CC256x eHCILL Low Power Protocol

Contents

Introduction

The HCILL is a TI proprietary protocol, providing the baseband controller and the Bluetooth host with a deterministic way to independently go into respective sleep modes. In order to maintain the synchronization between the host and the Bluetooth (BT) device, it is important for each side to know when the other is going into sleep mode.

The enhanced HCILL (eHCILL) protocol includes several additions to the original HCILL specification that provide easier implementation on the host side and, at the same time, continuing to offer backward compatibility so that modifications are not required for a host that has implemented the original HCILL.

Terms and Abbreviations

The table lists many of the terms and abbreviations used in this document.

Terms and Abbreviations

Abbreviation /Term	Meaning / Explanation
Baseband Controller	TI Bluetooth single-chip solution
FW	Firmware
GPIO	General-purpose input / output
Host	The host processor that controls the baseband controller
HCI	Host controller interface
HCILL	Host controller interface, low level
eHCILL	Enhanced host controller interface, low level
HW	Hardware
SW	Software

Protocol Description

HCILL Deep Sleep Commands List

The table below lists HCILL commands related to deep sleep mode:

List of HCILL Commands

HCILL Command	Opcode
HCILL_GO_TO_SLEEP_IND	0x30
HCILL_GO_TO_SLEEP_ACK	0x31
HCILL_WAKE_UP_IND	0x32
HCILL_WAKE_UP_ACK	0x33

HCILL Deep Sleep Commands Description

HCILL_GO_TO_SLEEP_IND This command is sent by the baseband controller only (the host does not need to send it) to indicate that it is requesting to go into the Sleep state.

HCILL_GO_TO_SLEEP_ACK This command is sent by the host in response to a HCILL_GO_TO_SLEEP_IND command to indicate that both the host and baseband controller can go to a low-power mode.

HCILL_WAKE_UP_IND This command is sent to wake up the sleeping party (host or controller).

HCILL_WAKE_UP_ACK This message is a response to HCILL_WAKE_UP_IND to acknowledge a state of being awake and ready to communicate.

Enhanced HCILL Rules for Host

Situation: HCILL_GO_TO_SLEEP_IND received.
Host should: Pull RTS high, enable wakeup from CTS, and send HCILL_GO_TO_SLEEP_ACK.

Situation: Wakeup from CTS.
Host should: Disable CTS interrupt and release RTS.

Situation: HCILL_WAKE_UP_IND received.
Host should: Send HCILL_WAKE_UP_ACK

Situation: HCILL_WAKE_UP_IND sent and HCILL_WAKE_UP_IND received.
Host should: Go directly to AWAKE state without sending an ACK.

