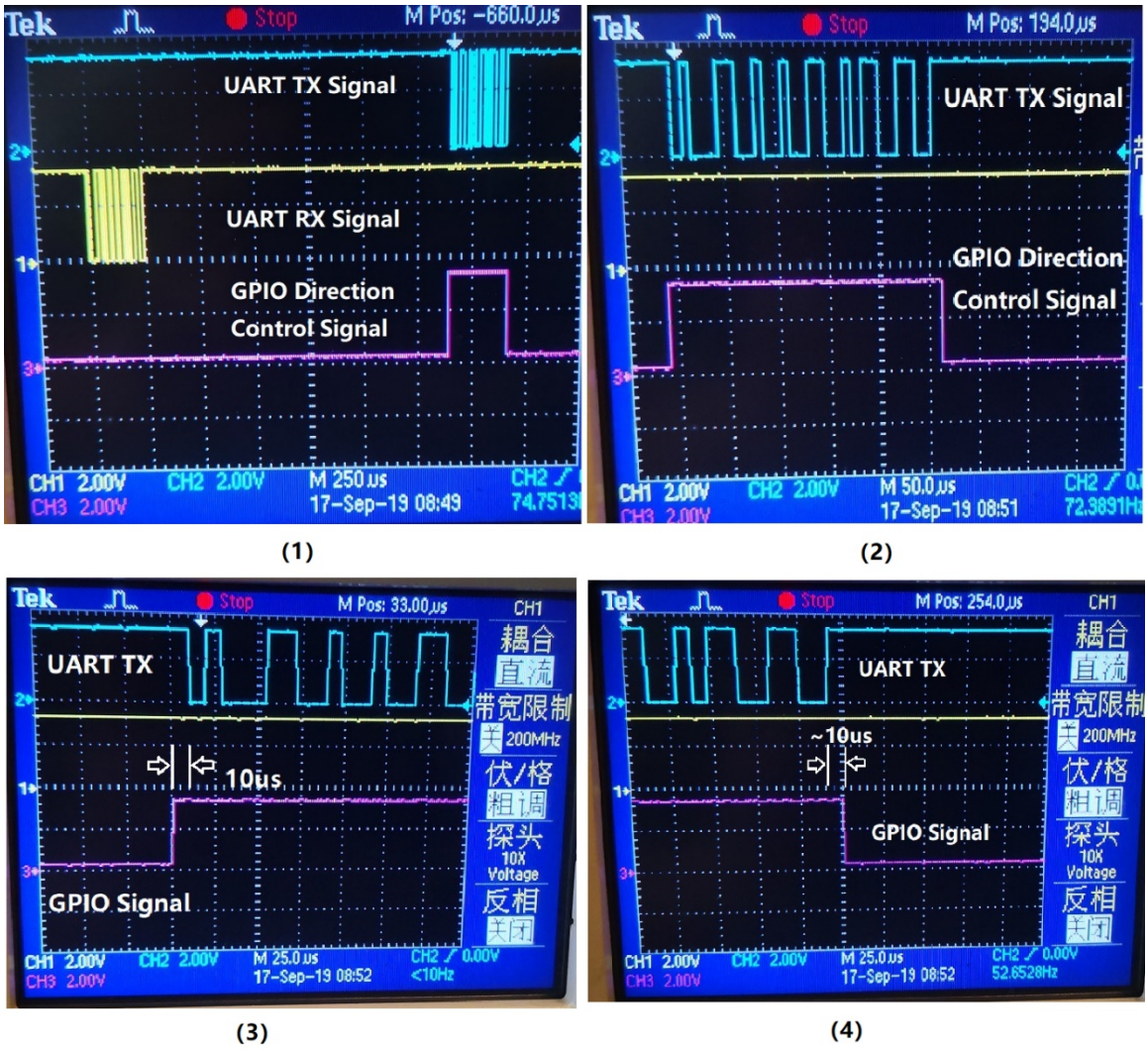

Figure 14. Timing Wave using GPIO control

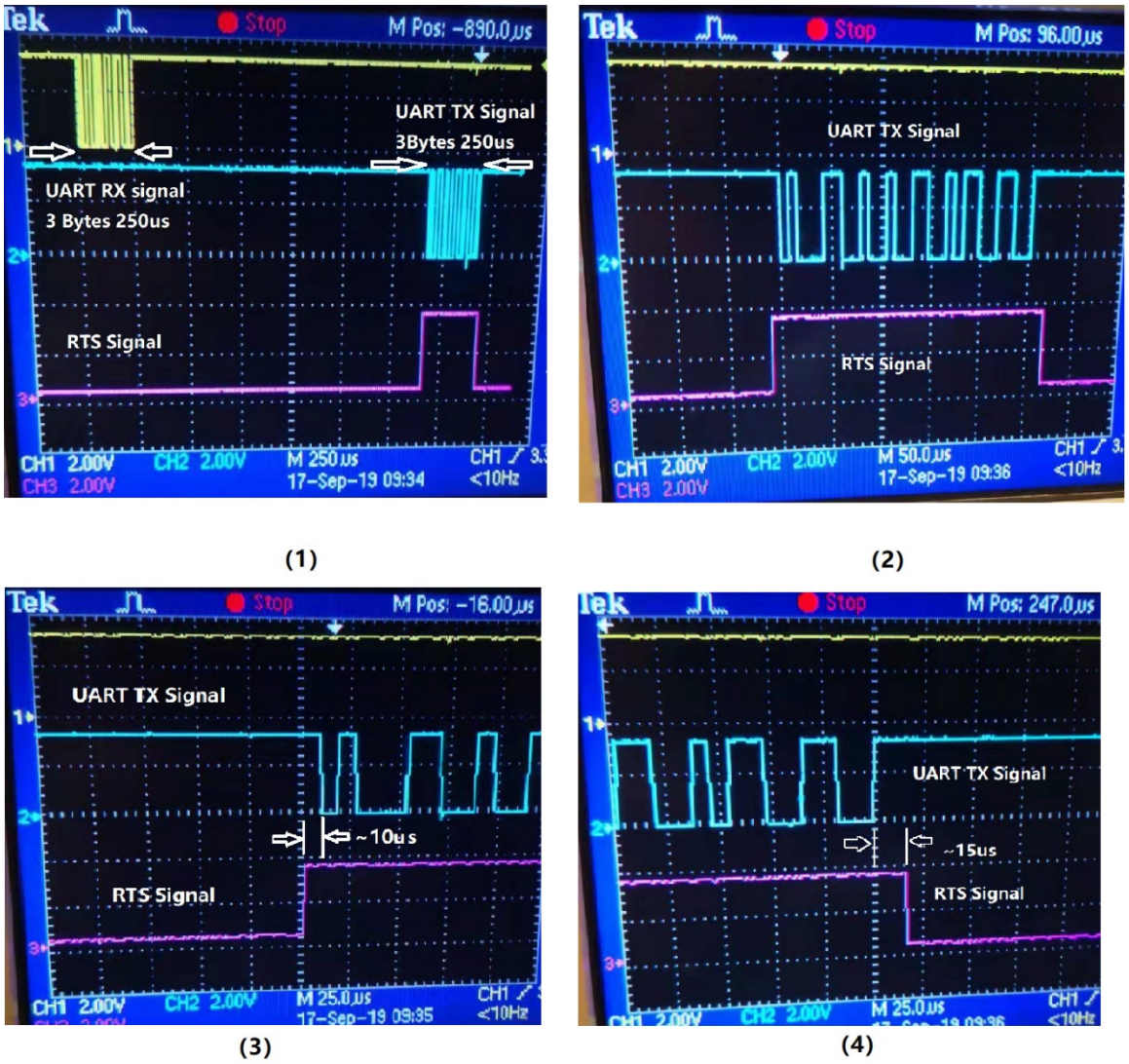

Figure 15. Timing Wave using RTS control

5 Conclusion

NXP SDK has provided enough DMA and UART driver APIs, which could be used for real high-speed RS485 applications. However, we need to modify the demos in the SDK to meet the actual requirements in real cases. In this application note, a new project was built to implement high-speed RS-485 communication, with considering to handle errors and unexpected events that happen in real cases. After long-time test on real customer projects, it is proved to be workable.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: December 2019

Document identifier: AN12679

arm