Knowledge Article

BLUETOOTH
(/S/TOPIC/0TO1M000000QHWHW...

[REFERENCE]: Using or bypassing flow control with UART communication

The BGAPI protocol and corresponding BGScript language provides two methods to ensure, or with some configurations at least to maximize, the integrity of data flowing in both directions. These

Jul 9, 2021 · Knowledge

DETAILS

The BGAPI protocol and corresponding BGScript language provides two methods to ensure, or with some configurations at least to maximize, the integrity of data flowing in both directions. These methods are **flow control** and **watermarking**. Flow control is a technique that is very commonly used with UART communication, but there are a few different approaches that may been used, it is important to clarify which is employed by our modules and how. Watermarking is less widely known, but also critical for reliable bidirectional communication when using our modules. Flow control is explained here, while watermarking is explained in a separate Knowledge Base article (http://community.silabs.com/t5/Wireless-Knowledge-Base/REFERENCE-Endpoint-watermarking-with-BGScript/ta-p/147684).

NOTE: There are some serial/UART devices which implement flow control but still exhibit small amounts of data loss when used with the BLE112 or BLE113 (especially those powered by FTDI chips). For details on this, see the section and logic capture screenshots at the end of this KB article.

Some common problems caused by the **lack of flow control**:

- Some data arrives from an external microcontroller or other host, but out of order or with some bytes missing (more pronounced at high baud rates).

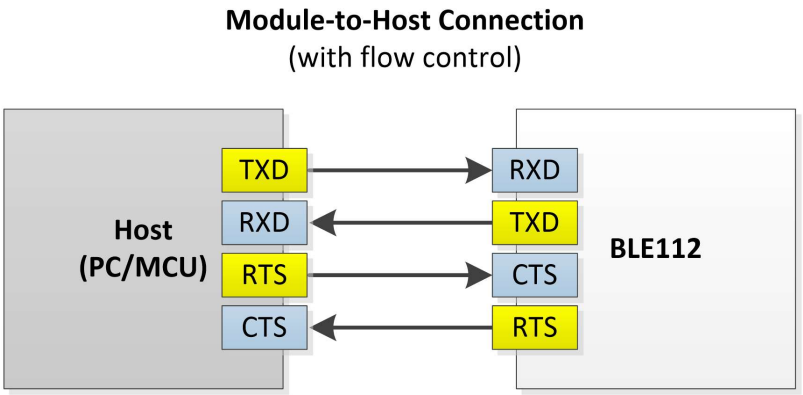
Some common problems caused by **improper use of flow control**:

- No data seems to be coming out of the module even though everything appears to be configured correctly.
- The module seems to lock up after some time working without any issues, particularly if debug info or other data is being sent out a UART endpoint but not read by another device, either because it is not connected to the port or because it is disabled or not processing data coming into its receive buffer.

Using Flow Control

Flow control is used only to indicate to the other end of the connection that it is safe to send data. For maximum reliability, we always recommend using hardware flow control for UART communication. In the case of our modules, this means that in addition to the **RXD** and **TXD** data pins, there are two more pins used to indicate when it is safe to send data from each side. These pins are known as **Request To Send (RTS)** and **Clear To Send (CTS)**. With most modern UART transcievers (including the ones used on our modules), the proper connections are to connect the local **RTS** to the remote **CTS** pin, and the local **CTS** pin to the remote **RTS** pin, much the same way that two devices' **RXD** and **TXD** pins are wired together.

The following diagram shows RTS/CTS flow control connections using our BLE112 module as an example, though the same connection scheme applies to any hardware that uses RTS/CTS flow control:



RTS is an output pin, driven **HIGH** or **LOW** with the following intended behavior:

- When **RTS** is driven **LOW**, it indicates to the remote side that it is **safe** for the remote side to send data to the local module. In other words, it indicates that the local module's receive buffer is not full and that it is not too busy with other operations.
- When **RTS** is driven **HIGH**, it indicates to the remote side that it is **not safe** for the remote side to send data to the local module, typically because either the receive buffer is full or the module is otherwise occupied and cannot service RX data interrupts.

CTS is an input pin, tested for **HIGH** or **LOW** logic with the following intended behavior (remember that the remote **RTS** pin should be connected to the local **CTS** pin):

- When **CTS** is detected as **LOW**, it indicates to the local module that it is **safe** to send data to the remote side.
- When **CTS** is detected as **HIGH**, it indicates to the local module that it is **not safe** to send data to the remote side, and that it should instead buffer (or, at worst, drop) any pending data which is wishes to send. In this situation, it is expected that any data sent to the module will not be processed.

Note that **neither CTS nor RTS are used to indicate that there is actually any data to send or receive**. They are only used to indicate whether the respective UART transceivers are in a state where they may safely **receive** any data.