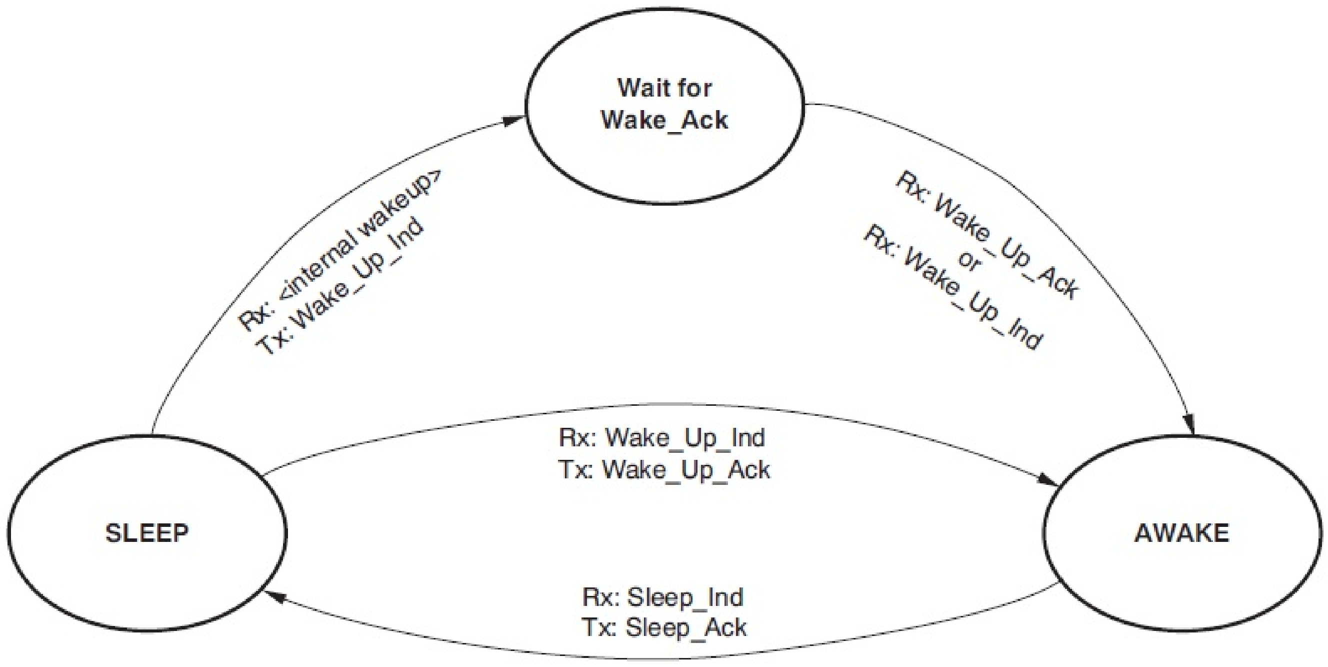
Situation: Host is in sleep state and has data or a command to send.
Host should: Disable wakeup from CTS, send HCILL_WAKE_UP_IND, and lower RTS.

```
NOTE:If an HCI command is sent immediately before an HCILL_SLEEP_IND message is received, the host must complete the sleep preparation procedure (pull RTS high, enable wakeup from CTS,
and send HCILL_SLEEP_ACK). The baseband controller then goes through the wakeup procedure to return the HCI_Command_Complete_Event.
```

Host Deep Sleep State Machine

The figure below shows the basic host state-machine diagram in the host Bluetooth deep-sleep implementation. As part of the eHCILL, the host is not required to send SLEEP_IND (see the Inactivity TimeOut feature in Section 2.7 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Configuring_eHCILL), Configuring eHCILL) but can if desired. Also, the host is programmed to wake up from asserted CTS and assert RTS on going to sleep to prevent the baseband controller from sending data.

As below figure shows, the state-machine states are logical states; that is, a device can be in sleep state but not in actual low-power mode.



Deep-Sleep Basic State-Machine

eHCILL Implementation Guidelines

The following summary is a general description of the sleep procedure, the wakeup procedure, and the CTS interrupt service routine (ISR) from a host perspective.

Wakeup ISR on CTS

- Inform the power manager system (if applicable) that Bluetooth is awake (for example, turn on the UART clock).
- Release RTS.

Sleep Procedure

- HCILL_SLEEP_IND is received from baseband controller (see the Inactivity TimeOut feature in Section 2.7 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Configuring_eHCILL), Configuring eHCILL).
- Enable the Wakeup CTS ISR.
- Pull RTS high.
- Reply with HCILL_SLEEP_ACK→change HCILL state to Sleep.
- Inform power manager system (if applicable) that Bluetooth is asleep (for example, the UART clock can be shut off).

Wakeup by Baseband Controller

- Wakeup ISR is executed because of CTS interrupt (see the RTS pulse width feature in Section 2.7 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Configuring_eHCILL),Configuring eHCILL).
- Disable the Wakeup CTS ISR.
- Release RTS.
- HCILL_WAKEUP_IND is received.
- Reply with HCILL_WAKEUP_ACK→change HCILL state to Awake (see the WAKEUP_IND retransmission feature in Section 2.7 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Configuring_eHCILL), Configuring eHCILL).

