CC3100 UART Host Interface

Contents

Introduction

Host low power modes

UART Host Topologies
- 5-Wire UART Topology
- 4-Wire UART Topology
- 3-Wire UART Topology

UART Configuration

UART Initialization

Calculating the host maximum working baud rate

Changing the UART baud rate

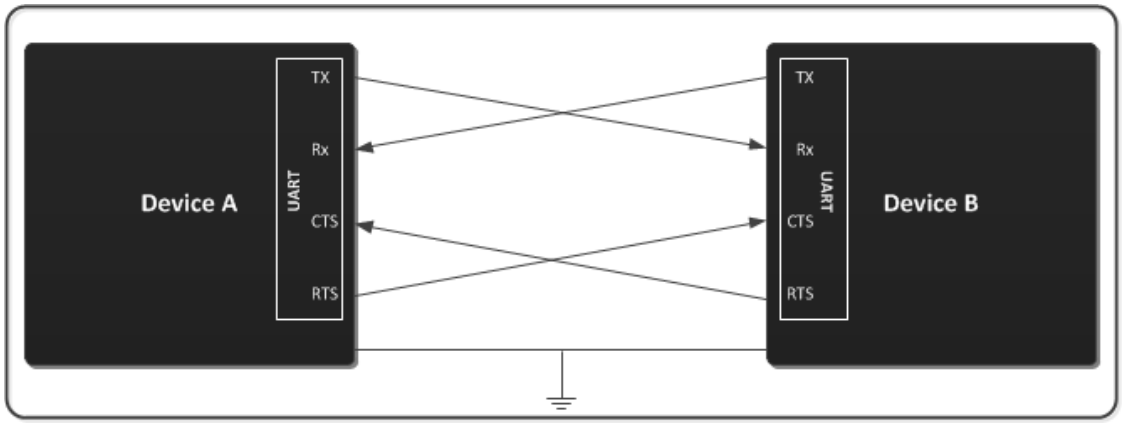
Implementing the UART Driver - Concept & Terminology
- UART Read API implementation
- UART Write API implementation

Register/Unregister Interrupt Handler API implementation
- Host Interface protocol – UART perspective
- Uart host command flow
- Synchronization Words

Introduction

The UART is a standard asynchronous serial communication that works between two entities and have a support for hardware flow control. In UART interface there is no Master/Slave relationship defined by the Hardware and each entity can send data to the other side independently in full duplex mode. The hardware flow control makes use of two hardware lines, RTS (Request to Send) and CTS (Clear to Send) to allow each side indicate to the other side if is ready to handle data.

The figure below illustrates a typical UART setup:



Typical UART Configuration

The perspectives of the lines' names further in this document are from the Host to the SimpleLink device:

TX – used to send the UART serial data from the Host to the CC3100 device
RX – used to send the UART serial data from the CC3100 device to the Host
RTS – used to instruct the CC3100 device to stop sending data (Host cannot handle more data)
CTS – used to instruct the Host to stop sending data (The SimpleLink device cannot handle more data)

Host low power modes

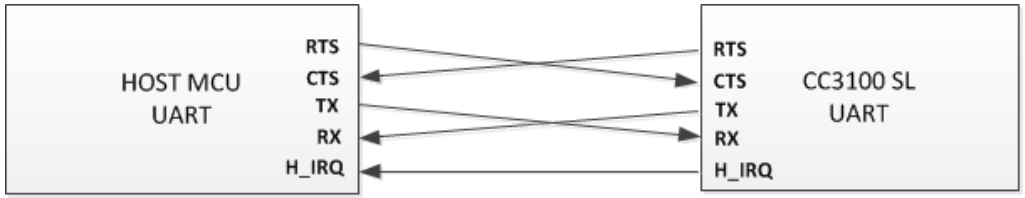
The SimpleLink device can send a message to the Host at any given time. The SimpleLink host protocol does not allow data loss. When the Host enters into a low power mode it must raise the RTS line to signal the SimpleLink device that it can't receive data.

However, when RTS line is raised, the SimpleLink device will not be able to wake up the Host. In this case, to allow the Host to wake up by the SimpleLink device, the auxiliary HOST_IRQ line should be used.