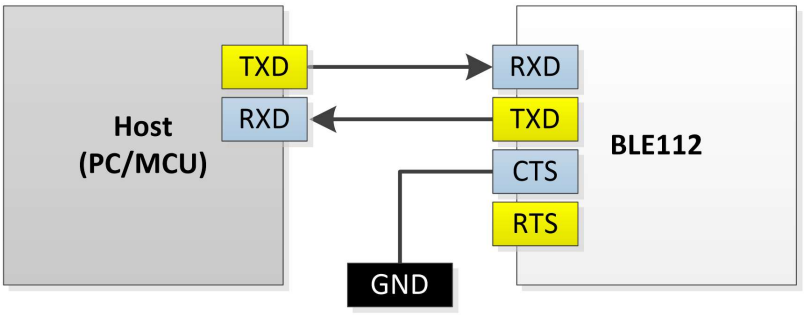
WARNING: some flow control systems, including the one used on our modules, have a limited internal sending buffer that cannot overflow without negative consequences. Specifically, if **CTS** pin is kept **HIGH** long enough so that outgoing data is buffered internally until no more space remains (64 bytes), the module may lock up entirely, requiring a reset or power cycle. Therefore, flow control should either be used properly or disabled.

Working Properly Without Flow Control

Some low-powered microcontrollers do not have hardware flow control pins on their UART transceivers, leaving only the RXD and TXD pins available to connect to our modules. In this case, flow control cannot be used and must be disabled by either of two methods.

- The easiest way is to disable it in your project's configuration files (either **project.xml** or **hardware.xml**, depending on the module being used). This requires no further connections or configuration and will cause the UART transceiver to ignore any flow control signals in either direction.
- If the project configuration cannot be changed or you do not wish to do this for some reason, you may also pull the **CTS** pin to **GND** and leave the **RTS** pin floating. Since **CTS** will always be at a logic **LOW** level, this will tell the module that it is *always* safe to send data. The following diagram illustrates this connection with the BLE112, though it applies to any module using RTS/CTS flow control:

Module-to-Host Connection
(flow control bypass*)



**Only necessary if flow control is enabled in BLE112 hardware configuration.*
CTS/RTS connections may be ignored entirely if flow control is disabled.

Note that without the use of flow control, it is not possible to guarantee data integrity. If you must do this, it can be helpful to send only short bursts of data (e.g. configuration and control data), and/or send data at a lower baud rate to give the remote side more time to process any incoming data.

You can also implement a sort of "one-way flow control" using software on the MCU side, by doing the following:

- Leave flow control enabled on the wireless module. **CTS** and **RTS** will still be used on this end.
- Pull **CTS** to **GND** on the wireless module. This will tell the module that it may always send data, preventing excessive outgoing data buffering and keeping the module from locking up.
- Connect **RTS** to any available input pin on the microcontroller.
- Before *sending* any data from the microcontroller to the wireless module, **poll the RTS-connected input pin and make sure it is LOW first**. This at least guarantees that the module will always be able to process data sent to it.

For additional information on improving reliability without flow control when using our BLE112/BLE113 module or BLED112 dongle, please read this Knowledge Base article (http://community.silabs.com/t5/Wireless-Knowledge-Base/BGLib-How-to-configure-DKBLE112-for-application-running-on/ta-p/147598) and pay attention specifically to the description of **Packet Mode**.

The Knowledge Base article on Watermarking (http://community.silabs.com/t5/Wireless-Knowledge-Base/REFERENCE-Endpoint-watermarking-with-BGScript/ta-p/147684) may also be of interest.

Data Loss Even When Using Flow Control

It has occurred many times that even when flow control is enabled on a UART peripheral on the BLE112/BLE113 module, some bytes are still lost, overwritten, or received incorrectly. The reason for this is somewhat complex and unfortunately difficult to address.

The UART standard is many decades old, and there are many interpretations and conventions concerning how it should work, especially when flow control is involved. **RTS/CTS handshaking** is one convention, as described in the sections above, and when implemented very precisely, it works quite well. However, not all serial devices out there implement it exactly the same way, even when they specify support for RTS/CTS handshaking. Therefore, the CC2540/CC2541 chip (inside the BLE112/BLE113 module) is not automatically compatible with all UART devices' flow control implementations. You should always verify the flow control capabilities of the remote end to ensure that it will work properly in your design.