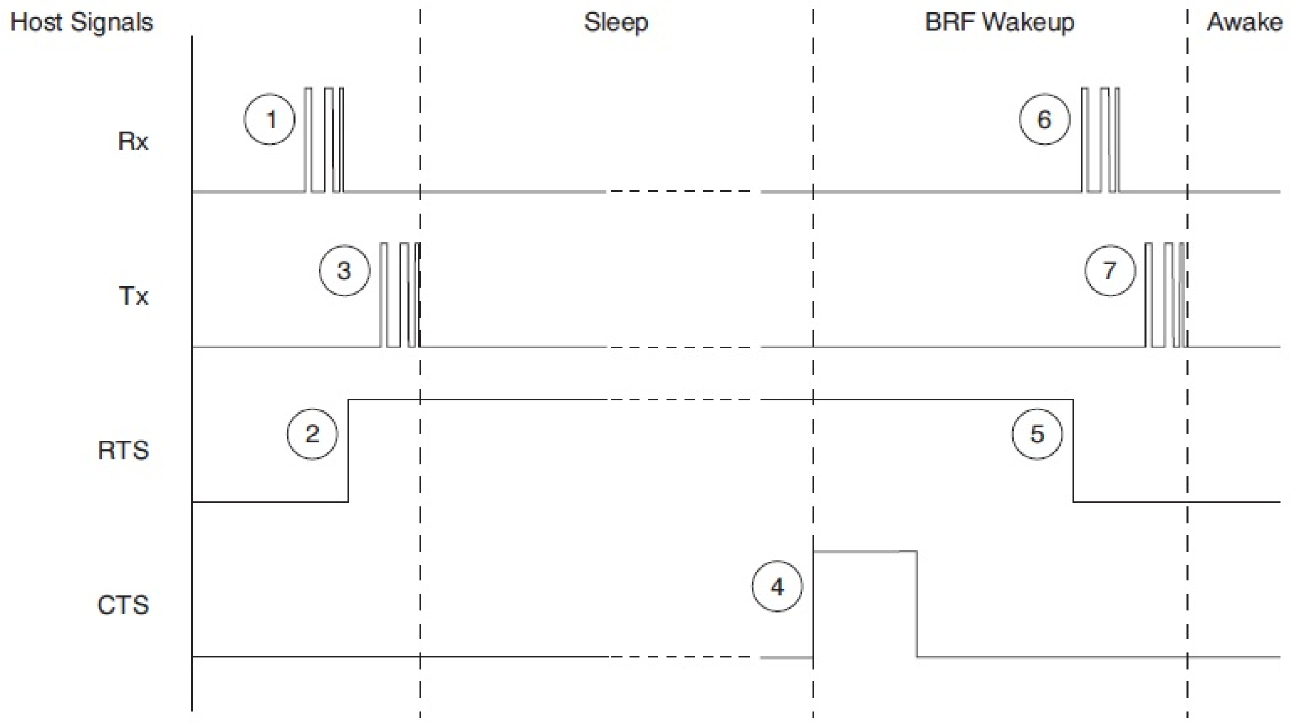
Wakeup by Host

- Disable Wakeup ISR.
- Turn UART on (if turned off).
- Release RTS.
- Send HCILL_WAKEUP_IND.
- Wait for HCILL_WAKEUP_ACK (or HCILL_WAKEUP_IND in case of a collision type 1; see Section 2.6.3 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Wakeup_Collision_1), Wakeup Collision 1, and Section 2.6.4 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Wakeup_Collision_2), Wakeup Collision 2) →change HCILL state to Awake.

Sleep and Wakeup Diagrams

Wakeup by Baseband Controller

The figure below shows the process of the baseband controller waking up the device.



Wake-Up by Baseband Controller

1) HCILL_SLEEP_IND received from baseband controller.

2) Host pulls RTS high and enables wake-up from CTS. Pulling RTS is so the baseband controller can send the HCILL_WAKEUP_IND to the host only when the host is awake and ready for it.

3) Host sends HCILL_SLEEP_ACK.