UART Host Topologies

5-Wire UART Topology

The following figure shows the typical 5-wire UART topology which is comprised of 4 standard UART lines, plus one IRQ line from the device to the host controller to allow efficient low power mode:

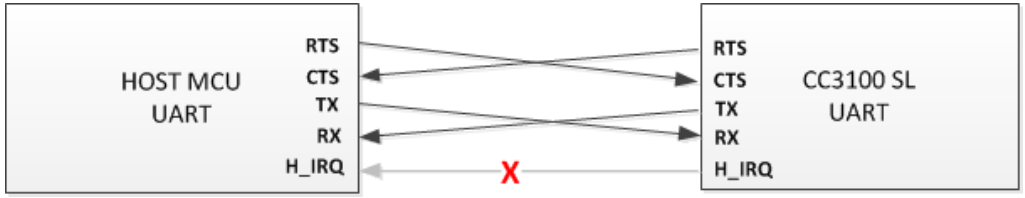


Typical 5-Wire UART Configuration

This is the typical and recommended UART topology as it gives the maximum communication reliability and flexibility between the host and the SimpleLink device.

4-Wire UART Topology

In this topology the Host IRQ line is omitted. Using this topology is allowed only if one of the following conditions is met: • Host always stays awake/active • Host goes to sleep but its UART module has receiver start-edge detection for auto wake up and does not loss data in this case



4-Wire UART Configuration

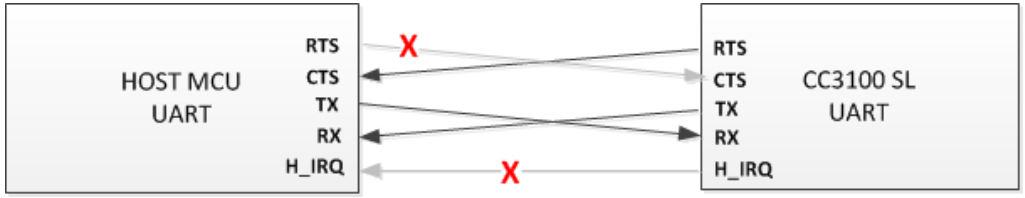
3-Wire UART Topology

In this topology, only the following lines are required:

- RX
- TX
- CTS

Using this topology is allowed only if one of the following conditions is met:

- Host always stays awake/active
- Host goes to sleep but its UART module has receiver start-edge detection for auto wake up and does not loss data in this case
- Host is always able to receive any amount of data transmitted by the SimpleLink device since there is no flow control in this direction



3-Wire UART Configuration

Since there is no full flow control, the host can't stop the SimpleLink device to send its data thus the following parameters must be carefully considered:

- Max baud rate
- RX character interrupt latency and low level driver jitter buffer
- Time consumed by user's application

UART Configuration

The SimpleLink device requires the following UART configuration:

