FEXT766x_BLSDK_SW_Manual-01

ML7661/ML7660 Batteryless SDK Software Manual

Issue Date: Feb 13, 2025

Preface

This document is a manual for the battery-less SDK sample software.

By using the SDK sample software, it is possible to control and acquire data from SPI peripheral devices (such as sensors) connected to the Rx side using the rotational reference design and other tools.

Notation

Classification	Notation	Description
Numeric value	XXh, XXH, 0xXX	Indicates a hexadecimal number.
Unit	word, W	1 word = 16 bits
	byte, B	1 byte = 8 bits
	nibble, N	1 nibble = 4 bits
	mega-, M	10^6
	kilo-, K	2^{10} = 1024
	kilo-, k	10^3 = 1000
	milli-, m	10^{-3}
	micro-, µ	10^{-6}
	nano-, n	10^{-9}
	second, s (lower case)	second

Terms and Abbreviations

Terms and Abbreviations	Description
GUI	Graphical User Interface
URI	Uniform Resource Identifier
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
Tx / Poller	An NFC Forum Device in Poll Mode (Poll mode: The mode of an NFC Forum Device in which it sends Commands and receives Responses)
Rx / Listener	An NFC Forum Device in Listen Mode (Listen mode: The mode of an NFC Forum Device in which it receives Commands and sends Responses)
Short Packet Format	ROHM’s original packet format
T3T	Role of a Listener when it has gone through a number of States. In this mode, the Listener supports the execution of Type 3 Tag commands to read or write NDEF messages.

Table of Contents

1. Overview

By using the SDK sample software, it is possible to control and acquire data from SPI peripheral devices (such as sensors) connected to the Rx side using the rotational reference design and other tools.

Aligning the antennas of the transmitting side (Tx side) and the receiving side (Rx side) supplies power to the peripheral devices connected to the receiving side. The transmitting side can periodically acquire data from or control the peripheral devices.

We provide SDK sample software for the ML7660-EVK-002 reference design. By using this SDK to develop software, you can change the peripheral devices used with the ML7660-EVK-002 reference design. As a sample application of this SDK, we provide code using the strain sensors (STREAL SR300) by GLOSEL Co., Ltd.

Additionally, this SDK sample software can also be applied to communication control software.

2. System Configuration

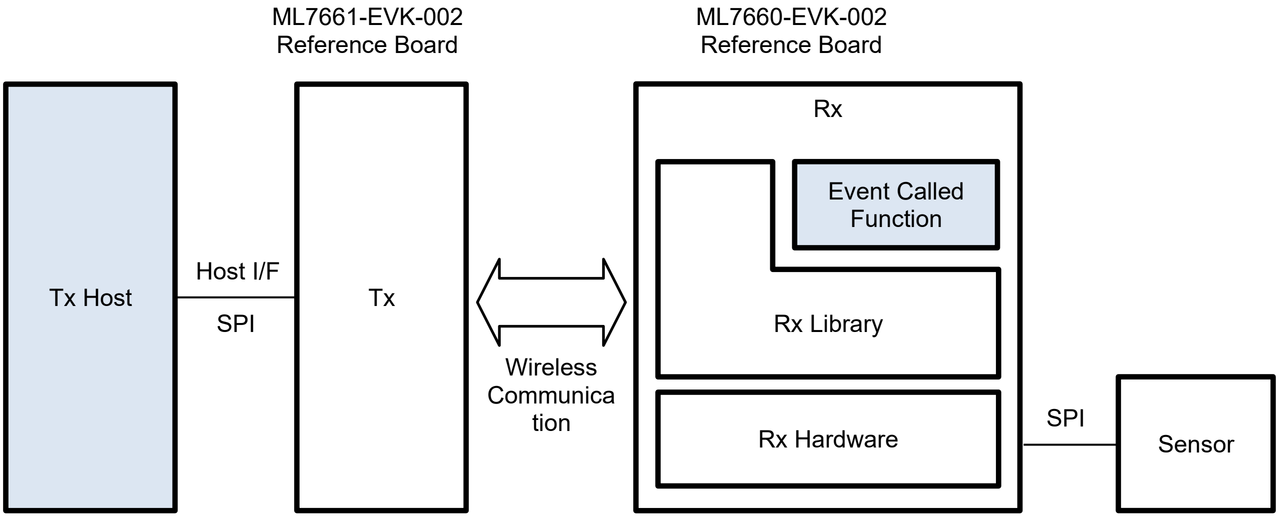
Below is the system configuration using the rotational reference design. The rotational reference design consists of the ML7661-EVK-002 reference board and the ML7660-EVK-002 reference board.

By developing the Host on the transmitting side (Tx side) and the Event Called Function” on the receiving side (Rx side), it is possible to use any peripheral device (sensor).

The Rx acquires sensor data via SPI. The Tx communicates with the Rx to obtain the sensor data. The Tx Host controls the Tx and receives the sensor data acquired by the Rx.

The Event Called Function is a function that is called when an event occurs within the Rx Library. If you want to use any sensor, please implement this function.

For the Host Interface (Host I/F) between the Tx Host and Tx, please refer to the “NFC Reference Software Host Command manual”.



3. Process Flow

Below is an example of the application process flow. In this example, data is periodically acquired from the sensor.

Batteryless Operation: Pre-processing

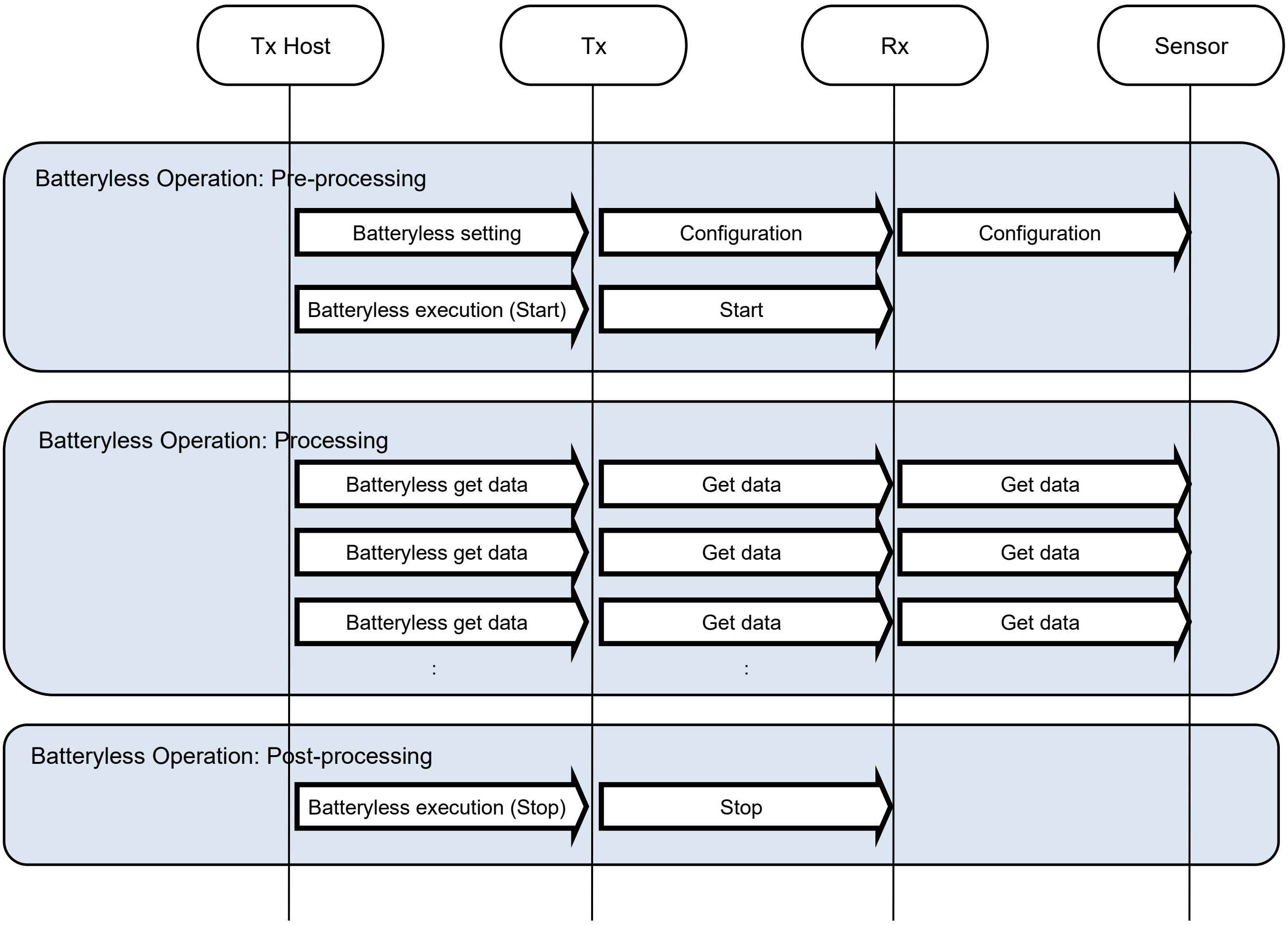
Configure the sensors.
Control the start of Batteryless execution.

Batteryless Operation: Processing

The Tx Host repeatedly acquires sensor data via the Tx and Rx.

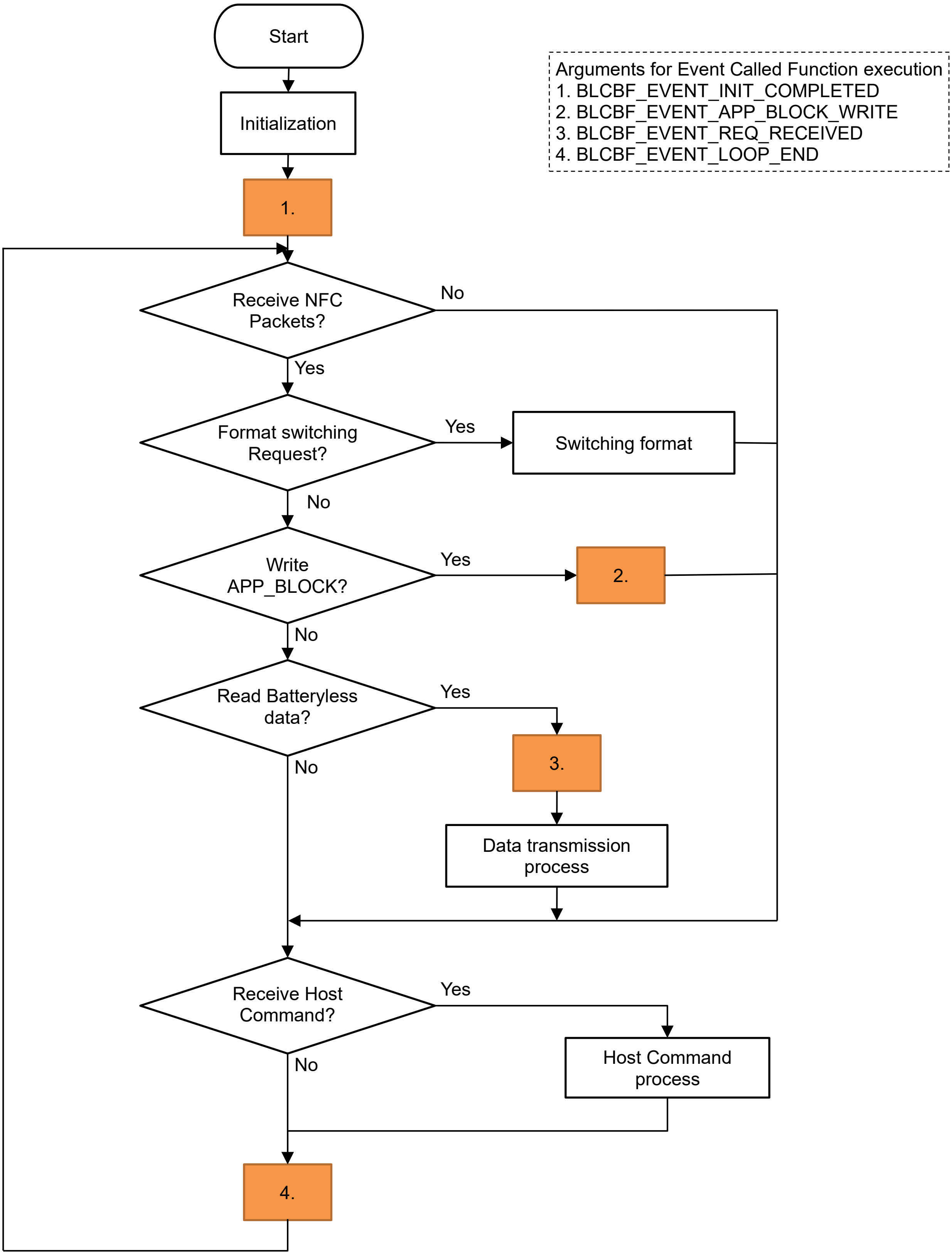
Batteryless Operation: Post-processing

Control the stop of Batteryless execution.



4. Calling the Event Called Function

The Rx Library has an internal process flow. It executes functions defined in the Event Called Function in response to events occurring within the process flow. The diagram below shows the process flow within the Rx Library. The Event Called Function is called at the points indicated 🟧.



5. API Specifications

5.1 Event Called Function

Function Name	Function
`void BLCBF_EventOccurred(BLCBF_Event event)`	This is a function that is called by an event.

5.1.1 Details of the Event Called Function

5.1.1.1 BLCBF_EventOccurred

Function Name	void BLCBF_EventOccurred(BLCBF_Event event)
Argument	BLCBF_Event event: Event ID
Return value	None
Processing	This is a function that is called by an event.

5.2 API function

Function Name	Function
`void BLAPI_listener_main(void)`	Main function of the Rx. Please call it after startup.
`void BLAPI_readAppBlock(uint8_t appBlockData[], uint8_t rdata_size)`	Reads the APP_BLOCK.
`void BLAPI_writeAppBlockRes(uint8_t appBlockResData[], uint8_t wdata_size)`	Writes the response data to APP_BLOCK.
`uint8_t BLAPI_setBatteryLessData(uint8_t data_array[], uint8_t data_size)`	Writes the Batteryless response data.
`void BLAPI_setHALT(void)`	Transitions the system to a HALT (low power consumption) state.
`void BLAPI_port_SetOutput(Const uint8_t num, Const uint8_t out)`	Set the output value of the port.
`uint8_t BLAPI_port_GetInput(Const uint8_t num)`	Gets the input value of the port.
`void BLAPI_port_SetControl(Const uint8_t num, Const BLAPI_PortCtrl ctrl)`	Sets the port configuration.
`void BLAPI_port_SetMode(Const uint8_t num, Const BLAPI_PortMode mode)`	Sets the port mode.
`uint32_t BLAPI_timer_GetTime(void)`	Gets the timer count value counted in milliseconds.
`BLAPI_TimerTimeout BLAPI_timer_CheckTimeout(Const uint32_t base_ms, Const uint32_t timeout_ms)`	Check if a timeout has occurred for the timer counted in milliseconds.
`uint32_t BLAPI_timer_45_GetTime100us(void)`	Gets the timer count value counted in 0.1 milliseconds.
`uint8_t BLAPI_timer4_GetStatus(void)`	Gets the status of the timer counted in 0.1 milliseconds.
`void BLAPI_wdt_clear(void)`	Clear the WDT counter.
`void BLAPI_setLed(Const BLAPI_LedCtrl on_off)`	Sets the state of the LED.
`void BLAPI_host_SetIRQ(Const uint8_t req)`	Set the interrupt of the host interface.
`uint8_t BLAPI_host_GetStatus(void)`	Get the value of the status register (STATUS) of host interface.
`void BLAPI_host_SetStatus(Const uint8_t status)`	Set the value of the status register (STATUS) of host interface.
`BLAPI_TimerTimeout BLAPI_WaitHostAccess(uint32_t timeout)`	Waits for the completion of the host command processing.

5.2.1 Details of the Event Called Function

5.2.1.1 BLAPI_listener_main

Function Name	void BLAPI_listener_main(void)
Argument	None
Return value	None
Processing	Main function of the Rx. Please call it after startup.

5.2.1.2 BLAPI_readAppBlock

Function Name	void BLAPI_readAppBlock(uint8_t appBlockData[], uint8_t rdata_size)
Argument	uint8_t appBlockData[]: Pointer to the area where the data to be read from APP_BLOCK is stored. uint8_t rdata_size: Size of the data to be read.
Return value	None
Processing	Reads the APP_BLOCK.

5.2.1.3 BLAPI_writeAppBlockRes

Function Name	void BLAPI_writeAppBlockRes(uint8_t appBlockResData[], uint8_t wdata_size)
Argument	uint8_t appBlockResData[]:Pointer to the area where the data to be written to APP_BLOCK is stored. uint8_t wdata_size: Size of the data to be written.
Return value	None
Processing	Writes the response data to APP_BLOCK.

5.2.1.4 BLAPI_setBatteryLessData

Function Name	uint8_t BLAPI_setBatteryLessData(uint8_t data_array[], uint8_t data_size)
Argument	uint8_t data_array[]: Data pointer uint8_t data_size: Data size
Return value	0: Setting succeeded 1: Setting failed
Processing	Writes the Batteryless response data.