4) Baseband controller wakes up host by asserting CTS for 150µs (recommended pulse width).

5) Host wakes up and lowers its RTS so the baseband controller can send the HCILL_WAKEUP_IND.

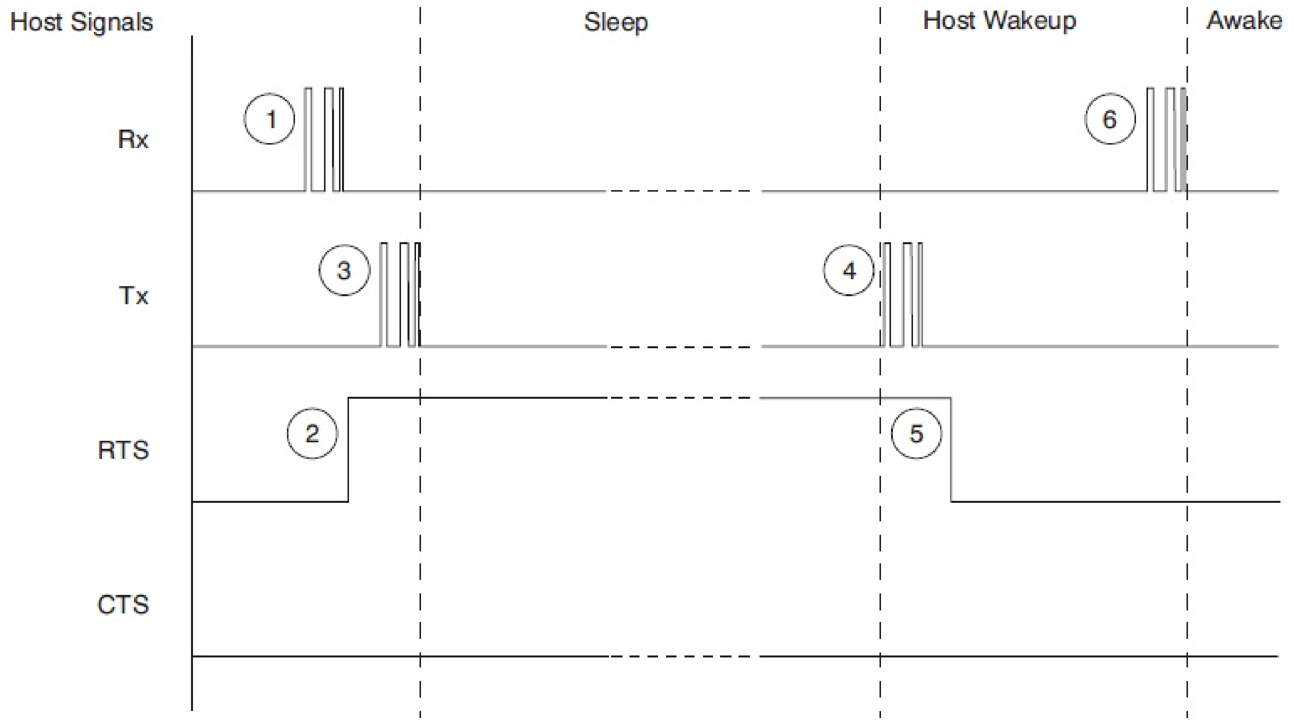
6) HCILL_WAKEUP_IND received from baseband controller once RTS is low.

7) Host sends HCILL_WAKEUP_ACK.

```
NOTE: If the host cannot control the RTS level, the RTS pin functionality might need to be changed to GPIO and asserted/deasserted as needed. When the device is awake,
the pin functionality might need to be changed back to RTS.
```

Wakeup by Host

The figure below shows the procedure for the host to wake up the device.



Wake-Up by Host

1) HCILL_SLEEP_IND is received from baseband controller.

2) Host pulls RTS high and enables wake-up from CTS so that the baseband controller can send HCILL_WAKEUP_IND to the host only when the host is awake and ready.

3) Host sends HCILL_SLEEP_ACK.