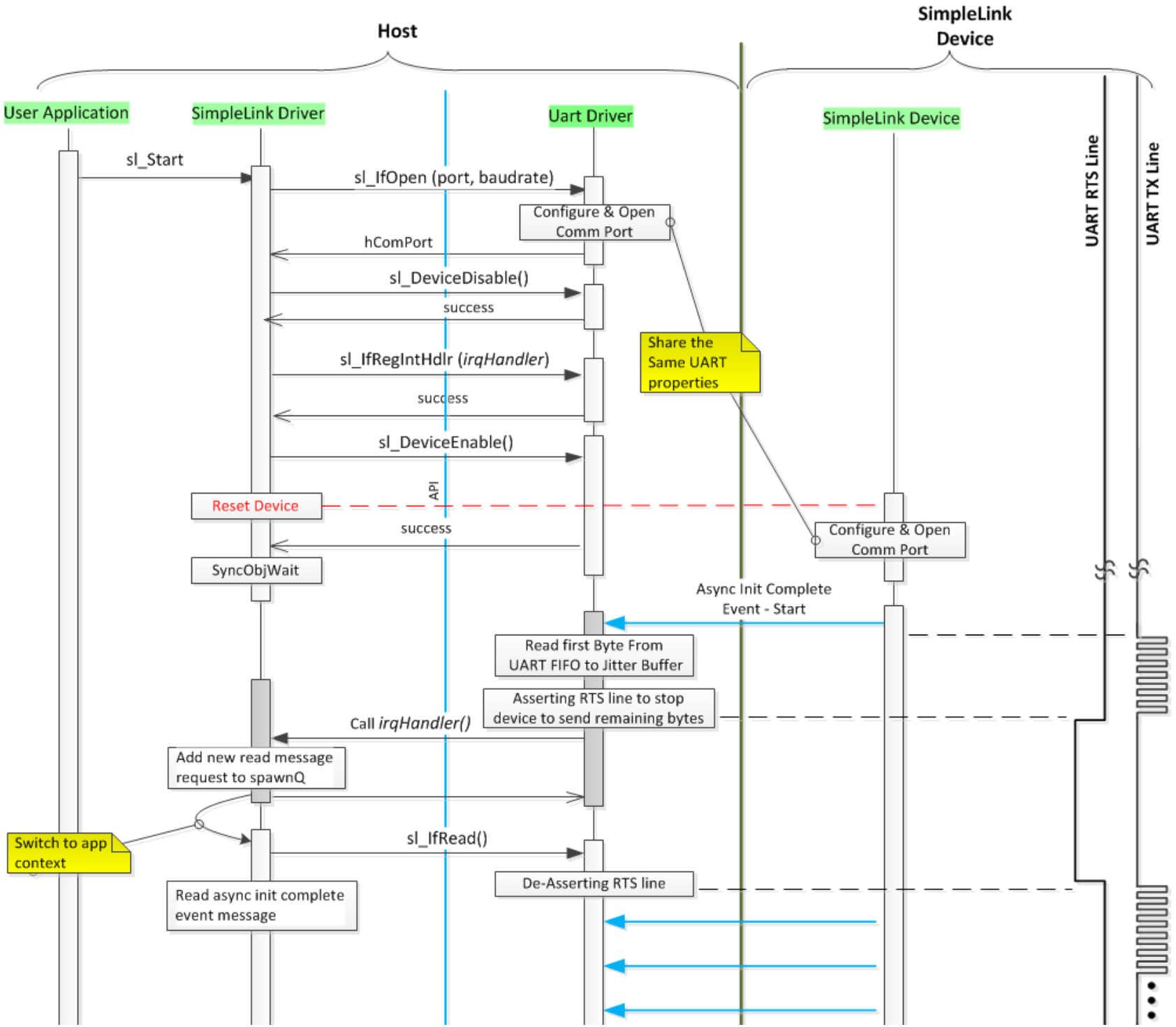
Property	Supported CC3100 Configuration
Baud rate	115200bps, No auto-baud rate detection, could be changed by the Host up to 3Mbps using special command
Data bits	8 bits
Flow Control	CTS/RTS
Parity	None
Stop bits	1
Bit order	LSBit first
Host Interrupt polarity	Active high
Host Interrupt mode	Rising edge or level '1'
Endianness	Little Endian only[1]

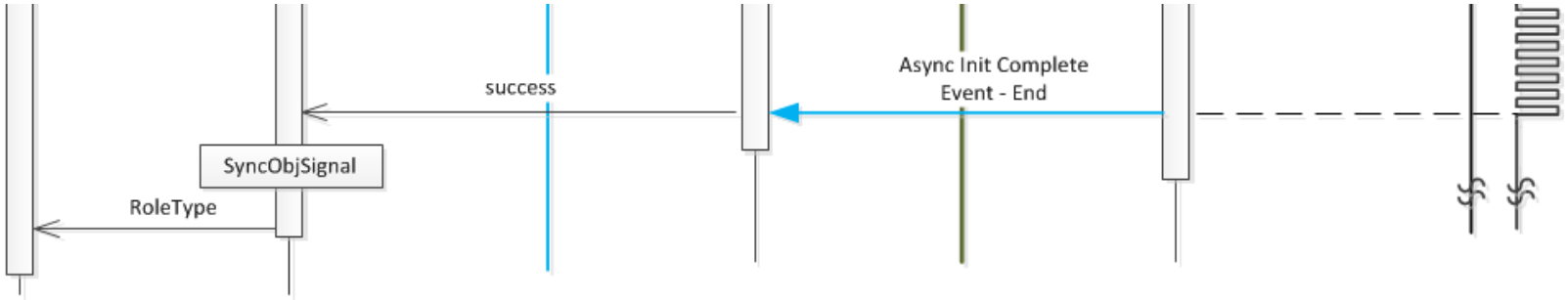
[1]The SimpleLink device does not support automatic detection of the Host length while using the UART interface.