As an example, many UART devices based on FDTI chips (including the Sewell Octal 8-port USB-to-RS232 converter (http://sewelldirect.com/Octal-USB-to-Serial.asp) and SparkFun FTDI adapter (https://www.sparkfun.com/products/9873)) exhibit the behavior shown in this logic capture, which is the result of a very simple "local echo" project (http://community.silabs.com/t5/Wireless-Knowledge-Base/BGScript-uart-echo-Simple-UART1-Alt1-loopback-local-echo-with/ta-p/147738) used for testing serial data flow:

Many controllers such as those in the FTDI-based devices above **flush their output buffers even when RTS line is pulled up**, and some drivers implement flow control in software inside the Windows kernel, which can negatively impact the precise timing required for reliable flow control. Based on the explanations earlier in this article, the **0×38** byte should not be sent since **RTS** is de-asserted (pulled up), but it very clearly is sent anyway.

In this situation, you might expect that the **0×38** byte will be lost, but in fact this is not what occurs. The BLE module's UART receiver is still enabled even though it has already told the remote device to stop sending data, and so what really happens is that the **0×38** byte is clocked into the RX register, **overwriting the 0×37 byte that was there before and not yet processed**. The lost byte is then the one that *should have been* sent last, instead of the one that actually *was* sent last.

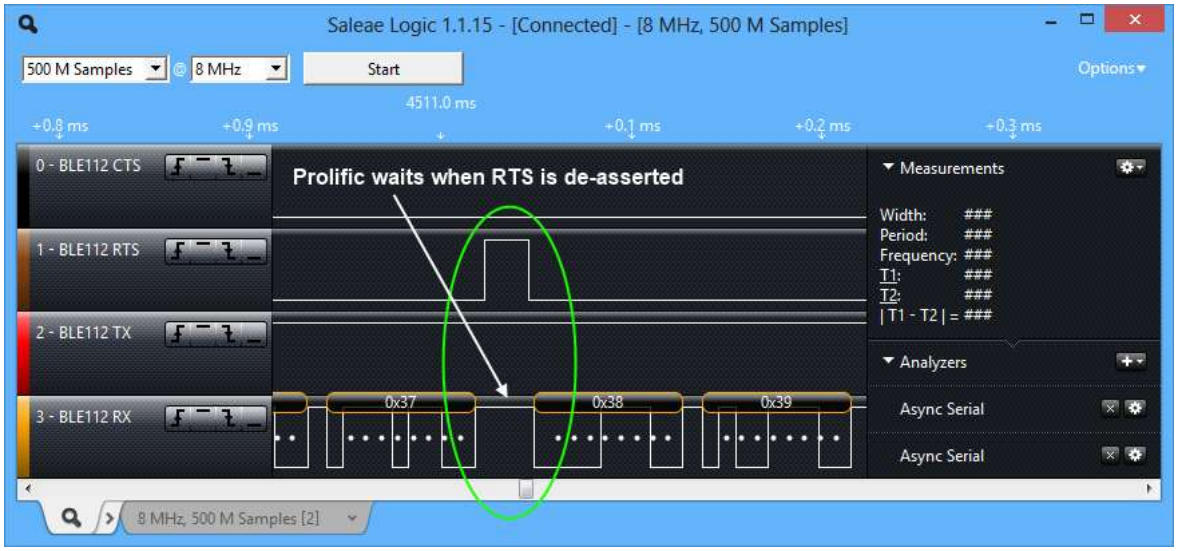
Although there are currently no 100% reliable workarounds when using the BLE112/BLE113 with a remote UART transceiver that exhibits this behavior, there are a few ways to improve things at least a little bit, which may be enough:

- **Use a slower baud rate.** This will allow the BLE module more time to process data and will make it less likely that flow control signalling will be needed, and so the non-ideal flow control behavior will happen less frequently or not at all.
- **Sent smaller chunks of data at a time.** This will have the same effect as the previous suggestion. Sending 64 or fewer bytes at a time with gaps between each chunk of data will give the module a better chance to process incoming bytes without completely filling the buffer and triggering the need for flow control. The logic capture above involved 80-byte chunks, and so the available data always exceeded the available space.
- **Configure the remote UART device to use 2 stop bits.** This is a non-intuitive solution, but it can actually help because the net effect is that there is simply a longer gap between each byte send to the BLE module. Under the right circumstances, this can make all the difference you need.
- **Use "packet" mode if your application uses BGAPI.** This offloads most of the UART RX interrupt processing task to the DMA controller inside the module, rather than requiring the main CPU to handle it. This has been known to solve BGAPI unreliability problems especially at higher baud rates. Note that you should **still use flow control** in this case if possible.

One simple guideline for BLE projects: If your application is reading UART data for the purpose of sending it out over BLE, then try to send data only as fast as you know the module can send it. BLE has pretty limited throughput, and even under the best circumstances, a 115200 UART stream will far outpace the BLE link. 38400 baud on the UART link is likely to be as fast as you can really use even with flow control enabled and perfect data integrity, simply because the BLE connection itself cannot transmit data faster than that when used in typical connection environments. If you have a 20ms connection interval and you can only send 20 bytes of payload data at a time, then try not to send more than that to the module over UART.

Working Flow Control Signals

In contrast to the above logic capture, the one below is a test run with the same BLE firmware but using a USB-to-serial adapter based on a Prolific chip. Devices such as **this one which uses the Prolific PL2303HX Rev. D chipset (http://www.amazon.com/computers-accessories/dp/B00425S1H8)** stop sending data as soon as the **RTS** line goes high:



UART transceivers based on the latest Prolific chips are known to work reliably like this.

Although this doesn't relate to UART and flow control specifically, it may be valuable for your application to note that the USB CDC virtual serial port wraps all of the data flow inside managed USB packets, and there are no data loss issues of this kind when using USB.

TITLE

[REFERENCE]: Using or bypassing flow control with UART communication

URL NAME

x-reference-using-or-bypassing-flow-control-with-uart-communication

Bluetooth (/s/topic/0TO1M000000qHWhWA...

Bluetooth Classic (/s/topic/0TO1M000000qHWmW...

Bluetooth Low Energy (/s/topic/0TO1M000000qHWrWA...

Bluetooth Mesh (/s/topic/0TO1M000000qHWwW...

Sort by:

Latest Posts

Search this feed...

Silicon Labs Admin (/s/profile/0051M00000AiYYkQAN) (Silicon Labs) published this new Knowledge.
July 9, 2021 at 5:37 PM (/s/feed/0D51M00007xeBKMSA2)

2 comments 96 views

Like Comment

More comments 1 of 2

S938088881 (/s/profile/00516000006TtKsAAK) (Xoserg70)
6 years ago

Unfortunately, have the same problem. BLE112 can not receive data reliably at a desired baud rate, without data flow. Even for small packets(not more than 36 bytes) and very long pauses between them(1 second). Very bad. Equipment which already exists have not flow control and requires baud rate of 38400. And that can not be changed. We have used many Bluetooth modules before, including BlueGiga WT11 and never faced such problem with same equipment.

Like

Log In to Comment

FOLLOW

Share This Topic

Files (0) (/s/relatedlist/ka01M000000gFYhQAM/AttachedContentDocuments)

RELATED ARTICLES

Make a comparison of UART Hardware Software and Legacy hardware flow control. (/s/article/make-a-comparison-of-uart-hardware-software-and-legacy-hardware-flow-control-x) 625

How can I implement hardware or software flow control for UART? (/s/article/how-can-i-implement-hardware-or-software-flow-control-for-uart-x) 660

BGAPI Usage without UART flow control (/s/article/bgapi-usage-without-uart-flow-control) 138

EFM32PG1/EFM32JG1/EFM32PG12/EFM32GG11 USART Hardware flow control introduction. (/s/article/efm32pg1-efm32jg1-efm32pg12-efm32gg11-usart-hardware-flow-control-introduction-x) 143

Getting started with EZSP-UART (/s/article/getting-started-with-ezsp-uart) 1.59K

TRENDING ARTICLES

Downloading CP210x drivers from Windows Update
(/s/article/downloading-cp210x-drivers-from-windows-update)

What's the role of CC pin in Type-C solution
(/s/article/what-s-the-role-of-cc-pin-in-type-c-solution)

Legacy OS Software and Driver Packages
(/s/article/legacy-os-software-and-driver-packages)

What is the serial enumeration driver and why would I need it?
(/s/article/what-is-the-serial-enumeration-driver-and-why-would-i-need-it-x)

What's the role of billboard device in Type-c solution
(/s/article/what-s-the-role-of-billboard-device-in-type-c-solution)

SILICON LABS (https://www.silabs.com/)

Stay Connected With Us
Plug into the latest on Silicon Labs products, including product releases and resources, documentation updates, PCN notifications, upcoming events, and more.

Enter Your Email

About Us (https://www.silabs.com/about-us)
Careers (https://www.silabs.com/about-us/careers)
Community (https://community.silabs.com/s/)
Contact Us (https://www.silabs.com/about-us/contact-us)
Cookies (https://www.silabs.com/about-us/legal/cookie-policy)
Corporate Responsibility (https://www.silabs.com/about-us/corporate-responsibility)
Investor Relations (http://investor.silabs.com/)
Press Room (http://news.silabs.com/)
Privacy and Terms (https://www.silabs.com/about-us/legal)
Site Feedback (mailto:feedback@silabs.com)

()

()

(https://www.linkedin.com/company/silicon-labs/)

(https://www.youtube.com/siliconlabs)

(https://www.silicon-labs.com/wechat-us/wechat)

Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit our webpage.

Copyright © 2022 Silicon Laboratories. All rights reserved.

粤ICP备15107361号 (https://beian.miit.gov.cn/)