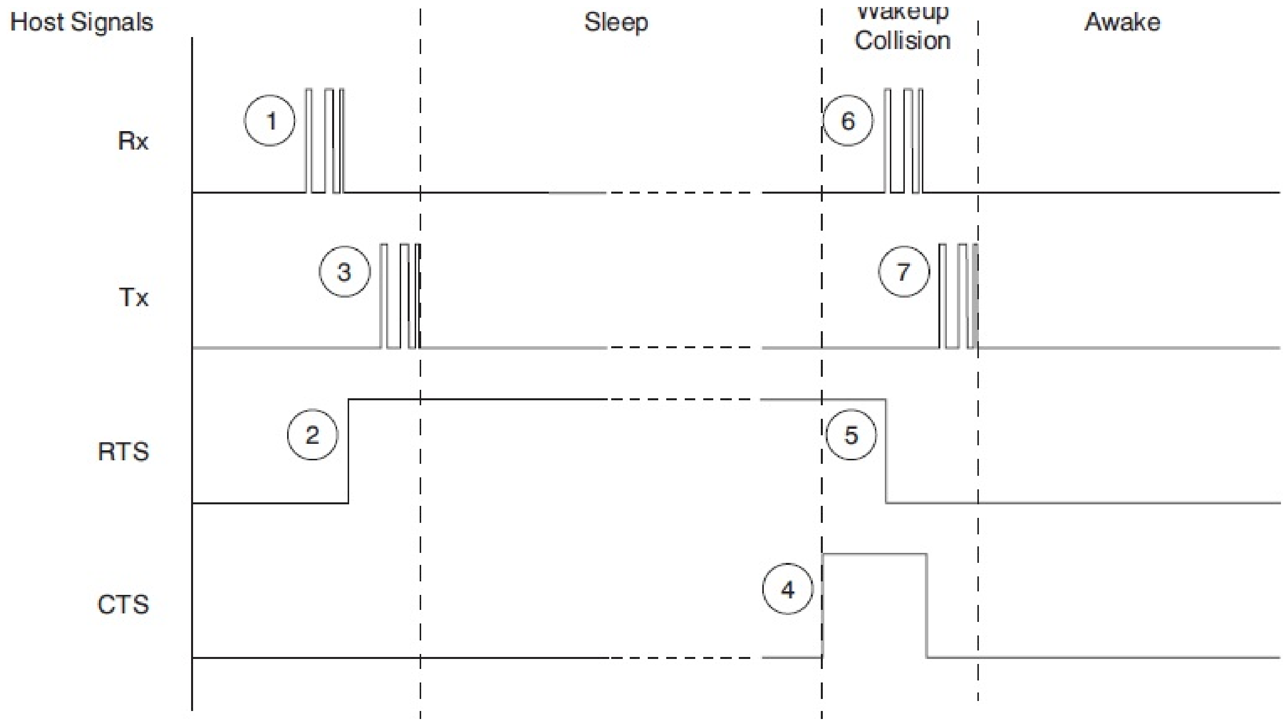
4) Host wakes up the baseband controller by sending a HCILL_WAKEUP_IND message.

5) Host lowers RTS so that the baseband controller can reply with HCILL_WAKEUP_ACK.

6) Host receives HCILL_WAKEUP_ACK once the baseband controller physically wakes up.

Wakeup Collision 1

The figure below shows the process of a wakeup collision type 1.



Wake-Up Collision Type 1

1) HCILL_SLEEP_IND is received from baseband controller.

2) Host pulls RTS high and enables wake-up from CTS so that the baseband controller can send HCILL_WAKEUP_IND to the host only when the host is awake and ready.

3) Host sends HCILL_SLEEP_ACK.

4) Baseband controller wakes up host by asserting CTS for 150 µs.

5) Host lowers RTS so that the baseband controller can reply with HCILL_WAKEUP_ACK (it is not aware this is a collision).

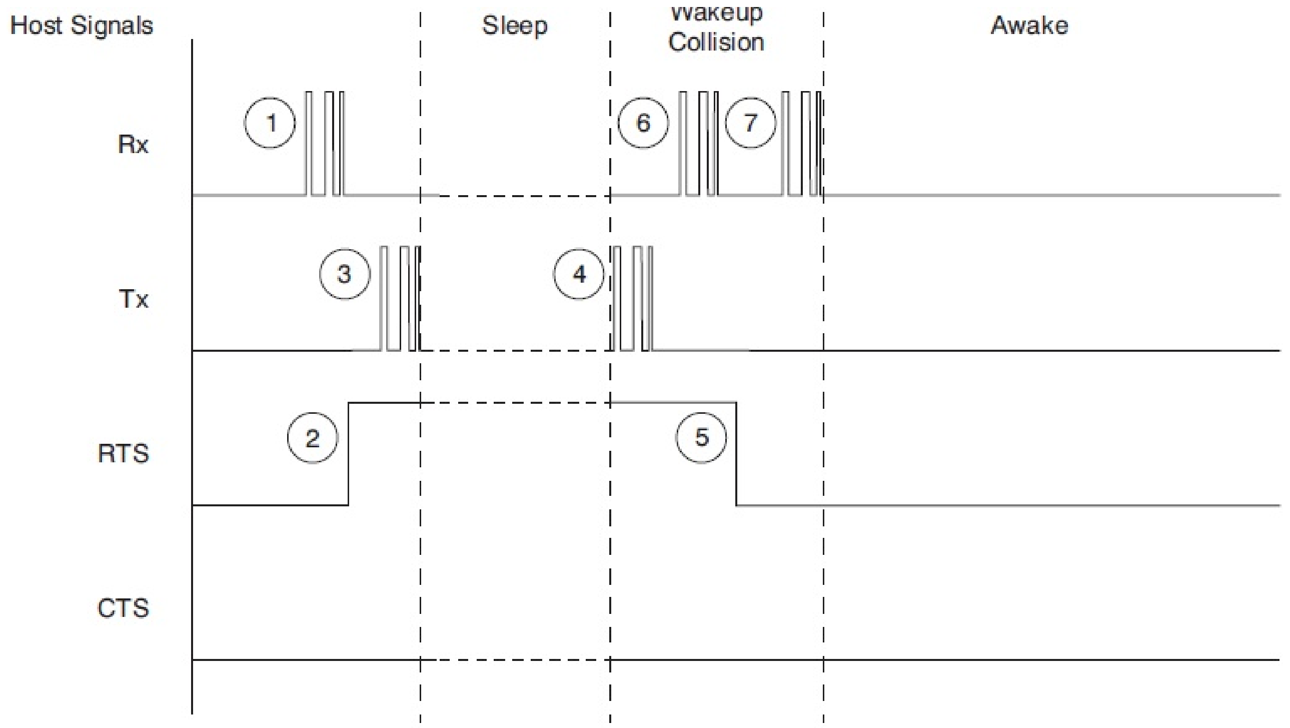
6) HCILL_WAKEUP_IND is received from the baseband controller (once RTS is low).

7) Host wakes up the baseband controller by sending a HCILL_WAKEUP_IND (delayed until CTS is low).

```
NOTE: If the host sends an HCILL_WAKEUP_IND, the host should consider reception of an HCILL_WAKEUP_IND as an HCILL_WAKEUP_ACK and go directly to the Wakeup state.
Sending an HCILL_WAKEUP_ACK to the baseband controller in this case does not harm its state-machine.
```

Wakeup Collision 2

The figure below shows the process of a wakeup collision type 2.



Wake-Up Collision Type 2

1) HCILL_SLEEP_IND is received from baseband controller.

2) Host pulls RTS high and enables wake-up from CTS so the baseband controller can send the HCILL_WAKEUP_IND to the host only when the host is awake and ready.

3) Host sends HCILL_SLEEP_ACK.

4) Host wakes up the baseband controller by sending a HCILL_WAKEUP_IND message.