UART Initialization

The UART module is initialized upon calling to the sl_start() API, which will call the function sl_IfOpen.

The diagram below illustrates the UART initialization sequence:


Host
SimpleLink Device
User Application
SimpleLink Driver
Uart Driver
SimpleLink Device
UART RTS Line
UART TX Line
sl_Start
sl_IfOpen (port, baudrate)
Configure & Open Comm Port
hComPort
sl_DeviceDisable()
success
Share the Same UART properties
sl_IfRegIntHdlr (irqHandler)
success
sl_DeviceEnable()
API
Reset Device
success
Configure & Open Comm Port
SyncObjWait
Async Init Complete Event - Start
Read first Byte From UART FIFO to Jitter Buffer
Asserting RTS line to stop device to send remaining bytes
Call irqHandler()
Add new read message request to spawnQ
Switch to app context
sl_IfRead()
De-Asserting RTS line
Read async init complete event message



UART initialization sequence

Calculating the host maximum working baud rate

As mentioned above, the maximum supported baud rate of the CC3100 device is 3M. Having said that, the host UART driver designer must calculate the maximum baud rate which ensures the protection of a potential overridden of its UART HW internal RX Buffer.

The maximum baud rate can be estimated using the following formula:

$$BRmax = \frac{RxFifoSize \cdot 10 \cdot 10^6}{Tlatency + RxFifoSize \cdot Tread}$$

Where:

BRmax	The maximum baud rate in bps units that could be used
RxFifoSize	The size of the HW UART FIFO buffer. In most of the low-cost and tiny controllers this size is 1
Tlatency	The time that elapses from the moment that a HW interrupt is generated to the moment that the service routine is called in worst case scenario
Tread	The time it takes to read one byte from the HW UART FIFO and store it in SW buffer. This time should take in account all SW logic involved in worst case scenario

This formula doesn't take in account all parameters and it is only for estimation purposes. The formula based on the following:

- 1 character equal to 10 bits (8 data bits, 1 start bit and 1 stop bit)
- Timing measured in micro seconds

For example, if we select the MSP430EXP5529 host running the TI UART driver example application in 25MHz clock, where:

Tlatency = 3uSec
Tread = 7uSec

Hence:

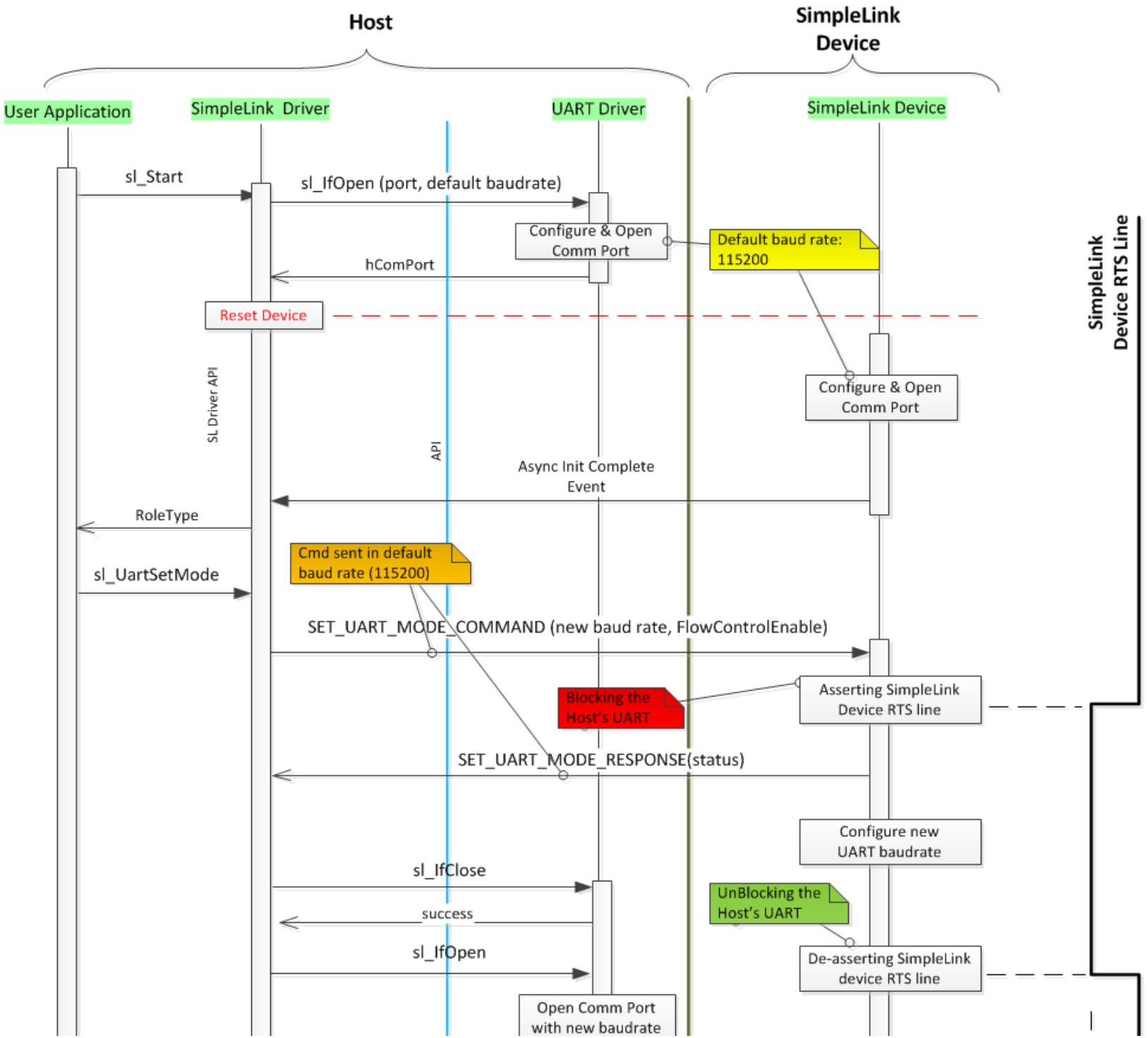
$$BRmax = \frac{1 \cdot 10^7}{3uSec + 1 \cdot 7uSec}$$