5.2.1.5 BLAPI_setHALT

Function Name	void BLAPI_setHALT(void)
Argument	None
Return value	None
Processing	Transitions the system to a HALT (low power consumption) state.

5.2.1.6 BLAPI_port_SetOutput

Function Name	void BLAPI_port_SetOutput(Const uint8_t num, Const uint8_t out)
Argument	Const uint8_t num: Port number Const uint8_t out: Setting value
Return value	None
Processing	Set the output value of the port.

5.2.1.7 BLAPI_port_GetInput

Function Name	uint8_t BLAPI_port_GetInput(Const uint8_t num)
Argument	Const uint8_t num: Port number
Return value	0: Low 1: High
Processing	Gets the input value of the port.

5.2.1.8 BLAPI_port_SetControl

Function Name	void BLAPI_port_SetControl(Const uint8_t num, Const BLAPI_PortCtrl ctrl)
Argument	Const uint8_t num: Port number Const BLAPI_PortCtrl ctrl: Setting value
Return value	None
Processing	Sets the port configuration.

5.2.1.9 BLAPI_port_SetMode

Function Name	void BLAPI_port_SetMode(Const uint8_t num, Const BLAPI_PortMode mode)
Argument	Const uint8_t num: Port number Const BLAPI_PortMode mode: Setting mode value
Return value	None
Processing	Sets the port mode.

5.2.1.10 BLAPI_timer_GetTime

Function Name	uint32_t BLAPI_timer_GetTime(void)
Argument	None
Return value	Timer count value (unit: ms)
Processing	Gets the timer count value counted in milliseconds.

5.2.1.11 BLAPI_timer_CheckTimeout

<table>
<tr><td>Function Name</td><td>BLAPI_TimerTimeout BLAPI_timer_CheckTimeout(Const uint32_t base_ms, Const uint32_t timeout_ms)</td></tr>
<tr><td>Argument</td><td>Const uint32_t base_ms: Base timer value (Unit: ms)
Const uint32_t timeout_ms: Timeout duration (Unit: ms)</td></tr>
<tr><td>Return value</td><td>BLAPI_TIMER_TIMEOUT: Timeout occurred
BLAPI_TIMER_NO_TIMEOUT: No timeout occurred</td></tr>
<tr><td>Processing</td><td>

Check if a timeout has occurred for the timer counted in milliseconds.

Example usage (exiting a while loop after 100ms)

```
uint32_t timeval;
uint32_t timeout_ms = 100;

timeval = BLAPI_timer_GetTime();
while(1)
{
            :
    if (BLAPI_timer_CheckTimeout(timeval, timeout_ms) == BLAPI_TIMER_TIMEOUT)
    {
        /* Timeout Operation */
        break;
    }
            :
}
```

</td></tr>
</table>

5.2.1.12 BLAPI_timer_45_GetTime100us

Function Name	uint32_t BLAPI_timer_45_GetTime100us(void)
Argument	None
Return value	Timer count value (unit: 100us)
Processing	Gets the timer count value counted in 0.1 milliseconds. To use this, the “Batteryless timer synchronous” host command must be executed. This is an evaluation feature and may be removed in the future.

5.2.1.13 BLAPI_timer4_GetStatus

Function Name	uint8_t BLAPI_timer4_GetStatus(void)
Argument	None
Return value	Status 0: The timer counter in 0.1 milliseconds is stopped 1: The timer counter in 0.1 milliseconds is running
Processing	Get the STATUS register value of host interface. This is an evaluation feature and may be removed in the future.

5.2.1.14 BLAPI_wdt_clear

Function Name	void BLAPI_wdt_clear(void)
Argument	None
Return value	None
Processing	Clear the WDT counter. The WDT will enter a detection state if it is not cleared for 8 seconds. Please execute this function as needed.

5.2.1.15 BLAPI_setLed

Function Name	void BLAPI_setLed(Const BLAPI_LedCtrl on_off)
Argument	Const BLAPI_LedCtrl on_off: BLAPI_LED_ON BLAPI_LED_OFF
Return value	None
Processing	Sets the state of the LED. When BLAPI_LED_ON is set, the P02 pin is set to Low, turning the LED on. When BLAPI_LED_OFF is set, the P02 pin is set to High, turning the LED off. (This cannot be used with the ML7660-EVK-002 reference board.)

5.2.1.16 BLAPI_host_SetIRQ

Function Name	void BLAPI_host_SetIRQ(Const uint8_t req)
Argument	Const uint8_t req: 0 1
Return value	None
Processing	Set the interrupt of the host interface.

5.2.1.17 BLAPI_host_GetStatus

Function Name	uint8_t BLAPI_host_GetStatus(void)
Argument	None
Return value	The status register (STATUS) value
Processing	Get the value of the status register (STATUS) of host interface.

5.2.1.18 BLAPI_host_SetStatus

Function Name	void BLAPI_host_SetStatus(Const uint8_t status)
Argument	The status register (STATUS) value
Return value	None
Processing	Set the value of the status register (STATUS) of host interface.

5.2.1.19 BLAPI_WaitHostAccess

Function Name	BLAPI_TimerTimeout BLAPI_WaitHostAccess(uint32_t timeout)
Argument	Const uint32_t timeout: Timeout duration (Unit: ms)
Return value	BLAPI_TIMER_TIMEOUT: Timeout occurred BLAPI_TIMER_NO_TIMEOUT: No timeout occurred
Processing	Waits for the completion of the host command processing. If a timeout value is set, timeout processing is added. If the host command processing is not completed by the timeout value, a timeout occurs. If the timeout value is set to 0, no timeout processing is performed, and it will wait indefinitely until the host command processing is completed, so please be careful.

5.3 Macro Definition

Definition	Description
#define EI_EN (1U)	Enable EI
#define EI_DIS (0U)	Disable EI
#define IRQ_EN (1U)	Enable IRQ
#define IRQ_CLR (0U)	Clear IRQ
#define IRQ_SetSIU00INT(flg) (write_bit(QSIU00, (flg)))	Set QSIU00
#define EI_SetSIU0INT(flg) (write_bit(ESIU00, (flg)))	Set ESIU0

Definition	Description
#define ML766X_PORT_P02 (2U)	P02 Port
#define ML766x_PORT_P13 (13U)	P13 Port
#define ML766X_PORT_P16 (16U)	P16 Port

Definition	Description
#define APP_BLOCK_SIZE (1U)	Block size of the APP_BLOCK area
#define APP_BLOCK_DATA_SIZE (APP_BLOCK_SIZE * 16U)	Byte size of the APP_BLOCK area
#define BL_DATA_BLOCK_SIZE_T3T (5U)	Block size used for T3T Format communication
#define BL_DATA_BYTE_SIZE_T3T (BL_DATA_BLOCK_SIZE_T3T * 16U)	Byte size used for T3T Format communication
#define BL_DATA_BYTE_SIZE_PROP (253U)	Data size available for use in Short Packet Format

5.4 Type Definition

Enum Name	Description

typedef enum {
    BLCBF_EVENT_INIT_COMPLETED = 0x00U,
    BLCBF_EVENT_APP_BLOCK_WRITE,
    BLCBF_EVENT_REQ_RECEIVED,
    BLCBF_EVENT_LOOP_END
} BLCBF_Event;
	ID of the event called by the Event Called Function

typedef enum {
    BLAPI_PORT_OUTPUT_HIZ = 0U,
    BLAPI_PORT_OUTPUT_P_OPEN_DRAIN,
    BLAPI_PORT_OUTPUT_N_OPEN_DRAIN,
    BLAPI_PORT_OUTPUT_CMOS,
    BLAPI_PORT_INPUT_HIZ,
    BLAPI_PORT_INPUT_PULLDOWN,
    BLAPI_PORT_INPUT_PULLUP
} BLAPI_PortCtrl;
	Port Control

typedef enum {
    BLAPI_PORT_FUNC_1ST = 0U,
    BLAPI_PORT_FUNC_2ND,
    BLAPI_PORT_FUNC_3RD,
    BLAPI_PORT_FUNC_4TH
} BLAPI_PortMode;
	Port Mode

typedef enum {
    BLAPI_TIMER_TIMEOUT,
    BLAPI_TIMER_NO_TIMEOUT
} BLAPI_TimerTimeout;
	Return code of checking timer

typedef enum {
    BLAPI_LED_ON = 1U,
    BLAPI_LED_OFF = 0U
} BLAPI_LedCtrl;
	LED ON/OFF Flag

5.4.1 Details of Type Definitions

5.4.1.1 BLCBF_Event

Enum Name	Member	Description
BLCBF_Event	BLCBF_EVENT_INIT_COMPLETED	Event at initialization completed
	BLCBF_EVENT_APP_BLOCK_WRITE	Event at block write (Application Block)
	BLCBF_EVENT_REQ_RECEIVED	Event at request command received
	BLCBF_EVENT_LOOP_END	Event at main loop end

5.4.1.2 BLAPI_PortCtrl

Enum Name	Member	Description
BLAPI_PortCtrl	BLAPI_PORT_OUTPUT_HIZ	Output Hiz (Output default)
	BLAPI_PORT_OUTPUT_P_OPEN_DRAIN	Output Pch Open drain
	BLAPI_PORT_OUTPUT_N_OPEN_DRAIN	Output Nch Open drain
	BLAPI_PORT_OUTPUT_CMOS	Output CMOS
	BLAPI_PORT_INPUT_HIZ	Input Hiz (Iutput default)
	BLAPI_PORT_INPUT_PULLDOWN	Input Pull down
	BLAPI_PORT_INPUT_PULLUP	Input Pull up

5.4.1.3 BLAPI_PortMode

Enum Name	Member	Description
BLAPI_PortMode	BLAPI_PORT_FUNC_1ST	1st Function (Output/Input default)
	BLAPI_PORT_FUNC_2ND	2nd Function
	BLAPI_PORT_FUNC_3RD	3rd Function (reserve)
	BLAPI_PORT_FUNC_4TH	4th Function (reserve)

5.4.1.4 BLAPI_TimerTimeout

Enum Name	Member	Description
BLAPI_TimerTimeout	BLAPI_TIMER_TIMEOUT	Timeout occurred
	BLAPI_TIMER_NO_TIMEOUT	Timeout no occurred

5.4.1.5 BLAPI_LedCtrl

Enum Name	Member	Description
BLAPI_LedCtrl	BLAPI_LED_ON	LED ON
	BLAPI_LED_OFF	LED OFF

6. API Specifications (SSIO Driver)

This chapter describes the API specifications for the SSIO driver. The SSIO hardware and SSIO driver of the ML7660 are reused from the hardware and driver of the ML62Q1000 series. Therefore, the content of this chapter is identical to that of the ML62Q1000 series. The ML7660 supports only one channel, limited to N=0. Only master mode is available.

6.1 Function Outline

This driver provides the functions for controlling the synchronous serial port. Use functions that use DMA only when CPU mode is wait mode.
Note: Please use HSCLK as System Clock when using SSIO.

6.2 API List (Channel: *N* = 0 to 5)

Below is the list of API Functions.
Note: Please refer to the Application Note for the availability of channels based on the number of pins.

6.2.1 SSIO*N*

Function Name	Function
void ssio*N*_putByte(uint8_t data)	Write 1-byte transmit data for SSIO*N*
void ssio*N*_putWord(uint16_t data)	Write 1-word (= 2-byte) transmit data for SSIO*N*
uint8_t ssio*N*_getByte(void)	Read 1-byte receive data for SSIO*N*
uint16_t ssio*N*_getWord(void)	Read 1-word (= 2-byte) receive data for SSIO*N*
void ssio*N*_disIntBeforeTransmit(void)	Disable SSIO*N* interrupt generation before transmission
void ssio*N*_enaIntBeforeTransmit(void)	Enable SSIO*N* interrupt generation before transmission
void ssio*N*_disIntBeforeReceive(void)	Disable SSIO*N* interrupt generation before reception
void ssio*N*_enaIntBeforeReceive(void)	Enable SSIO*N* interrupt generation before reception
void ssio*N*_setInterruptMode(uint8_t mode)	Sets interrupt generation mode
void ssio*N*_getReceiveFull(void)	Gets SSIO*N* receive buffer full
void ssio*N*_getTransmissionUnderrun(void)	Gets SSIO*N* Transmission Underrun error status
void ssio*N*_getTransmissionOverrun(void)	Gets SSIO*N* Transmission Overrun error status
void ssio*N*_getReceptionOverrun(void)	Gets SSIO*N* Reception Overrun error status
void ssio*N*_clrTransmissionUnderrun(void)	Clears SSIO*N* Transmission Underrun error status
void ssio*N*_clrTransmissionOverrun(void)	Clears SSIO*N* Transmission Overrun error status
void ssio*N*_clrReceptionOverrun(void)	Clears SSIO*N* Reception Overrun error status
void ssio*N*_getErrorStatus(void)	Gets SSIO*N* Error Status
void ssio*N*_clrErrorStatus(void)	Clears SSIO*N* Error Status
uint8_t ssio*N*_waitReceptionEnd(void)	Waits for SSIO*N* Reception completion
uint8_t ssio*N*_getTransferClock(void)	Gets SSIO*N* Transfer clock
void ssio*N*_setInterval(uint8_t cnt)	Sets SSIO*N* transmission interval
void ssio*N*_start(void)	Starts SSIO*N* communication
void ssio*N*_stop(void)	Stops SSIO*N* communication
uint8_t ssio*N*_getReceiveStatus(void)	Gets SSIO*N* receive status
uint8_t ssio*N*_getTransmitStatus(void)	Gets SSIO*N* transmit status
uint8_t ssio*N*_getTransmitFull(void)	Gets SSIO*N* transmit buffer full
uint8_t ssio*N*_getBitLength(void)	Gets SSIO*N* transfer bit length
uint8_t ssio*N*_getStat(void)	Gets SSIO*N* transmission, reception and transmit data buffer status
void ssio*N*_init(initSsio_t *st_initSsio)	Initializes the synchronous serial port channel *N*
int16_t ssio*N*_communicate(uint8_t trmod, void *rxData, void *txData, uint16_t size, cbfSsio_t func)	Setup SSIO*N* communication
int16_t ssio*N*_communicateDma(uint8_t trmod, void *rxData, void *txData, uint16_t size, cbfSsio_t func)	Setup SSIO*N* communication for DMA
int16_t ssio*N*_continue(void)	Continue SSIO*N* communication
void ssio*N*_transmitDmaEnd(uint8_t status)	SSIO*N* transmit end callback function for DMA
void ssio*N*_receiveDmaEnd(uint8_t status)	SSIO*N* receive end call back function for DMA
void ssio*N*_getCtrlParam(ssioCtrlParam_t *Param)	Gets SSIO*N* internal parameter for communication

6.3 Constant List

The following table shows the list of constants used by the API functions.