5) Host lowers RTS so that the baseband controller can reply with HCILL_WAKEUP_ACK.

6) HCILL_SLEEP_IND is received from the baseband controller. This command has been in the queue and must be ignored by the host (sent until RTS goes low; it is not aware of a collision).

7) Host must wait to receive HCILL_WAKEUP_ACK (acknowledge to Step 4) to wake up.

Configuring eHCILL

As part of the eHCILL, the following three features must be configured: Inactivity timeout, WAKEUP_IND retransmission, and RTS pulse width.

- Inactivity timeout is a timer implemented in the baseband controller that determines when to send the HCILL_SLEEP_IND. If there is no activity on the UART line, the baseband controller is assumed to allow the host to go to sleep. This time is separate from the actual power mode the baseband controller can enter because, in some cases, the host can enter a low-power mode even though the baseband controller cannot (an active voice call, for example).

- WAKEUP_IND retransmission specifies the interval time to retransmit the WAKEUP_IND message until acknowledged (until WAKEUP_ACK is received successfully at baseband controller) to increase protocol robustness. This feature can also be used on hosts that do not support a wakeup interrupt on the CTS line (see Section 3.1 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Host_Cannot_Wakeup_from_the_CTS_Signal), Host Cannot Wakeup from the CTS Signal). In this case, the UART RX is routed to an internal interruptible GPIO. The first WAKEUP_IND message is used for the wakeup interrupt and the second WAKEUP_IND is accepted as a WAKEUP byte on the host UART RX.

- RTS pulse width sets the minimum pulse width of the baseband RTS that is followed by a WAKEUP_IND message. The minimum pulse width should be set to 150 μs in case high baud rates require waking up some HLOS platforms.

The following command sets up eHCILL(for more information, see the HCI Vendor-Specific Commands documentSection 2.2.21 (http://processors.wiki.ti.com/index.php/CC256x_VS_HCI_Commands#HCI_VS_HCILL_Parameters_.28oxFD2B.29)):

HCI_VS_HCILL_Parameters(Inactivity_Timeout, WakeUp_Ind_Retransmission_TimeOut,RTS_pulse_width)

where:

- Inactivity_TimeOut is in Bluetooth frame units (1.25 ms); that is, an Inactivity_TimeOut value of 80 sets a timeout of 100 ms (80 × 1.25 ms). Default value is 100 ms.
- WakeUp_Ind_Retransmission_TimeOut is the timeout, in Bluetooth frame units (1.25 ms), for resending WAKEUP_IND if no acknowledgement is received. A value of 0 means no retransmission. Default value is 500 ms (400 × 1.25 ms).
- RTS_pulse_width sets the minimum pulse width of RTS (μs). A 0 value disables the pulse. Default value is 1 μs; however, 150 μs should be used.

Enabling Deep Sleep

Deep-sleep operation is disabled by default at power up. To enable the baseband controller deep-sleep feature and activate the eHCILL protocol, the host must send the HCI_VS_Sleep_Mode_Configurations command first(for more information, see the HCI Vendor-SpecificCommands document Section 2.2.22 (http://processors.wiki.ti.com/index.php/CC256x_VS_HCI_Commands#HCI_VS_Sleep_Mode_Configurations_.28oxFD0C.29)).

- HCI_VS_Sleep_Mode_Configurations 0xFD0C, 1, Deep_Sleep_Enable, Deep_Sleep_Mode, 0xFF, 0xFF, 0xFF, 0xFF, 100

– Deep_Sleep_Enable = 0 (disable), 1 (enable).

  - Default = 0 (disable).

– Deep_Sleep_Mode = 0 (HCILL),… ,0xFF (Don't change).

  - Default = 0 (HCILL).

For the current deep-sleep status, send HCI_VS_Get_System_Status 0xFE1F, 0xFE1F(for more information, see the HCI Vendor-SpecificCommands documentSection 2.2.23 (http://processors.wiki.ti.com/index.php/CC256x_VS_HCI_Commands#HCI_VS_Get_System_Status_.28oxFE1F.29)).