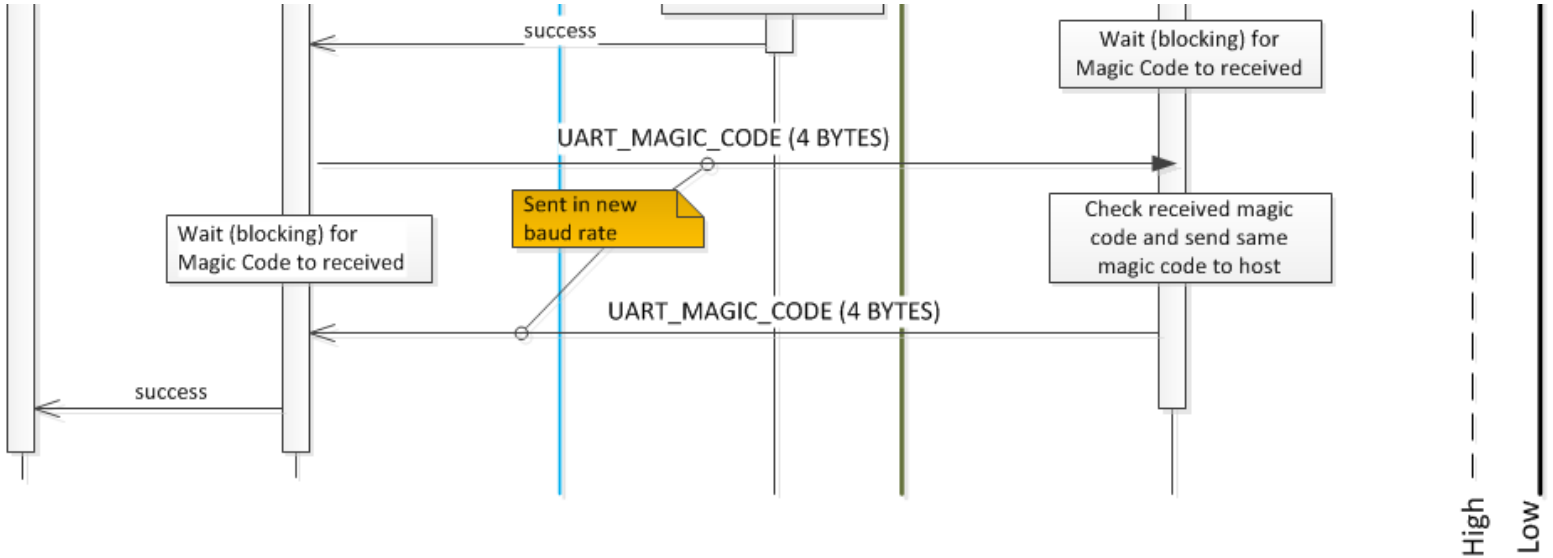
Changing the UART baud rate

Changing the baud rate can be done using the host sl_UartSetMode() command. The command takes the following 2 parameters:

uint32 BaudRate – the new desired baud rate
void* pParams – pointer to user specific parameter set