[Constants for setting]

Constant name	Defined value	Description
SSIO_NEG_POSITIVE	0x0	Positive clock pole
SSIO_NEG_NEGATIVE	0x1	Negative clock pole
SSIO_CKT_HIGH	0x0	Transfer clock phase: High
SSIO_CKT_LOW	0x1	Transfer clock phase: Low
SSIO_CK_LSCLK	0x00	1/1 LSCLK transfer clock
SSIO_CK_LSCLK_DIV2	0x01	1/2 LSCLK transfer clock
SSIO_CK_HSCLK	0x10	1/1 HSCLK transfer clock
SSIO_CK_HSCLK_DIV2	0x11	1/2 HSCLK transfer clock
SSIO_CK_HSCLK_DIV4	0x12	1/4 HSCLK transfer clock
SSIO_CK_HSLCK_DIV8	0x13	1/8 HSCLK transfer clock
SSIO_CK_HSLCK_DIV16	0x14	1/16 HSCLK transfer clock
SSIO_CK_HSLCK_DIV32	0x15	1/32 HSCLK transfer clock
SSIO_CK_HSLCK_DIV64	0x16	1/64 HSCLK transfer clock
SSIO_CK_HSLCK_DIV128	0x17	1/128 HSCLK transfer clock
SSIO_CK_EXTERNAL	0x18	External transfer clock
SSIO_LG_8BIT	0x0	8-bit length
SSIO_LG_16BIT	0x1	16-bit length
SSIO_MD_STOP	0x0	Stop mode
SSIO_MD_RECEIVE	0x1	Receive mode
SSIO_MD_TRANSMIT	0x2	Transmit mode
SSIO_MD_TR	0x3	Transmit/receive mode
SSIO_DIR_LSB	0x0	LSB first
SSIO_DIR_MSB	0x1	MSB first

[Constants for DMA status]

Constant name	Defined value	Description
SSIO_DMA_FIN	0	DMA status: complete
SSIO_DMA_STOP	1	DMA status: stop

[Constants for bit fields]

Constant name	Defined value	Description
SSIO_RECEIVE_BIT	0x01	Receive status bit
SSIO_TRANSMIT_BIT	0x02	Transmit status bit
SSIO_TR_BIT	0x03	Transmit-Receive Status bit
SSIO_16BIT_BIT	0x04	Bit length status bit
SSIO_RECEIVE_STOP_BIT	0x10	Receive stop status bit
SSIO_TRANSMIT_STOP_BIT	0x20	Transmit stop status bit
SSIO_RXF	0x20	Data reception status bit
SSIO_TXF	0x10	Data transmission status bit
SSIO_FUL	0x08	Transmit data buffer status bit
SSIO_DEVICE_CLK_SLAVE	0x18	Slave device clock
SSIO_RFUL	0x40	Receive data buffer status bit
SSIO_RECEIVE_EXE_BIT	0x80	Receive execution status bit

[Constants for return]

Constant name	Defined value	Description
SSIO_COMMUNICATION_OK	0x00	Communication continue
SSIO_COMMUNICATION_FIN	0x01	Communication finish
SSIO_ERROR_SIZE	0x02	Communication size error
SSIO_RECEPTION_OER	0x02	Reception overrun Error
SSIO_RECEPTION_END	0x01	Reception finished
SSIO_RECEPTION_WAIT	0x00	Reception continue

[Constants for interrupt modes]

Constant name	Defined value	Description
SSIO_INTMODE_ENDTX_ENDRX	0x00	Interrupt at the END of Transmission and Interrupt at the END of Reception
SSIO_INTMODE_STARTTX_ENDRX	0xC0	Interrupt at the START of Transmission and Interrupt at the END of Reception
SSIO_INTMODE_STARTTX_STARTRX	0xE0	Interrupt at the START of Transmission and Interrupt at the START of Reception

[Constants for operation state]

Constant name	Defined value	Description
SSIO_COMMUNICATION_END	0x00	Communication end
SSIO_RECEIVE_8BIT	0x01	Receive 8bit mode
SSIO_TRANSMIT_8BIT	0x02	Transmit 8bit mode
SSIO_TR_8BIT	0x03	Transmit and receive 8bit mode
SSIO_RECEIVE_16BIT	0x05	Receive 16bit mode
SSIO_TRANSMIT_16BIT	0x06	Transmit 16bit mode
SSIO_TR_16BIT	0x07	Transmit and receive 16bit mode
SSIO_RECEIVE_STOP	0x10	Receive stop
SSIO_TRANSMIT_STOP	0x20	Transmit stop
SSIO_TR_STOP	0x30	Transmit and receive stop
SSIO_TRANSMIT_8BIT_STOP	0x21	Transmit stop and Receive 8bit mode
SSIO_RECEIVE_8BIT_STOP	0x12	Receive stop and Transmit 8bit mode
SSIO_TRANSMIT_16BIT_STOP	0x25	Transmit stop and Receive 16bit mode
SSIO_RECEIVE_16BIT_STOP	0x16	Receive stop and Transmit 16bit mode
SSIO_ERROR_TUER	0x41	Transmission Underrun Error
SSIO_ERROR_ROER	0x42	Reception Overrun Error
SSIO_ERROR_TOER	0x43	Transmission Overrun Error
SSIO_ERROR_BIT	0x40	Error status bit
ERROR_BIT[a]	0x40	Error status bit

[a] = Constant is only for backward compatibility.

6.4 Structure and Typedef List

The following table shows the list of structures and typedef used by the API functions.

Structure Name	Member	Description
initSsio_t	uint8_t clockPole	Transfer clock pole SSIO_NEG_POSITIVE SSIO_NEG_NEGATIVE
	uint8_t clockType	Transfer clock type SSIO_CKT_HIGH SSIO_CKT_LOW
	uint8_t clock	Transfer clock SSIO_CK_LSCLK SSIO_CK_LSCLK_DIV2 SSIO_CK_HSCLK SSIO_CK_HSCLK_DIV2 SSIO_CK_HSCLK_DIV4 SSIO_CK_HSCLK_DIV8 SSIO_CK_HSCLK_DIV16 SSIO_CK_HSCLK_DIV32 SSIO_CK_HSCLK_DIV64 SSIO_CK_HSCLK_DIV128 SSIO_CK_HSCLK_EXTERNAL
	uint8_t bitLength	Transfer bit length SSIO_LG_8BIT SSIO_LG_16BIT
	uint8_t direction	Transfer data direction SSIO_DIR_LSB SSIO_DIR_MSB
cbfSsio_t	typedef void (*cbfSsio_t)(uint16_t size, uint8_t status)	Callback function
ssioCtrlParam_t	void *rxData	Receive data storage pointer
	void *txData	Transmit data storage pointer
	uint16_t size	Transfer data size
	uint16_t cnt	Transferred data size
	cbfSsio_t callback	Callback function pointer
	uint8_t state	Read/Write operation state

6.5 Static Variable List

Below is the list of static variables

Variable name	Type	Description
st_ctrlParam	ssioCtrlParam_t	Parameter accessed in the driver

6.6 API Function Details (Channel: *N* = 0 to 5)

The details of API functions are described below.

6.6.1 SSIO*N*

The APIs describe in this section is used to configure SSIO in Master and Slave Mode.

6.6.1.1 ssio*N*_putByte Function

This function writes 1-byte data to the SSIO*N* transmit buffer.

Function name:	void ssio*N*_putByte(uint8_t data)
Argument:	data … 1-byte transmit data
Return value:	None
Process:	Writes the data value to the SD*N*BUFL register

6.6.1.2 ssio*N*_putWord Function

This function writes 1-word (= 2-byte) data to the SSIO*N* transmit buffer.

Function name:	void ssio*N*_putWord(uint16_t data)
Argument:	data … 2-byte transmit data
Return value:	None
Process:	Writes the data value to the SD*N*BUF register

6.6.1.3 ssio*N*_getByte Function

This function reads 1-byte data from the SSIO*N* receive buffer.

Function name:	uint8_t ssio*N*_getByte(void)
Argument:	None
Return value:	1-byte receive data
Process:	Reads 1 byte from the SD*N*BUFL register

6.6.1.4 ssio*N*_getWord Function

This function reads 1-word (= 2-byte) data from the SSIO*N* receive buffer.

Function name:	uint16_t ssio*N*_getWord(void)
Argument:	None
Return value:	2-byte receive data
Process:	Reads 2 bytes from the SD*N*BUF register

6.6.1.5 ssio*N*_disIntBeforeTransmit Function

This function disables SSIO*N* interrupt before transmission.

Function name:	void ssio*N*_disIntBeforeTransmit(void)
Argument:	None
Return value:	None
Process:	Clears the SU*N*TIMD bit of the SU*N*MOD register

6.6.1.6 ssio*N*_enaIntBeforeTransmit Function

This function enables SSIO*N* interrupt before transmission.

Function name:	void ssio*N*_enaIntBeforeTransmit(void)
Argument:	None
Return value:	None
Process:	Sets the SU*N*TIMD bit of the SU*N*MOD register

6.6.1.7 ssio*N*_disIntBeforeReceive Function

This function disables SSIO*N* interrupt before reception.

Function name:	void ssio*N*_disIntBeforeReceive(void)
Argument:	None
Return value:	None
Process:	Clears the SU*N*RIMD bit of the SU*N*MOD register

6.6.1.8 ssio*N*_enaIntBeforeReceive Function

This function enables SSIO*N* interrupt before reception..

Function name:	void ssio*N*_enaIntBeforeReceive(void)
Argument:	None
Return value:	None
Process:	Sets the SU*N*RIMD bit of the SU*N*MOD register

6.6.1.9 ssio*N*_setInterruptMode Function

This function sets the SSIO*N* interrupt mode. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	void ssio*N*_setInterruptMode (mode)
Argument:	mode… Interrupt mode SSIO_INTMODE_ENDTX_ENDRX SSIO_INTMODE_STARTTX_ENDRX SSIO_INTMODE_STARTTX_STARTRX
Return value:	None
Process:	Sets the SU*N*INTS, SU*N*RIMD and SU*N*TIMD bits of the SU*N*MOD register

6.6.1.10 ssio*N*_setInterval Function

This function sets SSIO*N* transmission interval.

Function name:	void ssio*N*_setInterval(uint8_t cnt)
Argument:	cnt… interval count
Return value:	None
Process:	Writes the value of cnt in SU*N*DLY register

6.6.1.11 ssio*N*_start Function

This function starts SSIO*N* communication.

Function name:	void ssio*N*_start(void)
Argument:	None
Return value:	None
Process:	Sets the S*N*EN bit of the SU*N*CON register

6.6.1.12 ssio*N*_stop Function

This function stops SSIO*N* communication.

Function name:	void ssio*N*_stop(void)
Argument:	None
Return value:	None
Process:	Clears the S*N*EN bit of the SU*N*CON register

6.6.1.13 ssio*N*_getReceiveStatus Function

This function gets the SSIO*N* receive status.

Function name:	uint8_t ssio*N*_getReceiveStatus(void)
Argument:	None
Return value:	0 … Stop receive 1 … Execute receive
Process:	Reads the S*N*RXF bit of the SIO*N*STAT register

6.6.1.14 ssio*N*_getTransmitStatus Function

This function gets the SSIO*N* transmit status.

Function name:	uint8_t ssio*N*_getTransmitStatus(void)
Argument:	None
Return value:	0 … Stop transmit 1 … Execute transmit
Process:	Reads the S*N*TXF bit of the SIO*N*STAT register

6.6.1.15 ssio*N*_getTransmitFull Function

This function checks if SSIO*N* buffer is empty or full.

Function name:	uint8_t ssio*N*_getTransmitFull(void)
Argument:	None
Return value:	0 … Buffer empty 1 … Buffer full
Process:	Reads the S*N*FUL bit of the SIO*N*STAT register

6.6.1.16 ssio*N*_getBitLength Function

This function gets the SSIO*N* transfer bit length.

Function name:	uint8_t ssio*N*_getBitLength(void)
Argument:	None
Return value:	0 … 8-bit length 1 … 16-bit length

Process:	Reads the S*N*LG bit of the SIO*N*MOD register

6.6.1.17 ssio*N*_getStat Function

This function gets the SSIO*N*, reception, transmission status, and the transmit data buffer status. It additionally gets the error status and the receive data buffer status.

Function name:	uint8_t ssio*N*_getStat(void)
Argument:	None
Return value:	status… reception, transmission and transmit data buffer status
Process:	Reads the SIO*N*STAT register

6.6.1.18 ssio*N*_getReceiveFull Function

This function gets the SSIO*N* receive data buffer status. Please refer to the Application Note for the availability of the channels and usage of this function

Function name:	uint8_t ssio*N*_getReceiveFull (void)
Argument:	None
Return value:	0 … Buffer empty 1 … Buffer full
Process:	Reads the S*N*RFUL bit

6.6.1.19 ssio*N*_getTransmissionUnderrun Function

This function gets the SSIO*N* Transmission underrun error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	void ssio*N*_getTransmissionUnderrun (void)
Argument:	None
Return value:	0… No transmission underrun error 1… Transmission underrun error
Process:	Reads the S*N*TUER bit

6.6.1.20 ssio*N*_getTransmissionOverrun Function

This function gets the SSIO*N* Transmission overrun error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	uint8_t ssio*N*_getTransmissionOverrun (void)
Argument:	None
Return value:	0… No transmission overrun error 1… Transmission overrun error
Process:	Reads the S*N*TOER bit

6.6.1.21 ssio*N*_getReceptionOverrun Function

This function gets the SSIO*N* Reception overrun error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	uint8_t ssio*N*_getReceptionOverrun (void)
Argument:	None
Return value:	0… No reception overrun error 1… Reception overrun error
Process:	Reads the S*N*ROER bit

6.6.1.22 ssio*N*_clrTransmissionUnderrun Function

This function clears the SSIO*N* Transmission error underrun status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	void ssio*N*_clrTransmissionUnderrun (void)
Argument:	None
Return value:	None
Process:	Writes 0x01 to SIO*N*STAT register to clear the S*N*TUER bit

6.6.1.23 ssio*N*_clrTransmissionOverrun Function

This function clears the SSIO*N* Transmission overrun error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	void ssio*N*_clrTransmissionOverrun void)
Argument:	None
Return value:	None
Process:	Writes 0x04 to SIO*N*STAT register to clear the S*N*TOER bit

6.6.1.24 ssio*N*_clrReceptionOverrun Function

This function clears the SSIO*N* Reception overrun error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	void ssio*N*_clrReceptionOverrun (void)
Argument:	None
Return value:	None
Process:	Writes 0x02 to SIO*N*STAT register to clear the S*N*ROER bit

6.6.1.25 ssio*N*_getErrorStatus Function

This function gets the SSIO*N* error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	uint8_t ssio*N*_getErrorStatus (void)
Argument:	None
Return value:	Error status…transmission underrun and overrun, reception overrun
Process:	Reads the S*N*TUER, S*N*TOER, and S*N*ROER bit

6.6.1.26 ssio*N*_clrErrorStatus Function

This function clears the SSIO*N* error status. Please refer to the Application Note for the availability of the channels and usage of this function.

Function name:	void ssio*N*_clrErrorStatus (void)
Argument:	None
Return value:	None
Process:	Writes 0x07 to SIO*N*STAT to clear the S*N*TUER, S*N*TOER, and S*N*ROER bit

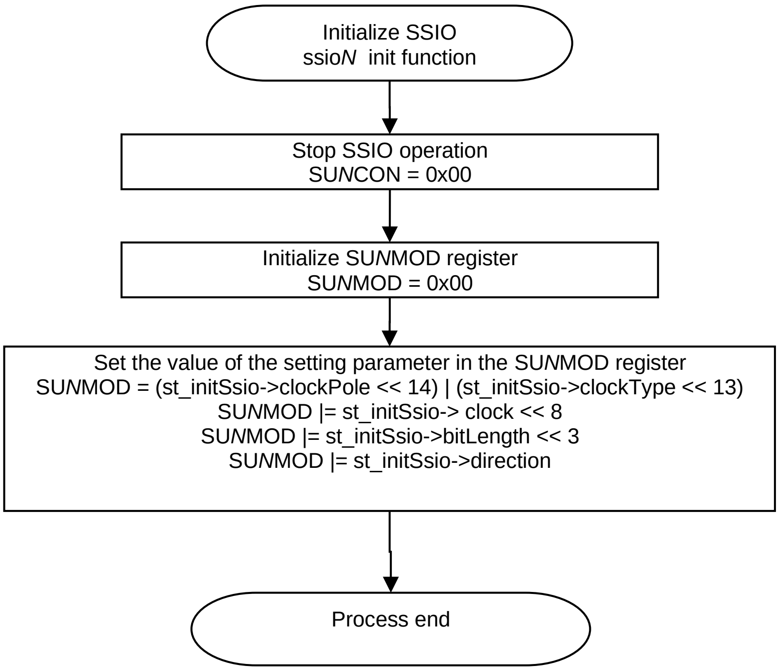
6.6.1.27 ssio*N*_getTransferClock Function

This function gets the SSIO*N* transfer clock value.

Function name:	uint8_t ssio*N*_getTransferClock (void)
Argument:	None
Return value:	Transfer clock…. Clock settings being used
Process:	Reads the value of S*N*CK4- S*N*CK0

6.6.1.28 ssioN_init Function

This function initializes the synchronous serial port channel *N*.

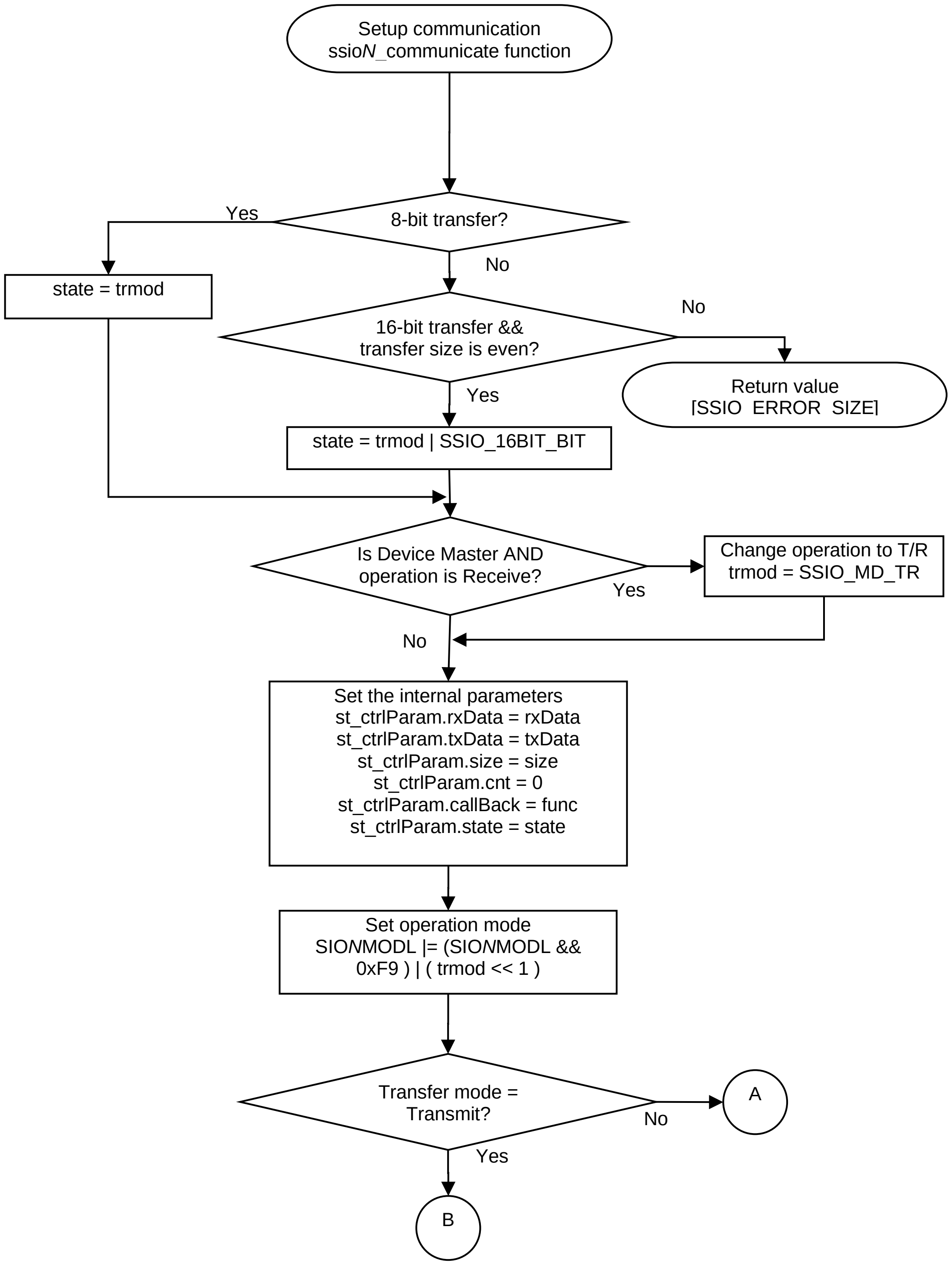
Function name:	void ssio*N*_init(initSsio_t *st_initSsio)
Argument:	initSsio_t *st_initSsio … Setting parameter initSsio_t structure
Return value:	None
Process:	 Initialize SSIO ssio*N* init function Stop SSIO operation SU*N*CON = 0x00 Initialize SU*N*MOD register SU*N*MOD = 0x00 Set the value of the setting parameter in the SU*N*MOD register SU*N*MOD = (st_initSsio->clockPole << 14) | (st_initSsio->clockType << 13) SU*N*MOD |= st_initSsio-> clock << 8 SU*N*MOD |= st_initSsio->bitLength << 3 SU*N*MOD |= st_initSsio->direction Process end

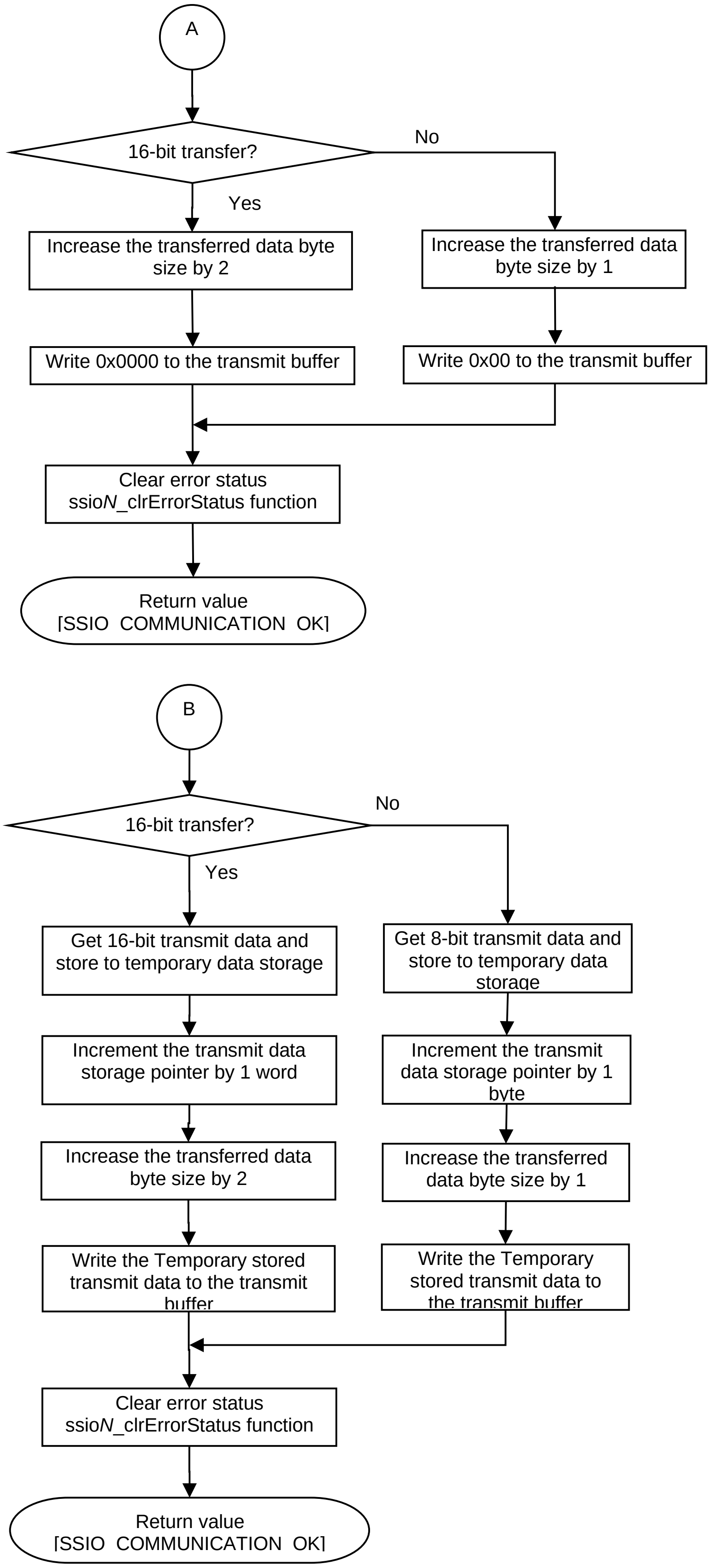
6.6.1.29 ssio*N*_communicate Function

This function sets up SSIO*N* communication.

Function name:	int16_t ssio*N*_communicate(uint8_t trmod, void *rxData, void *txData, uint16_t size, cbfSsio_t func)
Argument:	uint8_t trmod … Mode of operation SSIO_MD_RECEIVE SSIO_MD_TRANSMIT SSIO_MD_TR void *rxData … Receive data storage pointer (for 16-bit transfer, word address) void *txData … Transmit data storage pointer (for 16-bit transfer, word address) uint16_t size … Transfer byte size (for 16-bit transfer, even) cbfSsio_t func … Callback function pointer (set 0, if not necessary)
Return value:	SSIO_COMMUNICATION_OK … SSIO communication setup successful SSIO_ERROR_SIZE … SSIO communication error

Process:


Setup communication
ssioN_communicate function
8-bit transfer?
Yes
No
state = trmod
16-bit transfer &&
transfer size is even?
No
Yes
Return value
[SSIO ERROR SIZE]
state = trmod | SSIO_16BIT_BIT
Is Device Master AND
operation is Receive?
Yes
Change operation to T/R
trmod = SSIO_MD_TR
No
Set the internal parameters
st_ctrlParam.rxData = rxData
st_ctrlParam.txData = txData
st_ctrlParam.size = size
st_ctrlParam.cnt = 0
st_ctrlParam.callBack = func
st_ctrlParam.state = state
Set operation mode
SIONMODL |= (SIONMODL &&
0xF9) | (trmod << 1)
Transfer mode =
Transmit?
No
A
Yes
B

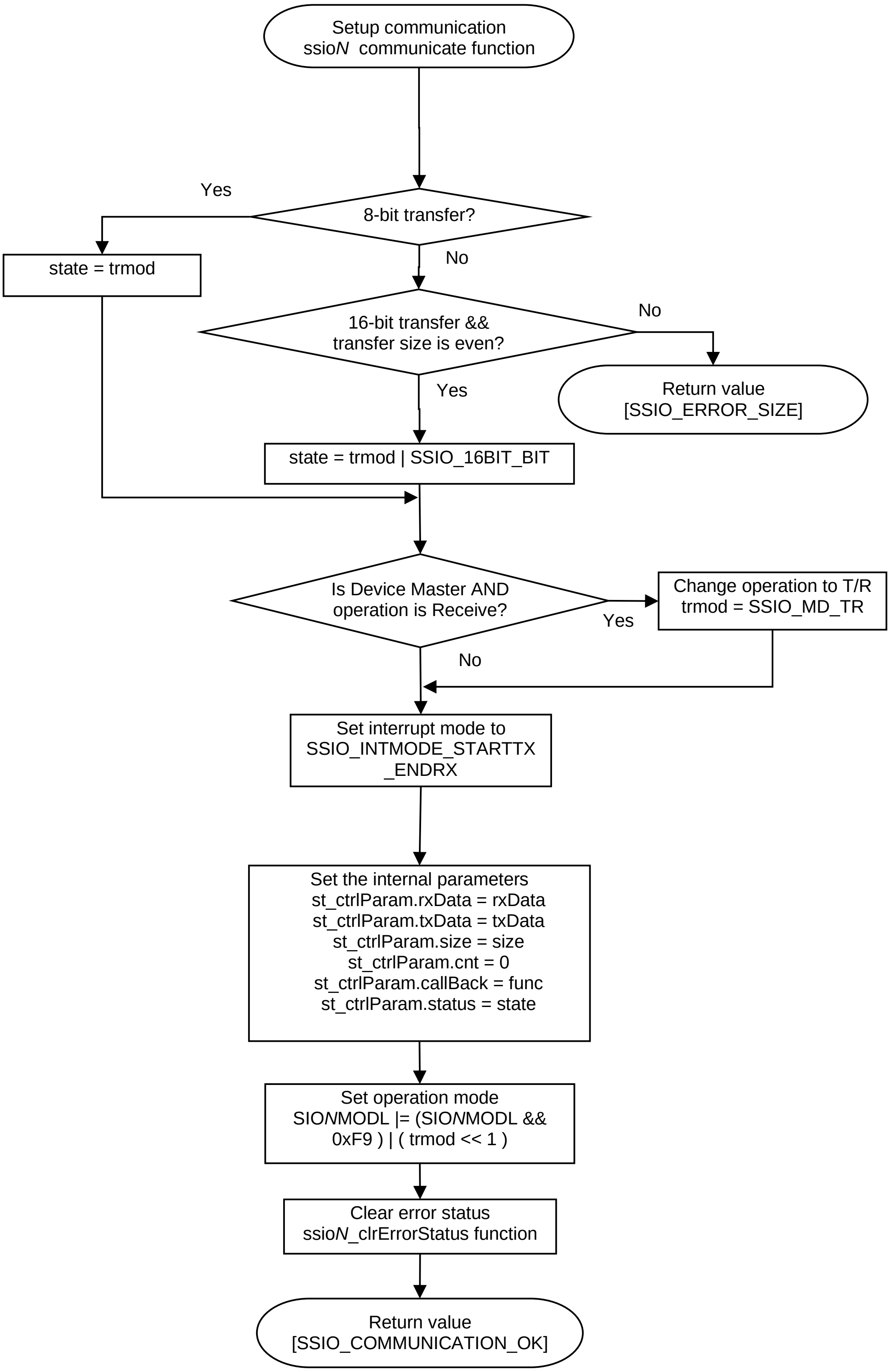

A
16-bit transfer?
No
Yes
Increase the transferred data byte size by 2
Increase the transferred data byte size by 1
Write 0x0000 to the transmit buffer
Write 0x00 to the transmit buffer
Clear error status ssioN_clrErrorStatus function
Return value [SSIO_COMMUNICATION_OK]
B
16-bit transfer?
No
Yes
Get 16-bit transmit data and store to temporary data storage
Get 8-bit transmit data and store to temporary data storage
Increment the transmit data storage pointer by 1 word
Increment the transmit data storage pointer by 1 byte
Increase the transferred data byte size by 2
Increase the transferred data byte size by 1
Write the Temporary stored transmit data to the transmit buffer
Write the Temporary stored transmit data to the transmit buffer
Clear error status ssioN_clrErrorStatus function
Return value [SSIO_COMMUNICATION_OK]

6.6.1.30 ssio*N*_communicateDma Function

This function sets up SSIO*N* communication for DMA.

Function name:	int16_t ssio*N*_communicateDma(uint8_t trmod, void *rxData, void *txData, uint16_t size, cbfSsio_t func)
Argument:	uint8_t trmod … Mode of operation SSIO_MD_RECEIVE SSIO_MD_TRANSMIT SSIO_MD_TR void *rxData … Receive data storage pointer (for 16-bit transfer, word address) void *txData … Transmit data storage pointer (for 16-bit transfer, word address) uint16_t size … Transfer byte size (for 16-bit transfer, even) cbfSsio_t func … Callback function pointer (set 0, if not necessary)
Return value:	SSIO_COMMUNICATION_OK … SSIO communication setup successful SSIO_ERROR_SIZE … SSIO communication error

Process:


Setup communication
ssio*N* communicate function
Yes
8-bit transfer?
No
state = trmod
16-bit transfer &&
transfer size is even?
No
Yes
Return value
[SSIO_ERROR_SIZE]
state = trmod | SSIO_16BIT_BIT
Is Device Master AND
operation is Receive?
Yes
Change operation to T/R
trmod = SSIO_MD_TR
No
Set interrupt mode to
SSIO_INTMODE_STARTTX
_ENDRX
Set the internal parameters
st_ctrlParam.rxData = rxData
st_ctrlParam.txData = txData
st_ctrlParam.size = size
st_ctrlParam.cnt = 0
st_ctrlParam.callBack = func
st_ctrlParam.status = state
Set operation mode
SIO*N*MODL |= (SIO*N*MODL &&
0xF9) | (trmod << 1)
Clear error status
ssio*N*_clrErrorStatus function
Return value
[SSIO_COMMUNICATION_OK]

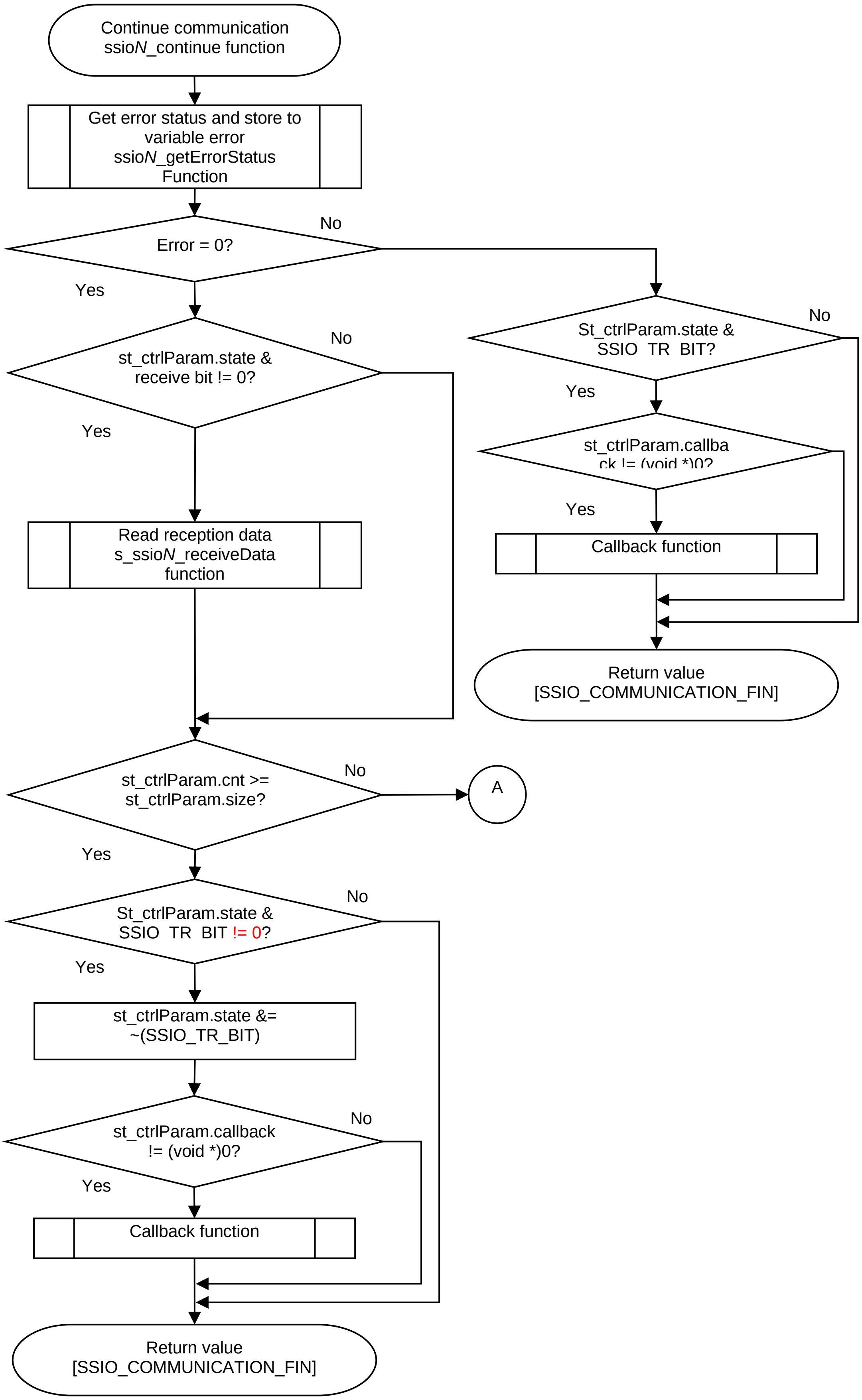
6.6.1.31 ssio*N*_continue Function

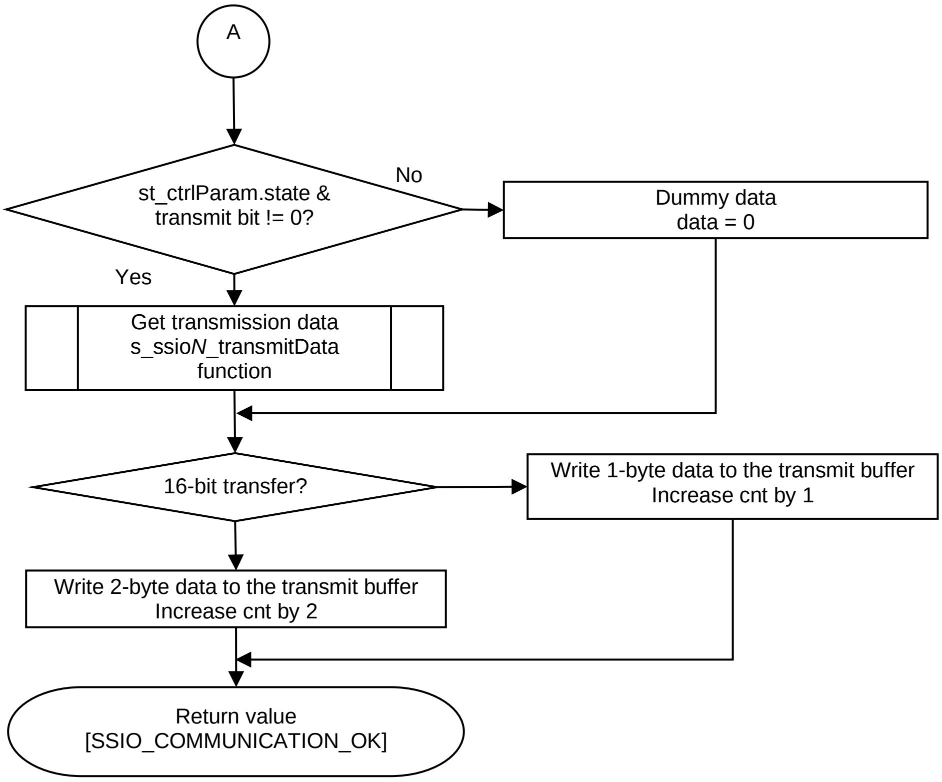
This function continues transfer.

Note: This function is optimized for speed.

Function name:	int16_t ssio*N*_continue(void)
Argument:	None
Return value:	SSIO_COMMUNICATION_OK … Communication is continued (success) SSIO_COMMUNICATION_FIN … Communication is finished

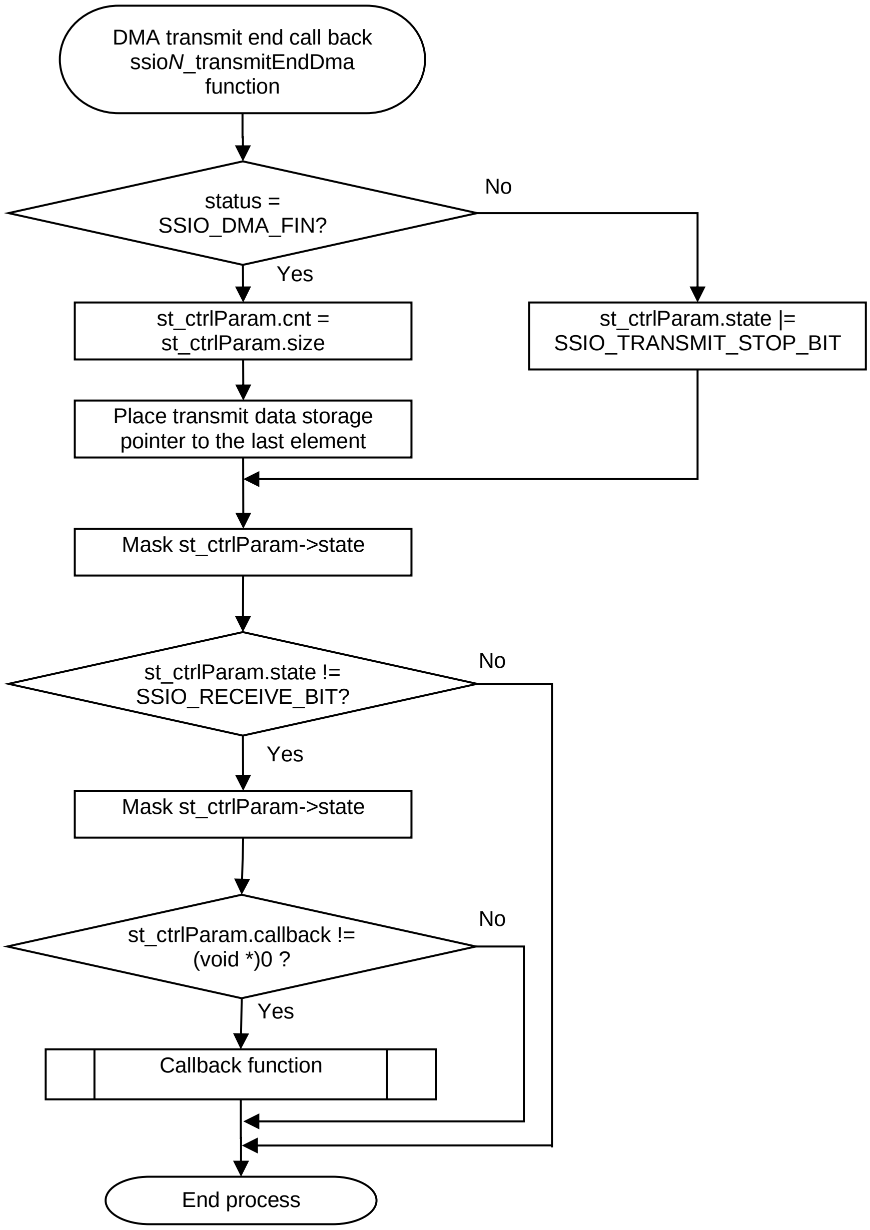
Process:


Continue communication
ssioN_continue function
Get error status and store to
variable error
ssioN_getErrorStatus
Function
Error = 0?
No
Yes
st_ctrlParam.state &
receive bit != 0?
No
Yes
Read reception data
s_ssioN_receiveData
function
St_ctrlParam.state &
SSIO TR BIT?
No
Yes
st_ctrlParam.callba
ck != (void *)0?
Yes
Callback function
Return value
[SSIO_COMMUNICATION_FIN]
st_ctrlParam.cnt >=
st_ctrlParam.size?
No
A
Yes
St_ctrlParam.state &
SSIO TR BIT != 0?
No
Yes
st_ctrlParam.state &=
~(SSIO_TR_BIT)
st_ctrlParam.callback
!= (void *)0?
No
Yes
Callback function
Return value
[SSIO_COMMUNICATION_FIN]


A
st_ctrlParam.state &
transmit bit != 0?
No
Dummy data
data = 0
Yes
Get transmission data
s_ssio*N*_transmitData
function
16-bit transfer?
Write 1-byte data to the transmit buffer
Increase cnt by 1
Write 2-byte data to the transmit buffer
Increase cnt by 2
Return value
[SSIO_COMMUNICATION_OK]

6.6.1.32 ssio*N*_transmitDmaEnd Function

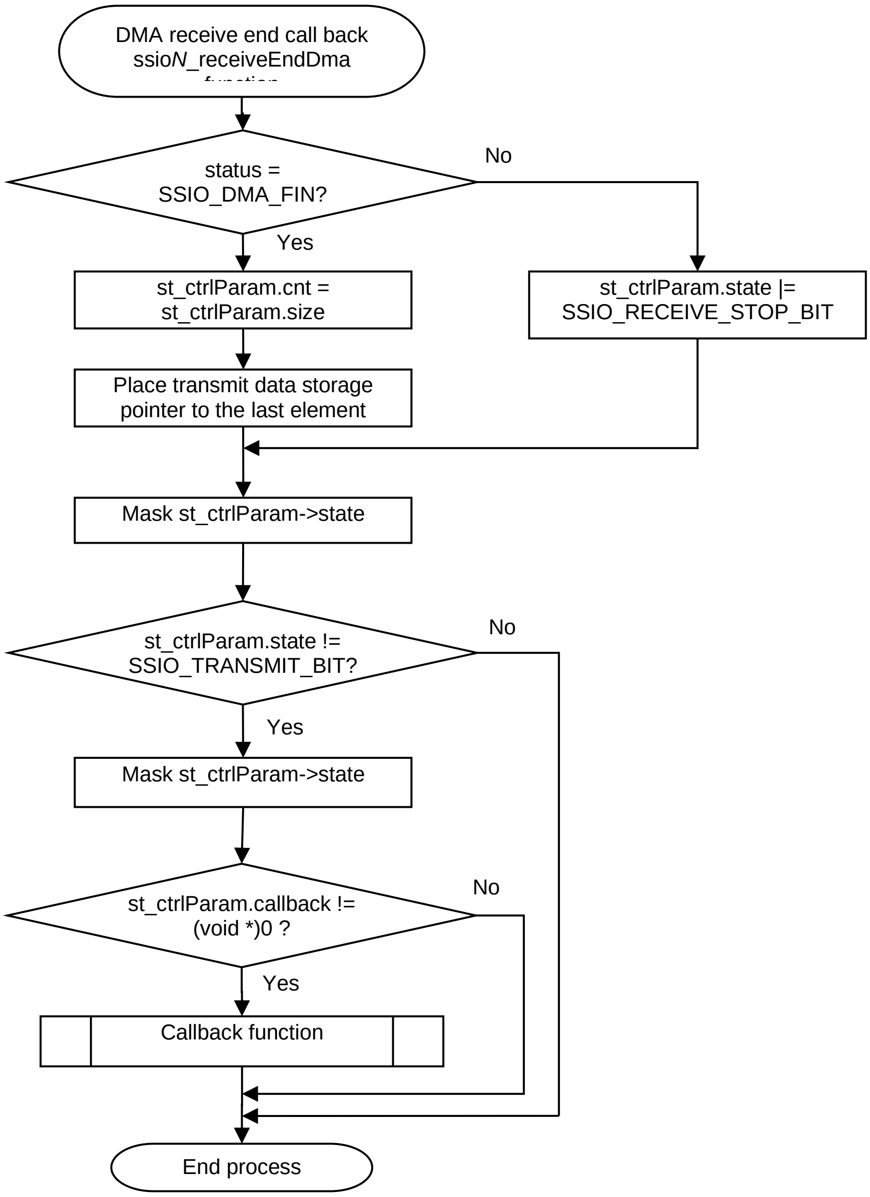
This function serves as the call back function for transmit end for DMA.

Function name:	void ssio*N*_transmitDmaEnd(uint8_t status)
Argument:	status… Dma status SSIO_DMA_FIN SSIO_DMA_STOP
Return value:	None
Process:	

DMA transmit end call back ssioN_transmitEndDma function
status = SSIO_DMA_FIN?
No
Yes
st_ctrlParam.cnt = st_ctrlParam.size
st_ctrlParam.state \|= SSIO_TRANSMIT_STOP_BIT
Place transmit data storage pointer to the last element
Mask st_ctrlParam->state
st_ctrlParam.state != SSIO_RECEIVE_BIT?
No
Yes
Mask st_ctrlParam->state
st_ctrlParam.callback != (void *)0 ?
No
Yes
Callback function
End process

6.6.1.33 ssio*N*_receiveDmaEnd Function

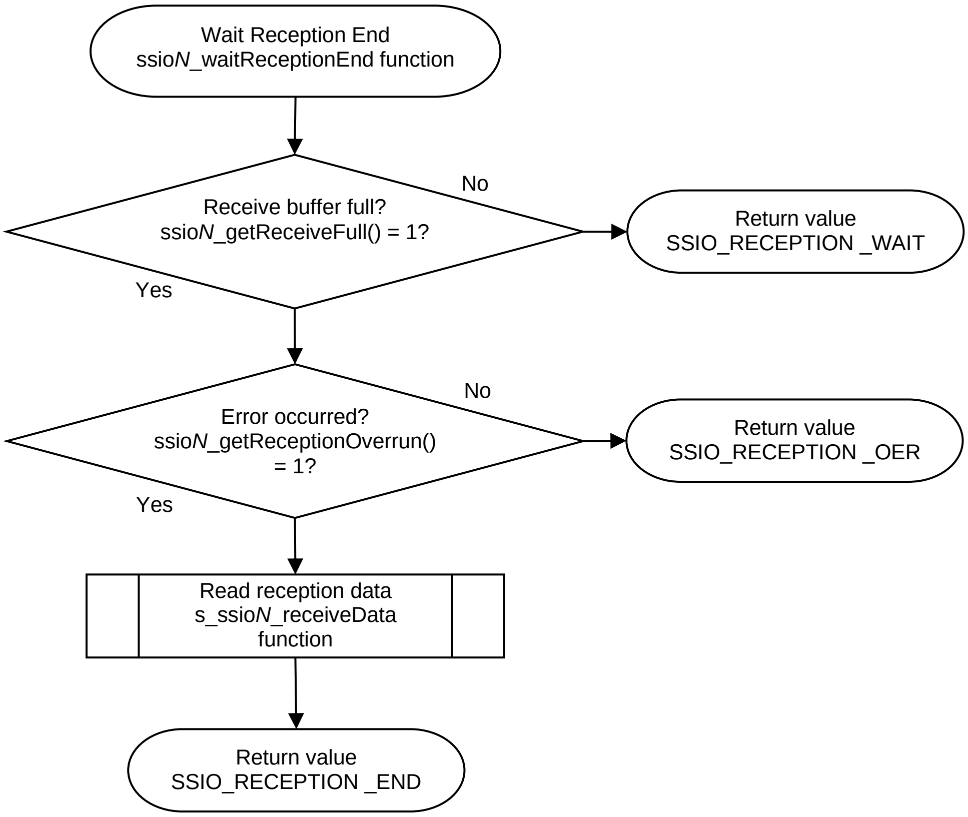
This function serves as the call back function for receive end for DMA.

Function name:	void ssio*N*_receiveDmaEnd(uint8_t status)
Argument:	status… Dma status SSIO_DMA_FIN SSIO_DMA_STOP
Return value:	None
Process:	 DMA receive end call back ssio*N*_receiveEndDma function status = SSIO_DMA_FIN? No Yes st_ctrlParam.cnt = st_ctrlParam.size st_ctrlParam.state |= SSIO_RECEIVE_STOP_BIT Place transmit data storage pointer to the last element Mask st_ctrlParam->state st_ctrlParam.state != SSIO_TRANSMIT_BIT? No Yes Mask st_ctrlParam->state st_ctrlParam.callback != (void *)0 ? No Yes Callback function End process

6.6.1.34 ssio*N*_waitReceptionEnd Function

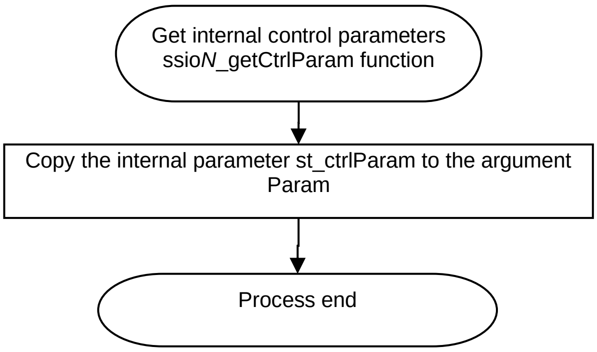
This function waits for the completion of reception of final the data by polling.

Note: This function is applicable only for Transmit-Receive or Master-Receive.

Function name:	uint8_t ssio*N*_ waitReceptionEnd (void)
Argument:	None
Return value:	SSIO_RECEPTION_WAIT… wait for reception completion SSIO_RECEPTION_END … reception finish SSIO_RECEPTION_OER… reception overrun error occured
Process:	 Wait Reception End ssio*N*_waitReceptionEnd function Receive buffer full? ssio*N*_getReceiveFull() = 1? No Return value SSIO_RECEPTION _WAIT Yes Error occurred? ssio*N*_getReceptionOverrun() = 1? No Return value SSIO_RECEPTION _OER Yes Read reception data s_ssio*N*_receiveData function Return value SSIO_RECEPTION _END

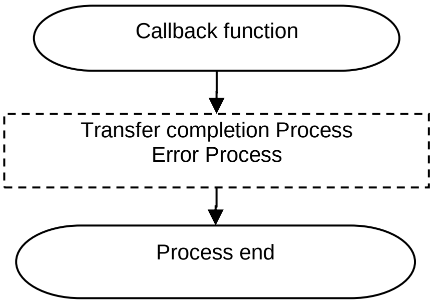
6.6.1.35 ssio*N*_getCtrlParam Function

This function gets the internal parameter for communication.

Function name:	void ssio*N*_getCtrlParam(ssioCtrlParam_t *Param)
Argument:	ssioCtrlParam_t *Param … Copy destination pointer of internal parameter
Return value:	None
Process:	 Get internal control parameters ssio*N*_getCtrlParam function Copy the internal parameter st_ctrlParam to the argument Param Process end

6.6.1.36 Callback Function

This function is called from the driver when the transfer is completed or when an error occurs.
If 0 is set in the callback function argument of the ssio*N*_communicate/ssio*N*_communicateDma function, no callback function is called.

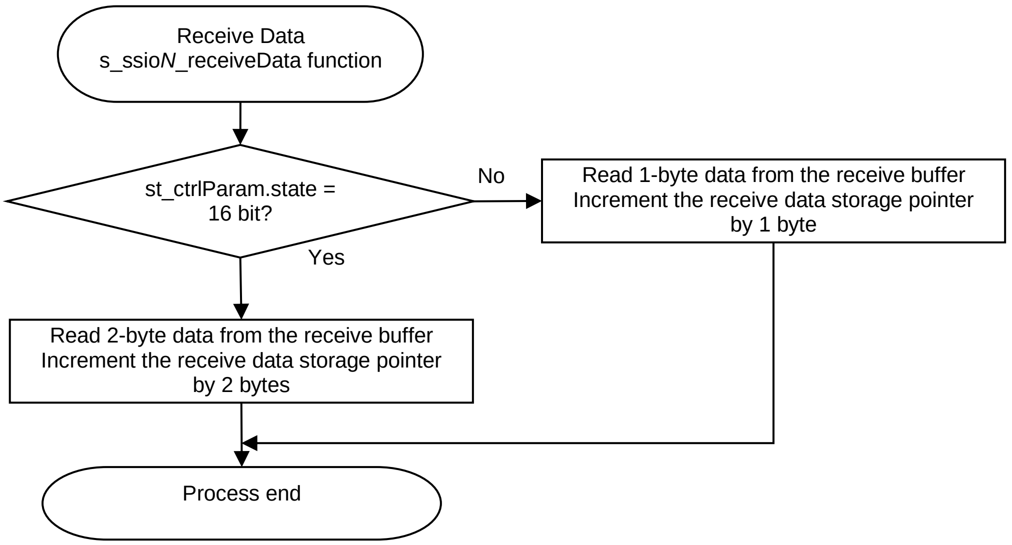
Function name:	void (*cbfSsio_t)(uint16_t size, uint8_t status)
Argument:	size … Transferred byte data size status… Communication status SSIO_COMMUNICATION_END SSIO_RECEIVE_8BIT SSIO_TRANSMIT_8BIT SSIO_TR_8BIT SSIO_RECEIVE_16BIT SSIO_TRANSMIT_16BIT SSIO_TR_16BIT SSIO_RECEIVE_STOP SSIO_TRANSMIT_STOP SSIO_TR_STOP SSIO_TRANSMIT_8BIT_STOP SSIO_RECEIVE_8BIT_STOP SSIO_TRANSMIT_16BIT_STOP SSIO_RECEIVE_16BIT_STOP
Return value:	None
Process:	 Callback function Transfer completion Process Error Process Process end

6.7 Static Function List

Below is the list of static functions.

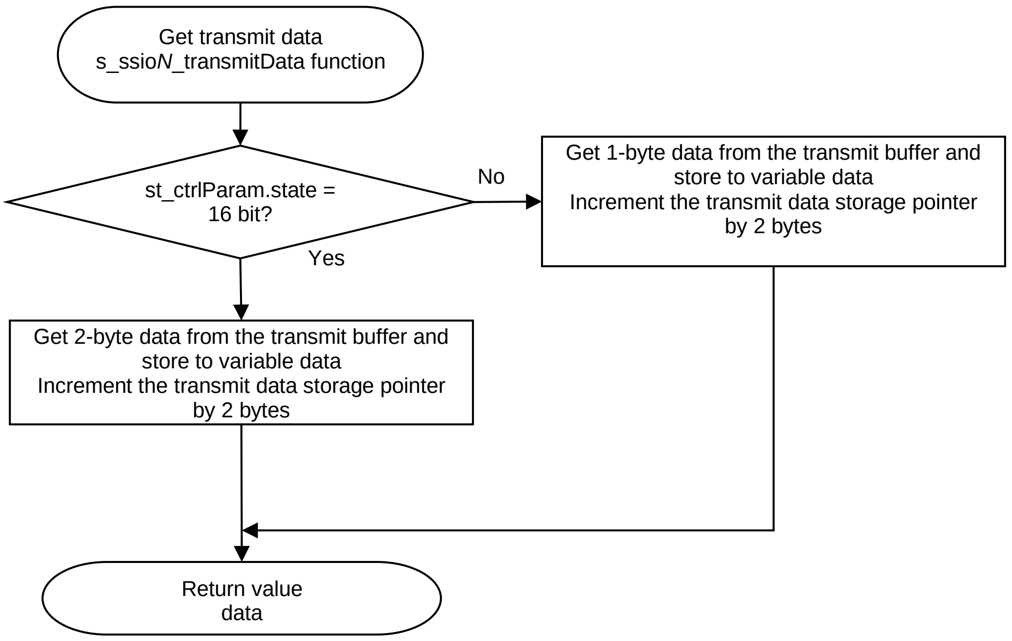
6.7.1 s_ssio*N*_receiveData

This function is an inline function that is used to read reception data from the buffer.

Function name:	static void s_ssio*N*_receiveData(void)
Argument:	None
Return value:	None
Process:	 Receive Data s_ssio*N*_receiveData function st_ctrlParam.state = 16 bit? No Read 1-byte data from the receive buffer Increment the receive data storage pointer by 1 byte Yes Read 2-byte data from the receive buffer Increment the receive data storage pointer by 2 bytes Process end

6.7.2 s_ssio*N*_transmitData

This function is an inline function that is used to get transmission data from the buffer.

Function name:	static uint16_t s_ssio*N*_transmitData(void)
Argument:	None
Return value:	Data… transmit data
Process:	 Get transmit data s_ssio*N*_transmitData function st_ctrlParam.state = 16 bit? No Get 1-byte data from the transmit buffer and store to variable data Increment the transmit data storage pointer by 2 bytes Yes Get 2-byte data from the transmit buffer and store to variable data Increment the transmit data storage pointer by 2 bytes Return value data

7. SDK Sample Software

As a sample application for the rotational reference design, we provide SDK sample software using the STREAL SR300 sensors. This SDK sample software acquires strain and temperature values from the STREAL SR300 sensors and sends the data to the Tx Host.

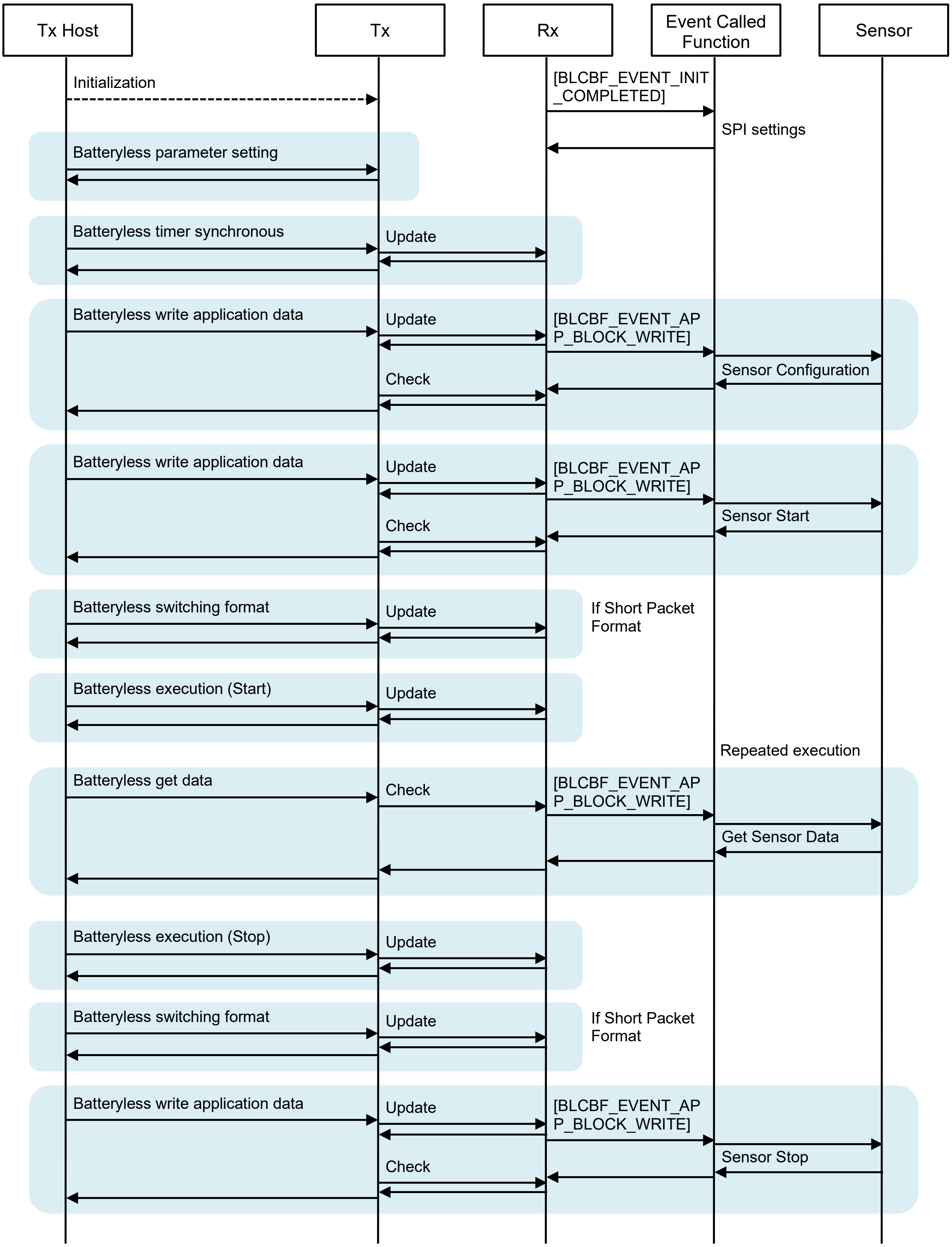
7.1 File Structure

The file structure of this SDK sample software is described below.

Folder / File Names	Description
BL_SDK_SampleApp_Rx_SR300	SDK sample software folder using STREAL SR300 sensors.
.cproject	Project file for LEXIDE
.project	
BL_SDK_SampleApp_Rx_SR300 Debug Debug.launch	Debug configuration file for LEXIDE
include	Folder containing header files
drvdefs.h	Header file for selecting definitions used by the driver
mcu.h	Header file of target device
rdwr_reg.h	Header file for accessing registers
ssio0.h	API definition header file of ssio0 (SPI) driver
ssio_common.h	Header file of ssio0 (SPI) driver.
stdint_t.h	Type definition header file
Library	Library folder
BatteryLess_Lib_Listener.lib	Rx Library file
SampleApp	Sample application folder
blapi.h	API definition header file of Rx Library
main.c	Source file of main routine
ml766xblsample.c	Files related to the execution of the Event Called Function
ml766xblsample.h	
spi.c	Files that control the SPI driver
spi.h	
sr300sample.c	Files that control STREAL SR300 sensors
sr300sample.h	
DeviceInformation	Device information files for ML7661/ML7660
LEXIDE	
Dcl	Device information files
Inc	Device information files
Startup	Device information files
Trg	Device information files

7.2 Overall Sequence

This section presents the overall sequence of the sample application.

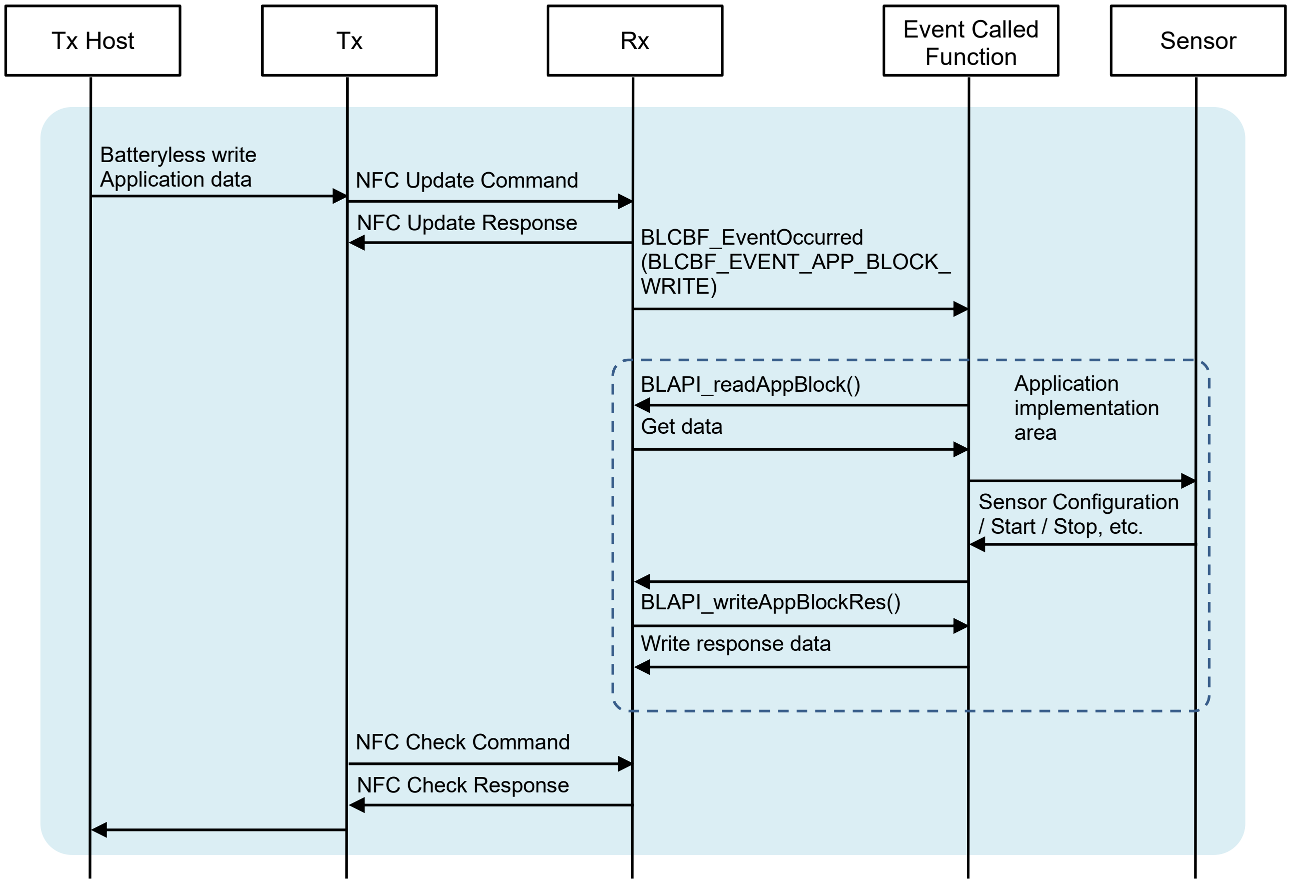

Tx Host
Tx
Rx
Event Called Function
Sensor
Initialization
[BLCBF_EVENT_INIT_COMPLETED]
SPI settings
Batteryless parameter setting
Batteryless timer synchronous
Update
Batteryless write application data
Update
[BLCBF_EVENT_APP_BLOCK_WRITE]
Sensor Configuration
Check
Batteryless write application data
Update
[BLCBF_EVENT_APP_BLOCK_WRITE]
Sensor Start
Check
Batteryless switching format
Update
If Short Packet Format
Batteryless execution (Start)
Update
Repeated execution
Batteryless get data
Check
[BLCBF_EVENT_APP_BLOCK_WRITE]
Get Sensor Data
Batteryless execution (Stop)
Update
Batteryless switching format
Update
If Short Packet Format
Batteryless write application data
Update
[BLCBF_EVENT_APP_BLOCK_WRITE]
Sensor Stop
Check

7.3 Detailed Sequence for Application Data Writing Command

In this sample application, the Tx Host performs the following three write operations:

Sensor Configuration
Sensor Start
Sensor Stop

In the Event Called Function of Rx, application data is received and processed for the sensors.



7.4 Application Data Format

Application data format is as follows. Please use the APP_BLOCK area to exchange data.

7.4.1 Sensor Configuration

<table>
<tr><th>Byte No</th><th>Description</th></tr>
<tr><td>0</td><td>0x00</td></tr>
<tr><td>1</td><td>Target Channel Information
<table>
<tr><td>Bit7</td><td>Bit6</td><td>Bit5</td><td>Bit4</td><td>Bit3</td><td>Bit2</td><td>Bit1</td><td>Bit0</td></tr>
<tr><td>Unused</td><td>Unused</td><td>Unused</td><td>Unused</td><td>Unused</td><td>Unused</td><td>Channel 1</td><td>Channel 0</td></tr>
</table></td></tr>
<tr><td>2－8</td><td>Sensor Configuration value for Channel 0 (ADR000 to ADR110)</td></tr>
<tr><td>9－15</td><td>Sensor Configuration value for Channel 1 (ADR000 to ADR110)</td></tr>
</table>

7.4.2 Sensor Start

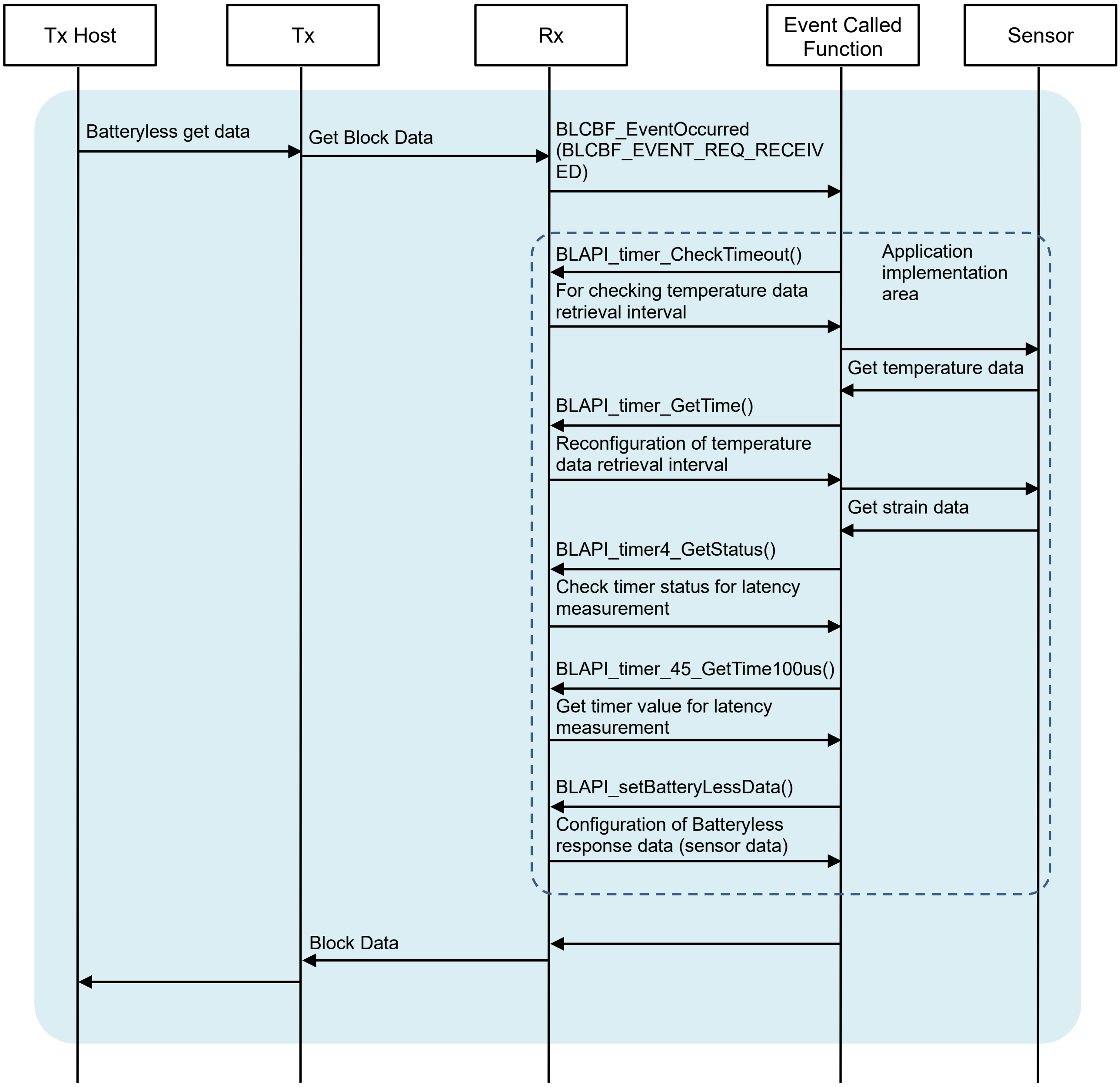
<table>
<tr><th>Byte No</th><th>Description</th></tr>
<tr><td>0</td><td>0x01</td></tr>
<tr><td>1</td><td>Measurement Information for Channel 0
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x00</td><td>No acquisition</td></tr>
<tr><td>0x01</td><td>Acquisition of strain data</td></tr>
<tr><td>0x02</td><td>Acquisition of temperature data</td></tr>
<tr><td>0x03</td><td>Acquisition of both strain and temperature data</td></tr>
</table></td></tr>
<tr><td>2</td><td>Measurement Information for Channel 1
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x00</td><td>No acquisition</td></tr>
<tr><td>0x01</td><td>Acquisition of strain data</td></tr>
<tr><td>0x02</td><td>Acquisition of temperature data</td></tr>
<tr><td>0x03</td><td>Acquisition of both strain and temperature data</td></tr>
</table></td></tr>
<tr><td>3－15</td><td>Unused</td></tr>
</table>

7.4.3 Sensor Stop

Byte No	Description
0	0x02
1－15	Unused

7.5 Detailed Sequence for Batteryless get data Command

The data acquired from the sensor by the Rx Event Called Function will be transmitted to the Tx Host according to the following sequence.


Tx Host
Tx
Rx
Event Called Function
Sensor
Batteryless get data
Get Block Data
BLCBF_EventOccurred (BLCBF_EVENT_REQ_RECEIVED)
BLAPI_timer_CheckTimeout()
Application implementation area
For checking temperature data retrieval interval
Get temperature data
BLAPI_timer_GetTime()
Reconfiguration of temperature data retrieval interval
Get strain data
BLAPI_timer4_GetStatus()
Check timer status for latency measurement
BLAPI_timer_45_GetTime100us()
Get timer value for latency measurement
BLAPI_setBatteryLessData()
Configuration of Batteryless response data (sensor data)
Block Data

7.6 Get Data Format (Short Packet Format)

The format of the sensor data sent to the Tx Host is as follows.

When temperature data is available:

Byte No	Description
0－3	Sensor data acquisition time 32-bit unsigned integer in little-endian format. (If the timer synchronization command is omitted, Byte 0-3 will be omitted, improving the communication rate.)
4－5	Strain Data of Channel 0 (* Sensor Data Format)
6－7	Strain Data of Channel 1 (* Sensor Data Format)
8－9	Temperature Data of Channel 0 (* Sensor Data Format)
10－11	Temperature Data of Channel 1 (* Sensor Data Format)
12	Data Validity Information (* Data Validity Information Format)

When temperature data is not available:

Byte No	Description
0－3	Sensor data acquisition time 32-bit unsigned integer in little-endian format. (If the timer synchronization command is omitted, Byte 0-3 will be omitted, improving the communication rate.)
4－5	Strain Data of Channel 0 (* Sensor Data Format)
6－7	Strain Data of Channel 1 (* Sensor Data Format)
8	Data Validity Information (* Data Validity Information Format)

(* Sensor Data Format)

<table>
<tr><th>Byte No</th><th>Bit7</th><th>Bit6</th><th>Bit5</th><th>Bit4</th><th>Bit3</th><th>Bit2</th><th>Bit1</th><th>Bit0</th></tr>
<tr><td>1st byte</td><td colspan="4"><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x0</td><td>Normal</td></tr>
<tr><td>0x1</td><td>SPI communication failure</td></tr>
<tr><td>0x2</td><td>Data format error</td></tr>
<tr><td>0xF</td><td>Data acquisition not started</td></tr>
</table></td><td colspan="4">Sensor Value (D11 to D8)</td></tr>
<tr><td>2nd byte</td><td colspan="8">Sensor Value (D7 to D0)</td></tr>
</table>

<table>
<tr><th>Byte No</th><th>Bit7</th><th>Bit6</th><th>Bit5</th><th>Bit4</th><th>Bit3</th><th>Bit2</th><th>Bit1</th><th>Bit0</th></tr>
<tr><td>1st byte</td><td colspan="4"><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x0</td><td>Normal</td></tr>
<tr><td>0x1</td><td>SPI communication failure</td></tr>
<tr><td>0x2</td><td>Data format error</td></tr>
<tr><td>0xF</td><td>Data acquisition not started</td></tr>
</table></td><td colspan="4">Sensor Value (D11 to D8)</td></tr>
<tr><td>2nd byte</td><td colspan="8">Sensor Value (D7 to D0)</td></tr>
</table>

(* Data Validity Information Format)

<table>
<tr><th>Byte No</th><th>Bit7</th><th>Bit6</th><th>Bit5</th><th>Bit4</th><th>Bit3</th><th>Bit2</th><th>Bit1</th><th>Bit0</th></tr>
<tr><td>1st byte</td><td colspan="6">Unused</td><td>Presence of valid data since the last get data command<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>None</td></tr><tr><td>1</td><td>Present</td></tr></table></td><td>Status when last acquired from Listener<table><tr><th>Value</th><th>Description</th></tr><tr><td>0x0</td><td>Failure</td></tr><tr><td>0x1</td><td>Success</td></tr></table></td></tr>
</table>

7.7 Get Data Format (T3T Format)

Byte No	Description
0－3	Sensor data acquisition time 32-bit unsigned integer in little-endian format. (If the timer synchronization command is omitted, Byte 0-3 will be omitted, improving the communication rate.)
4－5	Strain Data of Channel 0 (* Sensor Data Format)
6－7	Strain Data of Channel 1 (* Sensor Data Format)
8－9	Temperature Data of Channel 0 (* Sensor Data Format)
10－11	Temperature Data of Channel 1 (* Sensor Data Format)
12	Data Validity Information (* Data Validity Information Format)
13－15	Unused

(* Sensor Data Format)

<table>
<tr><th>Byte No</th><th>Bit7</th><th>Bit6</th><th>Bit5</th><th>Bit4</th><th>Bit3</th><th>Bit2</th><th>Bit1</th><th>Bit0</th></tr>
<tr><td>1st byte</td><td colspan="4"><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x0</td><td>Normal</td></tr>
<tr><td>0x1</td><td>SPI communication failure</td></tr>
<tr><td>0x2</td><td>Data format error</td></tr>
<tr><td>0xF</td><td>Data acquisition not started</td></tr>
</table></td><td colspan="4">Sensor Value (D11 to D8)</td></tr>
<tr><td>2nd byte</td><td colspan="8">Sensor Value (D7 to D0)</td></tr>
</table>

(* Data Validity Information Format)

<table>
<tr><th>Byte No</th><th>Bit7</th><th>Bit6</th><th>Bit5</th><th>Bit4</th><th>Bit3</th><th>Bit2</th><th>Bit1</th><th>Bit0</th></tr>
<tr><td>1st byte</td><td colspan="6">Unused</td><td>Presence of valid data since the last get data command<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>None</td></tr>
<tr><td>1</td><td>Present</td></tr>
</table></td><td>Status when last acquired from Listener<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x0</td><td>Failure</td></tr>
<tr><td>0x1</td><td>Success</td></tr>
</table></td></tr>
</table>

8. Software Development using SDK Sample Software

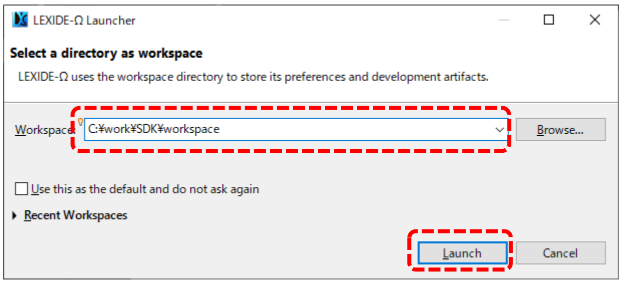
Software development using the SDK sample software makes use of the integrated development environment LEXIDE-Ω. This chapter provides an overview of how to start software development using LEXIDE-Ω.

For detailed instructions, please refer to the various manuals of LEXIDE-Ω.

Step-1. Install the LEXIDE-Ω.

Copy the Device Information Files (*) into the installed folder (**).
(*): The four folders under ROHM_BatteyrLess_SDK_Vxxx¥Firmware¥Rx¥DeviceInformation¥LEXIDE.
(**): Default path is C:¥LAPIS¥LEXIDE.
“Vxxx” shows version number.

Step-2. Launch the installed LEXIDE-Ω and create a workspace by specifying an appropriate folder.


LEXIDE-Ω Launcher

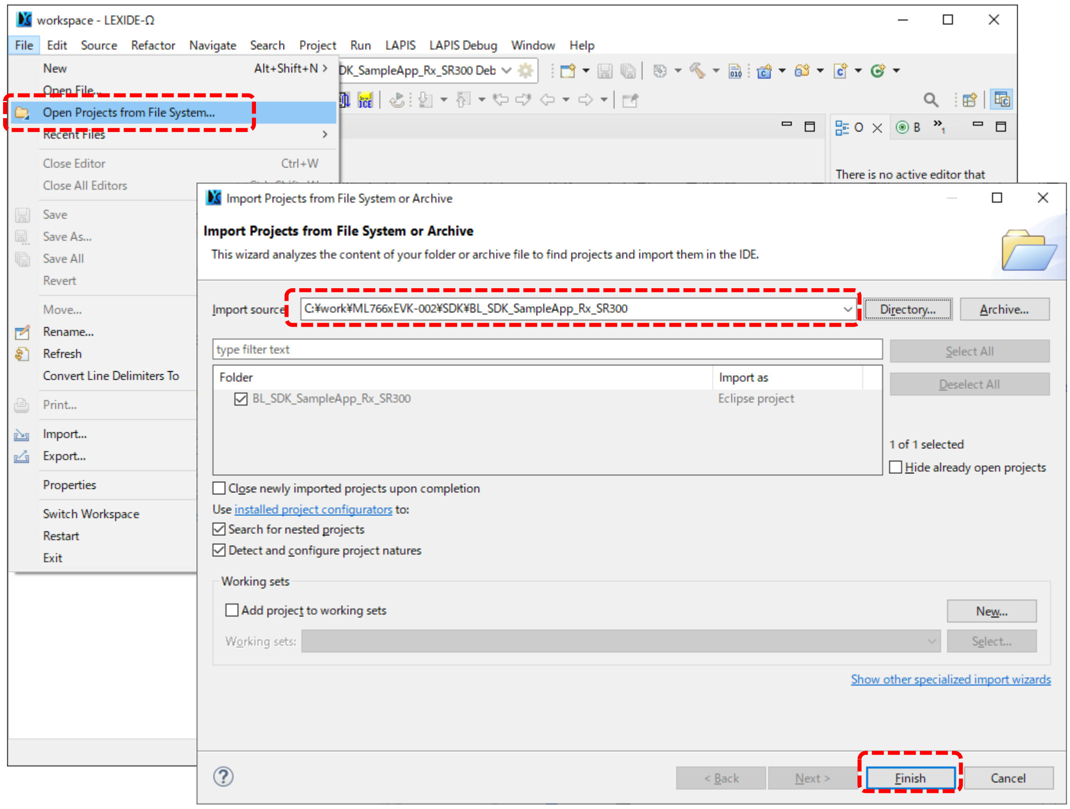
Select a directory as workspace

LEXIDE-Ω uses the workspace directory to store its preferences and development artifacts.

Workspace: C:¥work¥SDK¥workspace

□ Use this as the default and do not ask again

Step-3. Import the SDK sample software project.


workspace - LEXIDE-Ω

Import Projects from File System or Archive

Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: C:¥work¥ML766xEVK-002¥SDK¥BL_SDK_SampleApp_Rx_SR300

type filter text

Folder	Import as
☑ BL_SDK_SampleApp_Rx_SR300	Eclipse project

1 of 1 selected

□ Hide already open projects

□ Close newly imported projects upon completion

Use installed project configurators to:

☑ Search for nested projects

☑ Detect and configure project natures

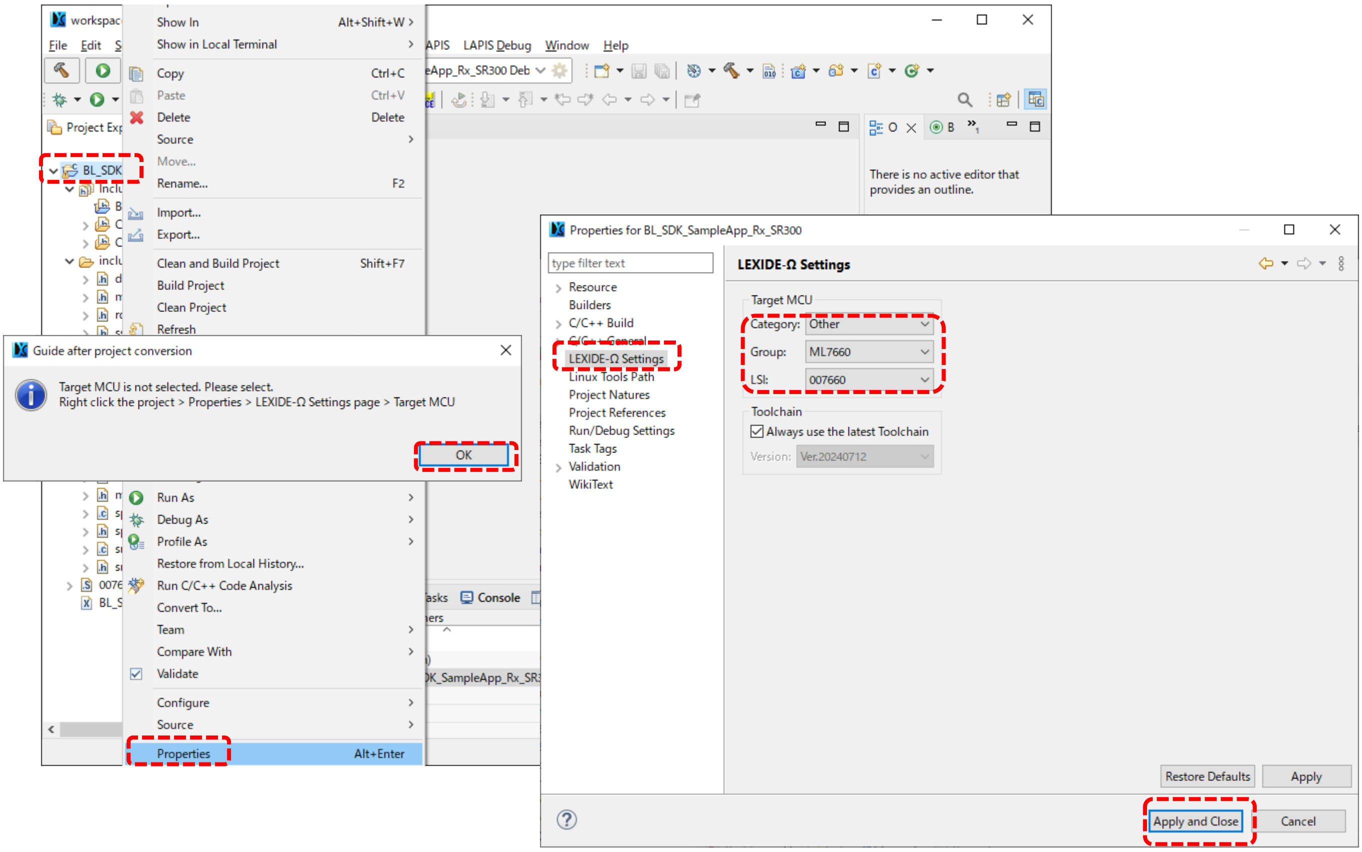
Working sets

□ Add project to working sets

Working sets:

There is no active editor that

Step-4. Set the target from the project Properties settings.



Guide after project conversion

Target MCU is not selected. Please select.
Right click the project > Properties > LEXIDE-Ω Settings page > Target MCU

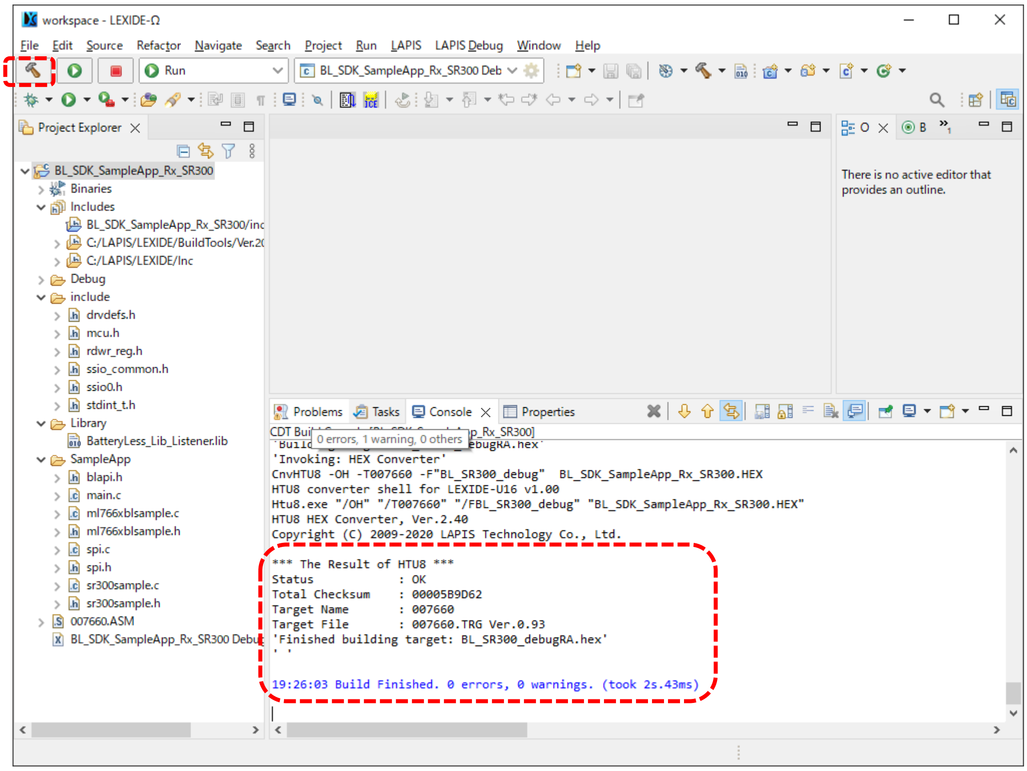
Properties for BL_SDK_SampleApp_Rx_SR300

LEXIDE-Ω Settings

Target MCU
Category: Other
Group: ML7660
LSI: 007660

Toolchain
☑ Always use the latest Toolchain
Version: Ver.20240712

Step-5. Build and verify that there no errors.



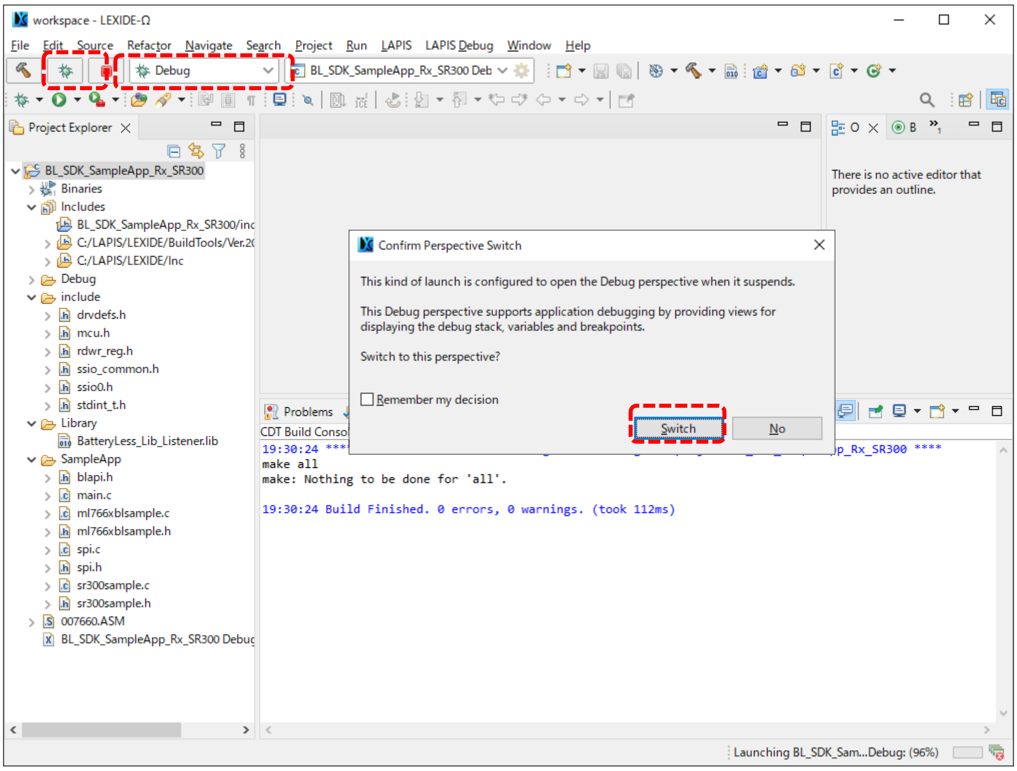
```
'Invoking: HEX Converter'
CnvHTU8 -OH -T007660 -F"BL_SR300_debug"  BL_SDK_SampleApp_Rx_SR300.HEX
HTU8 converter shell for LEXIDE-U16 v1.00
Htu8.exe "/OH" "/T007660" "/FBL_SR300_debug" "BL_SDK_SampleApp_Rx_SR300.HEX"
HTU8 HEX Converter, Ver.2.40
Copyright (C) 2009-2020 LAPIS Technology Co., Ltd.

*** The Result of HTU8 ***
Status            : OK
Total Checksum    : 00005B9D62
Target Name       : 007660
Target File       : 007660.TRG Ver.0.93
'Finished building target: BL_SR300_debugRA.hex'
' '

19:26:03 Build Finished. 0 errors, 0 warnings. (took 2s.43ms)
```

Step-6. Connect the EASE100V2 to the Rx board, download the above build results to the chip, and start debugging.

The Rx side operates by receiving a magnetic field supply from the Tx side. Therefore, please debug the following operations with the Tx side activated and supplying the magnetic field.


workspace - LEXIDE-Ω
File Edit Source Refactor Navigate Search Project Run LAPIS LAPIS Debug Window Help
Debug
BL_SDK_SampleApp_Rx_SR300 Deb
Project Explorer
BL_SDK_SampleApp_Rx_SR300
Binaries
Includes
BL_SDK_SampleApp_Rx_SR300/inc
C:/LAPIS/LEXIDE/BuildTools/Ver.2(
C:/LAPIS/LEXIDE/Inc
Debug
include
drvdefs.h
mcu.h
rdwr_reg.h
ssio_common.h
ssio0.h
stdint_t.h
Library
BatteryLess_Lib_Listener.lib
SampleApp
blapi.h
main.c
ml766xblsample.c
ml766xblsample.h
spi.c
spi.h
sr300sample.c
sr300sample.h
007660.ASM
BL_SDK_SampleApp_Rx_SR300 Debuc
There is no active editor that provides an outline.
Problems
CDT Build Consol
Launching BL_SDK_Sam...Debug: (96%)

Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.

Switch to this perspective?

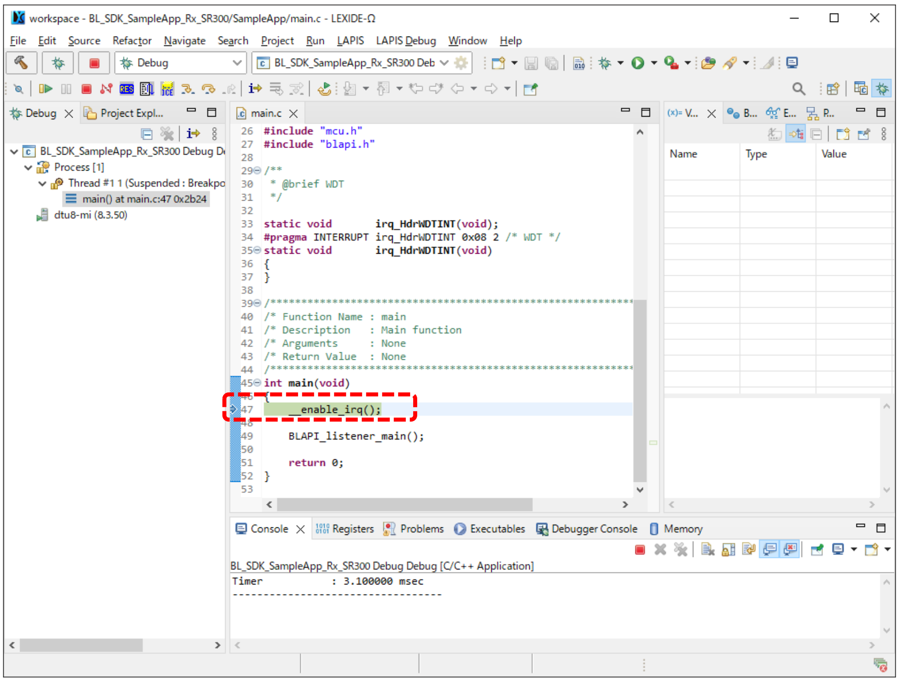
□ Remember my decision

Switch   No

```
19:30:24 ****                                       p_Rx_SR300 ****
make all
make: Nothing to be done for 'all'.

19:30:24 Build Finished. 0 errors, 0 warnings. (took 112ms)
```

Step-7. The debugging state will be as follows.


workspace - BL_SDK_SampleApp_Rx_SR300/SampleApp/main.c - LEXIDE-Ω
File Edit Source Refactor Navigate Search Project Run LAPIS LAPIS Debug Window Help
Debug
BL_SDK_SampleApp_Rx_SR300 Deb
Debug
Project Expl...
BL_SDK_SampleApp_Rx_SR300 Debug D
Process [1]
Thread #1 1 (Suspended : Breakpo
main() at main.c:47 0x2b24
dtu8-mi (8.3.50)
main.c
Name
Type
Value
Console
Registers
Problems
Executables
Debugger Console
Memory

```
#include "mcu.h"
#include "blapi.h"

/**
 * @brief WDT
 */

static void        irq_HdrWDTINT(void);
#pragma INTERRUPT irq_HdrWDTINT 0x08 2 /* WDT */
static void        irq_HdrWDTINT(void)
{
}

/*************************************************************
/* Function Name : main
/* Description   : Main function
/* Arguments     : None
/* Return Value  : None
/*************************************************************
int main(void)
{
    __enable_irq();

    BLAPI_listener_main();

    return 0;
}
```

BL_SDK_SampleApp_Rx_SR300 Debug Debug [C/C++ Application]

```
Timer           : 3.100000 msec
----------------------------------
```

9. Supplementary Information

When using the SDK with sensors, the parameter for the Batteryless execution command can only be set to ‘synchronous’ mode.
When using the SDK with sensors, it is recommended to use the Short Packet Format for communication when acquiring sensor data.

10. Revision History

No.	Date	Page		Descriptions
		Previous Edition	Current Edition	
1	Feb 13, 2025	-	-	First edition issued

Notice

Precaution on using ROHM Products

1) When using ROHM Products, refer to the latest product information and ensure that usage conditions (absolute maximum ratings[*1], recommended operating conditions, etc.) are within the ranges specified. ROHM disclaims any and all liability for any malfunctions, failure or accident arising out of or in connection with the use of ROHM Products outside of such usage conditions specified ranges, or without observing precautions. Even if it is used within such usage conditions specified ranges, semiconductors can break down and malfunction due to various factors. Therefore, in order to prevent personal injury, fire or the other damage from break down or malfunction of ROHM Products, please take safety at your own risk measures such as complying with the derating characteristics, implementing redundant and fire prevention designs, and utilizing backups and fail-safe procedures.
   *1: Absolute maximum ratings: a limit value that must not be exceeded even momentarily.

2) The Products specified in this document are not designed to be radiation tolerant.

3) Descriptions of circuits, software and other related information in this document are provided only to illustrate the standard operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. And the peripheral conditions must be taken into account when designing circuits for mass production. ROHM disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, and other related information.

4) No license, expressly or implied, is granted hereby under any intellectual property rights or other rights of ROHM or any third party with respect to ROHM Products or the information contained in this document (including but not limited to, the Product data, drawings, charts, programs, algorithms, and application examples, etc.). Therefore, ROHM shall have no responsibility whatsoever for any dispute, concerning such rights owned by third parties, arising out of the use of such technical information.

5) ROHM intends our Products to be used in a way indicated in this document. Please be sure to contact a ROHM sales office if you consider the use of our Products in different way from original use indicated in this document. For use of our Products in medical systems, please be sure to contact a ROHM representative and must obtain written agreement. Do not use our Products in applications which may directly cause injuries to human life, and which require extremely high reliability, such as aerospace equipment, nuclear power control systems, and submarine repeaters, etc. ROHM disclaims any and all liability for any losses and damages incurred by you or third parties arising by using the Product for purposes not intended by us without our prior written consent.

6) Please use the Products in accordance with any applicable environmental laws and regulations, such as the RoHS Directive. ROHM shall have no responsibility for any damages or losses resulting non-compliance with any applicable laws or regulations.

7) When providing our Products and technologies contained in this document to other countries, you must abide by the procedures and provisions stipulated in all applicable export laws and regulations, including without limitation the US Export Administration Regulations and the Foreign Exchange and Foreign Trade Act.

8) Please contact a ROHM sales office if you have any questions regarding the information contained in this document or ROHM's Products.

9) This document, in part or in whole, may not be reprinted or reproduced without prior consent of ROHM.

(Note) “ROHM” as used in this document means ROHM Co., Ltd.

Other Precaution

1) All information contained in this document is subject to change for the purpose of improvement, etc. without any prior notice. Before purchasing or using ROHM Products, please confirm the latest information with a ROHM sales office.

2) ROHM has used reasonable care to ensure the accuracy of the information contained in this document, however, ROHM shall have no responsibility for any damages, expenses or losses arising from inaccuracy or errors of such information.

TSZ72037・01・001