Alternative Implementations

Previous sections describe the recommended way to implement the eHCILL mechanism on the host. This section provides more implementation options that can be used when it is difficult to use the host RTS and CTS for purposes other than flow control.

Host Cannot Wakeup from the CTS Signal

There are three possibilities in this case, from a host perspective:

1) Normally, each UART line can be multiplexed with a GPIO line. Thus, instead of using the CTS input pin as CTS, first change its functionality to be GPIO (interruptible). Then, apply the wake-up ISR previously described for CTS to this new GPIO. In some cases, the hardware flow control might need to be disabled first.

2) At hardware design time, connect the host CTS signal to a separate GPIO and apply the wake up ISR described previously for CTS to this new GPIO.

3) If the CTS cannot be controlled, the UART RX can be routed to an internal interruptible GPIO. In this case, the first WAKEUP_IND message is used for the wakeup interrupt and the second WAKEUP_IND is accepted as a WAKEUP byte on the host UART RX. WAKEUP_IND retransmission value in HCI_VS_HCILL_Parameters command must be configured accordingly (see Section 2.7 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Configuring_eHCILL), Configuring eHCILL).

Host Cannot Control the RTS Line

There are two possibilities in this case:

1) Normally, each UART line can be multiplexed with a GPIO line. Thus, instead of controlling the RTS output pin as RTS, first change its functionality to be GPIO. Asserting or de-asserting a GPIO should not be a problem. In some cases, the hardware flow control might need to be disabled first.

2) The purpose of asserting the host RTS line during sleep mode is so the host can wake up from a HCILL_WAKEUP_IND byte only if awake and ready. If the RTS is not asserted (during sleep mode) and baseband controller must wake up, it sends the HCILL_WAKEUP_IND immediately after sending a CTS wakeup pulse whether the host is ready or not. In this case, the host must respond with a HCILL_WAKEUP_ACK as a result of the CTS wake-up indication regardless whether the HCILL_WAKEUP_IND is received or not. Recall that the baseband controller may keep sending HCILL_WAKEUP_IND messages until acknowledged(see Section 2.7 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Configuring_eHCILL), Configuring eHCILL).

Baseband Controller Hardware Behavior

The baseband controller hardware behaves according to these parameters:

- The baseband controller wakes up on a falling edge of the RX_HCI signal.
- The first byte received while in deep sleep (HCI_WAKE_UP_IND) wakes up the device but is discarded because reception is usually corrupted (first few bits are lost).
- To allow the host to send the HCI_WAKE_UP_IND byte, RTS is always low while in deep-sleep mode(enabling data reception).

Performing HCI_Reset

The power management protocol relies on both the baseband controller and the host being aware of the counterpart state. Breaking this synchronization can cause the baseband controller to seem inaccessible to the host. During normal operation, power management sequencing performs smoothly if the rules discussed here are followed.

However, to ensure that power management operation continues to run after an HCI_Reset, the host must implement the following steps as the reset procedure:

Step 1. Disable Deep-Sleep mode (see Section 2.8 (http://processors.wiki.ti.com/index.php/CC256x_eHCILL_Low_Power_Protocol#Enabling_Deep_Sleep) Enabling Deep Sleep).
Step 2. Wait for Command Complete event.
Step 3. Send HCI_Reset.
Step 4. Wait for Command Complete event (and wait for CTS indication that baseband controller is active again).

{{ 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 **CC256x eHCILL Low Power Protocol** here.	Keystone= ▪ 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 **CC256x eHCILL Low Power Protocol** here.	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *here.*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *here.*	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *here.*	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *here.*	*For technical support please post your questions at http://e2e.ti.com. Please post only comments about the article* ***CC256x eHCILL Low Power Protocol*** *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

Retrieved from "https://processors.wiki.ti.com/index.php?title=CC256x_eHCILL_Low_Power_Protocol&oldid=165952"

This page was last edited on 10 December 2013, at 02:30.