The diagram below illustrates the changing baud rate sequence:


Host
SimpleLink Device
User Application
SimpleLink Driver
UART Driver
SimpleLink Device
sl_Start
sl_IfOpen (port, default baudrate)
Configure & Open Comm Port
Default baud rate: 115200
hComPort
Reset Device
SimpleLink Device RTS Line
SL Driver API
API
Configure & Open Comm Port
Async Init Complete Event
RoleType
Cmd sent in default baud rate (115200)
sl_UartSetMode
SET_UART_MODE_COMMAND (new baud rate, FlowControlEnable)
Blocking the Host's UART
Asserting SimpleLink Device RTS line
SET_UART_MODE_RESPONSE(status)
Configure new UART baudrate
sl_IfClose
UnBlocking the Host's UART
success
sl_IfOpen
De-asserting SimpleLink device RTS line
Open Comm Port with new baudrate



UART change baud rate sequence

Implementing the UART Driver - Concept & Terminology

The user which implements the low level UART driver must consider the following components in mind:

- **Jitter buffer** - An internal buffer (minimum of 4 bytes size) which is responsible to store the bytes sent by the SimpleLink device till the host read operation starts. The host driver is informed on new data reception upon character detection on the RX UART line, and is expected to start its read operation immediately afterwards.
- **SW Flow Control Manager** – The SimpleLink device requires the UART to use HW flow control. In some low-cost controllers there is no support for HW flow control in their UART peripheral. In these cases the user must implement SW flow control that would protect the jitter buffer from being overridden by asserting the RTS line before it gets full, and follow the CC3100 device flow control state by testing the CTS line before sending any data.
- **Active Buffer** – points to the current buffer to accept the incoming bytes. At the beginning of received message points to the jitter buffer, and upon UART read operation start is switched to the host driver supplied one.

A full UART driver example code can be found in the SDK package. This example includes an implementation of SW flow control.

UART Read API implementation

This API is responsible to read bytes from an opened communication port into a buffer starting at pBuff. The read operation will be blocked till all the expected data received.

The following table lists the required parameters for such function:

Name	Type	Description
Fd	Fd_t	handle to the Uart control block, This structure could be changed by the user to include the required parameters in the target platform
pBuff	char*	pointer to the first location of a buffer that contains enough space for all expected data
Len	unsigned int	number of bytes to read from the communication port

The UART read API should implements the logic below:

1. Disabling RX interrupt and switch Active buffer to point to the supplied buffer (pBuff)
2. Copy all bytes from the jitter buffer to the Active Buffer
3. Clear the RTS line as the user provide buffer to accept the remaining data.
4. Enabling RX interrupt and waits till all bytes (length) are fully written to the supplied buffer (*pBuff*). These bytes are expected to be read from the UART RX FIFO upon UART RX interrupt service routine.
5. Once length bytes received, switch back the active buffer to point to the jitter buffer

UART Write API implementation

This API is responsible to write all required bytes to the opened communication port.

The table below lists the required parameters for such function:

Name	Type	Description
Fd	Fd_t	handle to the Uart control block, This structure could be changed by the user to include the required parameters in the target platform
pBuff	char*	pointer to the first location of a buffer that contains the data to send to the communication port
Len	unsigned int	number of bytes to write to the communication port

The UART write implementation has no special behavior. The function should send the bytes from the buffer on the UART lines. The function must make sure that the actual transmission will be executed only if the following conditions are met:

1. The host UART HW is ready to transmit data
2. The CTS line is low meaning the SimpleLink device is ready to accept data from the host

If these conditions are not met, the function should wait until they will (blocked)

Register/Unregister Interrupt Handler API implementation

This API is responsible to register/unregister an interrupt service routine that will be called upon detection of new UART message from the SimpleLink device. The service routine might be registered directly to the interrupt vector table, or can be registered internally as callback that would be called by other function on the UART driver.

The UART driver will call the handler only once, at the beginning of every UART message. This handler is expected to be masked by the host SimpleLink driver (by calling the *sl_IfMaskIntHdlr* API), before the driver starts to handle the message received. The handler will be unmasked when the SimpleLink driver finished to handles the message (by calling the *sl_IfUnMaskIntHdlr* API).

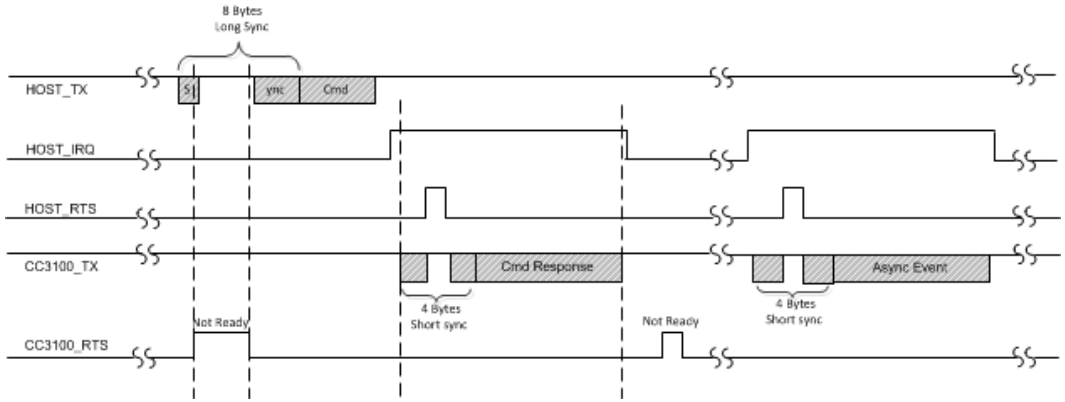
Host Interface protocol – UART perspective

As described on Message Types, the communication between the host and the CC3100 device is comprised of several types of messages:

1. Command
2. Command complete
3. Data
4. Asynchronous events

Uart host command flow

The diagram below describes the flow of a command from the host to the device along with the command complete indication from the device to the host followed by some async event. It also illustrates the behavior as appear in the sample code provided with the SimpleLink SDK for MSP430 processors.



CC3100 UART Host Command Flow

As seen in the drawing, the communication starts with the host sending the long (8 bytes) SYNC word followed by the command itself which includes header information and payload (when applicable).

Once the command has been analyzed by the device, it asserts the RTS line for the sleep exit transition duration (if was in sleep mode). The device then begins to transmit the command response which starts with the short (4 bytes) SYNC word, which being suspended by the host flow control RTS line (due to its very small 4 bytes jitter buffer). Once the host de-asserts the RTS line, the command response transmission is resumed. Later on the device may enter (if required) again to sleep mode.

The last transmission shown in the drawing above is an independent async event sent by the CC3100 device.

Synchronization Words

The communication to and from the host uses synchronization words to keep the host and CC3100 device in sync.

There are 2 types of synchronization words in use:

- Host to Device (8 bytes)
- Device to host (4 bytes).

The patterns are given in the following table:

Sync Word	Pattern (Hex)
Host to Device (Long Sync)	12 34 43 21 BB DD EE FF
Device to Host (Short Sync)	AB CD DC BA

The first 4 bytes of the Long sync are dummy bytes and have no meaning to the CC3100 device. The long sync is required to prevent data loss in case that the Host does not stop the transmission immediately on the next byte when the RTS is raised.

{{ Keystone= C2000=*For* DaVinci=*For* MSP430=*For* OMAP35x=*For* OMAPL1=*For* MAVRK=*For For technical support*

1. switchcategory:MultiCore=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **CC3100 UART Host Interface** here.

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **CC3100 UART Host Interface** here.

*technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **CC3100 UART Host Interface** here.*

*technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **CC3100 UART Host Interface** here.*

*technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **CC3100 UART Host Interface** here.*

*technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **CC3100 UART Host Interface** here.*

*technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **CC3100 UART Host Interface** here.*

*technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **CC3100 UART Host Interface** here.*

*please post your questions at http://e2e.ti.com. Please post only comments about the article **CC3100 UART Host Interface** here.*

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity