Application Note

DA14580 External Wake-Up Mechanisms

AN-B-026

Abstract

This document describes the external wake-up mechanisms that can be used in external processor based applications of the DA14580 Bluetooth® Smart SoC. Two wake-up mechanisms can be implemented: external processor to DA14580 and DA14580 to external processor. These can be used in either UART or SPI interface. An external processor can wake up the DA14580 using a GPIO signal as the interrupt line. This can be used as an alternative to the existing DA14580 auto wake-up process. Similarly, when the DA1580 wakes up due to internal kernel timer expiration or due to a pre-programmed BLE event, it can toggle a GPIO to wake up the external processor.

Contents

Figures

1 Terms and definitions

BLE	Bluetooth© Low Energy
CS	Chip Select
CTS	Clear to Send
GPIO	General Purpose Input Output
GTL	Generic Transport Layer
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
RTS	Request to Send
SDK	Software Development Kit
SoC	System-on-Chip
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter

2 References

1. UM-B-004, DA14580 Peripheral drivers, User manual, Dialog Semiconductor
2. UM-B-015, DA14580 Software architecture, User manual, Dialog Semiconductor
3. UM-B-013, DA14580 External processor interface over SPI, User manual, Dialog Semiconductor
4. UM-B-006, DA14580 Sleep mode configuration, User manual, Dialog Semiconductor
5. DA14580 Datasheet, Dialog Semiconductor
6. UM-B-010, DA14580 Proximity application, User manual, Dialog Semiconductor

3 Introduction

In systems that use the DA14580 Bluetooth® Smart SoC together with an external processor, one could use special wake-up mechanisms to further extend the already ultra-low power features of the DA14580.

Two new features will be described here:

- The DA14580 sleeps forever and wakes up via an external processor signal (see Section 8 for more information).
- The external processor is in sleep mode and wakes up via the DA14580.

For the external processor to DA14580 wake-up feature, the DA14580 Wakeup Timer, as described in [5] and [1] can be used. This particular DA14580 internal hardware block can wake up the system after a pre-programmed number of GPIO events. Any of the GPIO inputs can be selected and configured to generate a wake-up interrupt. A simple toggle of the pre-configured GPIO will wake up the system.

In this particular wake-up architecture, the DA14580 sleeps forever (see Section 8 for more information) and only wakes up when the external host wants it to. This external host processor can wake up the system by toggling a DA14580 GPIO assigned to a DA14580 wake-up interrupt and send the required message.

The following DA14580 signals can be used as wake-up signals:

- Any GPIO
- The DA14580 UART CTS signal, when the UART is being used as the transport layer.
- The DA14580 SPI_EN (CS) when SPI is being used as the transport layer.

When the last two signals are used the following applies: a DA14580 GPIO cannot have multiple functions at the same time. For example, it cannot act as a UART CTS signal and as a wake up interrupt simultaneously. Therefore, the software has to configure the GPIO functionality accordingly. Before going to sleep, the software has to configure the GPIO as a wake up interrupt. When the DA14580 wakes up, it should reconfigure it as UART CTS.

Sections 4, 5 and 8 provide a more detailed analysis of the external processor to DA14580 wake-up procedure. Three examples are provided: using any GPIO, using the UART CTS signal and using the SPI_EN signal.

Similarly, the DA14580 can wake up an external processor by toggling a GPIO signal. The DA14580 system could wake itself up to service BLE events, respond to internal kernel timer expirations and perform other tasks. When the DA14580 system wakes up, the UART or the SPI interface is automatically activated to enable the communication with the external processor and possibly send a message to it. If the external processor is in sleep mode, the DA14580 can toggle a GPIO to wake up the external processor.

Section 7 provides a more detailed analysis of the DA14580 to external processor wake-up mechanism.

4 Waking up the DA14580 using any GPIO

Any DA14580 GPIO can be used as wake-up interrupt for the DA14580. This generic implementation implies that the user is aware of the system hardware and that the GPIO used will not conflict with any other external hardware.

When the user decides to use any GPIO in either the UART or the SPI transport interface, it should be noted that the flow control of the transport layer should definitely be used. The flow control will provide a safe mechanism for communication synchronisation. Only by using the flow control the host system will be sure that the DA14580 has woken up and is ready to receive messages.

Figure 2 illustrates the generic GPIO wake-up process with flow control and Figure 1 shows the required connections.

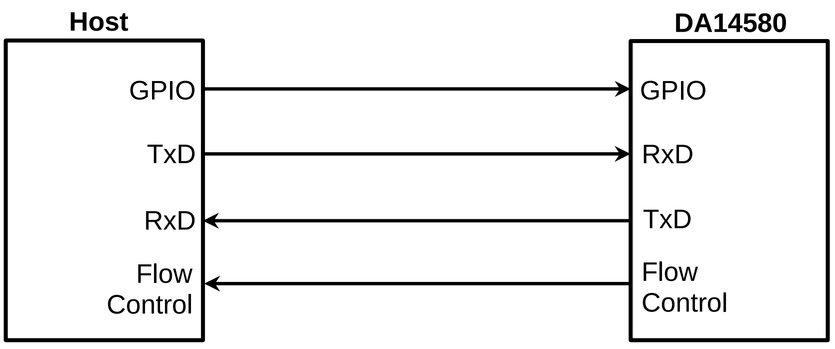

Figure 1: External CPU to DA14580 generic wake-up connections

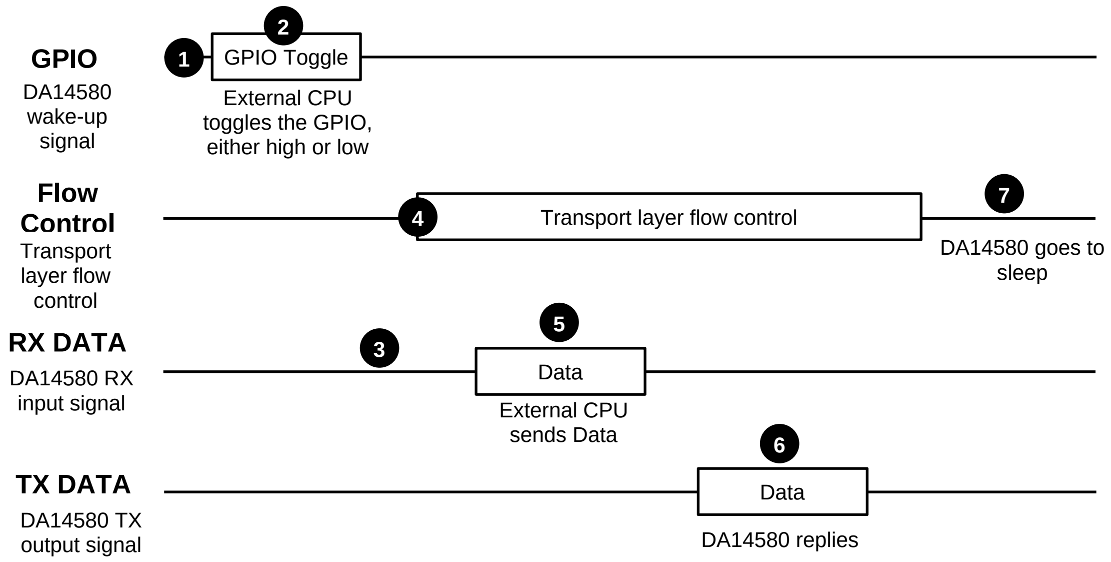

Figure 2: External CPU to DA14580 generic wake-up process

Details:

1. The DA14580 is in sleep mode. The GPIO input signal has simple GPIO functionality and is configured to have a wake-up interrupt, active either LOW or HIGH.
2. The external processor wants to send a message to the DA14580. It toggles the GPIO.
3. The external processor cannot send any data yet. The DA14580 is still in the wake-up process and therefore the external processor should wait for the flow control signal.
4. The DA14580 has woken up and asserts or de-asserts (depending on the transport layer used) the flow control signal. It is ready to receive messages.
5. The external processor detects the flow control signal and sends the data.
6. The DA14580 replies.
7. Actions on the DA14580 are performed, according to the received message and when finished it goes to sleep by appropriate flow control signalling.

5 Waking up the DA14580 using the UART CTS signal

In this external wake-up configuration using the UART CTS as a wake-up signal, a special requirement exists: the external processor should be able to modify its RTS signal functionality. That means that it should be able to toggle its RTS signal before sending anything via the UART interface. This may not be feasible in some processors, as the UART driver might be a fixed hardware block with dedicated signals that their functionality cannot be changed. In that case the process described in Section 4 can be used.

Figure 3 shows the connections needed and Figure 4 illustrate the external wake-up process. A more detailed description follows.

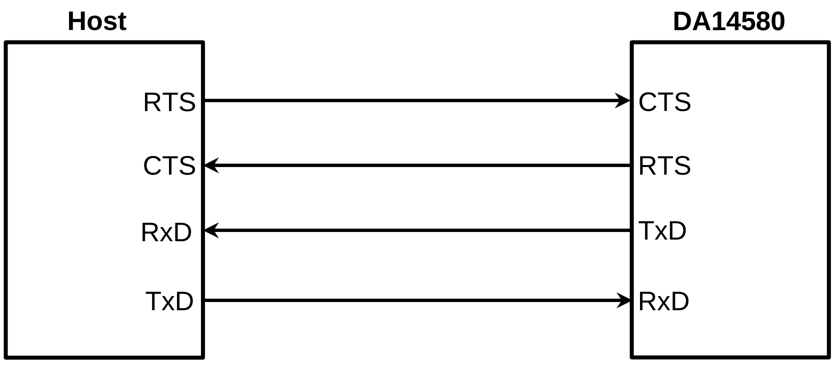

Figure 3: External CPU to DA14580 wake-up UART connections

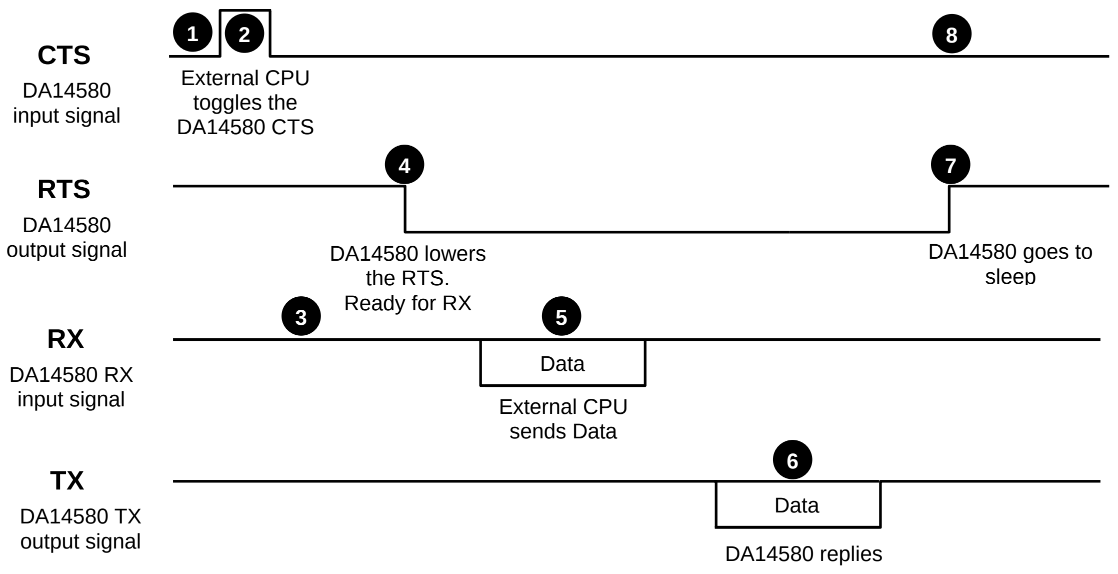

Figure 4: External CPU to DA14580 wake-up process via UART

Details:

1. The DA14580 is in sleep mode. The CTS input signal has simple GPIO functionality and is configured to have an active HIGH wake-up interrupt.
2. The external processor wants to send a message to the DA14580. It should take control of the CTS signal from its UART driver block and toggle it HIGH.
3. The external processor cannot send any data yet because the RTS signal is still HIGH. The DA14580 is still in the wake-up process.
4. The DA14580 has woken-up and sets the RTS signal LOW. It is ready to receive messages.
5. The external processor detects the LOW level of the RTS signal and sends the data.
6. The DA14580 replies.
7. Actions on the DA14580 are performed, according to the received message and when finished it goes to sleep, making RTS HIGH.
8. At the same time and just before entering sleep, the DA14580 changes the CTS signal from UART CTS functionality to simple GPIO functionality with external wake-up interrupt.

6 Waking up the DA14580 using the SPI_EN signal

The SPI external wake-up process uses a similar procedure to the UART described above. The same requirement exists, which implies that the external CPU should be able to control the SPI_EN (CS) SPI signal apart from any dedicated SPI driver it may have.

Figure 5 and Figure 6 are described in [3]. They illustrate an external wake-up process with the SPI using the SPI_EN as the wake-up signal.

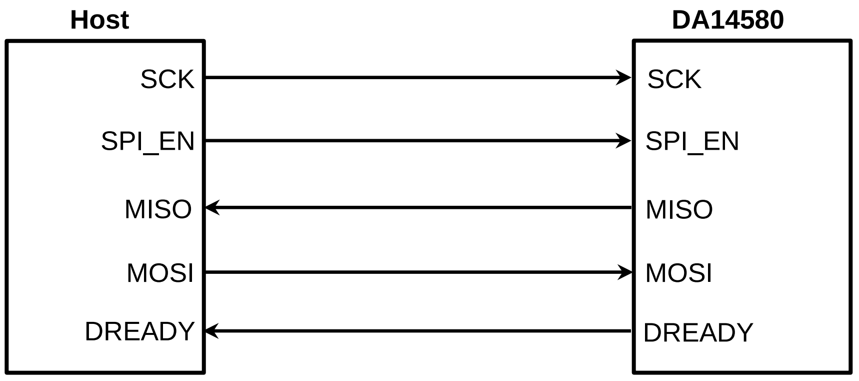

Figure 5: External CPU to DA14580 wake-up SPI connections

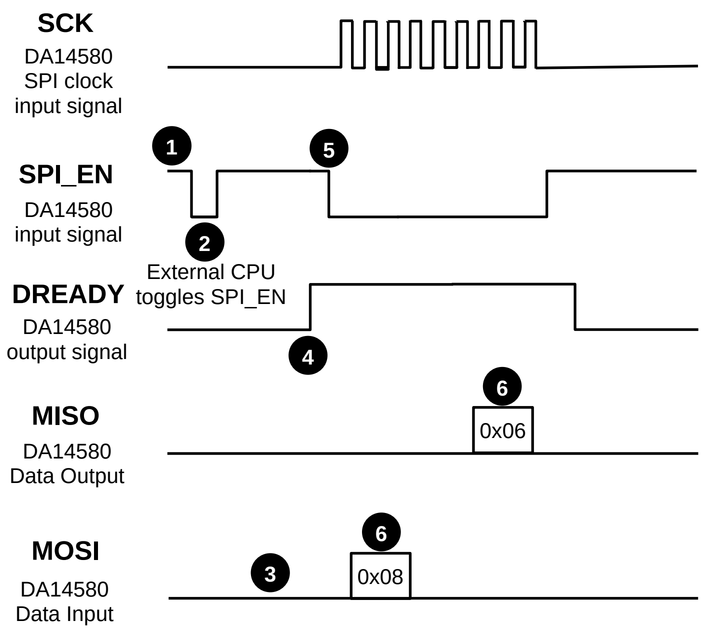

Figure 6: External CPU to DA14580 wake-up process via SPI

Details:

1. The DA14580 is in sleep mode. The SPI_EN input signal has simple GPIO functionality and is configured to have an active LOW wake-up interrupt.
2. The external processor wants to send a message to the DA14580. It should take control of the SPI_EN signal from its SPI driver block and toggle it LOW.
3. The external processor cannot send any data yet, since the SPI custom flow control is enabled as described in [3].
4. The DA14580 has woken up and sets the DREADY signal HIGH as described in [3].
5. The external processor detects the HIGH level on the DREADY signal and sets the SPI_EN LOW.

6. The MOSI and MISO bytes and the rest of the flow control process and SPI data exchange are described in [3].

7 Waking up an external processor

Figure 7 shows a block diagram of the connections needed for the DA14580 to wake up an external processor. Figure 8, illustrates the process. It shows how the DA14580 can wake up an external processor before sending a message over the transport layer interface. A simple DA14580 GPIO signal is used and is been toggled according to the external processor requirements. Flow control should be used in either SPI or UART interface in order to ensure communication synchronisation. The steps outlined next give a more detailed description of the process.

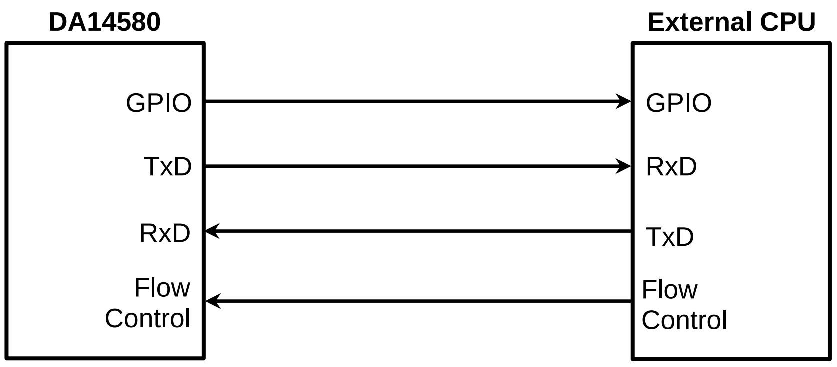

Figure 7: DA14580 to external processor wake-up connections

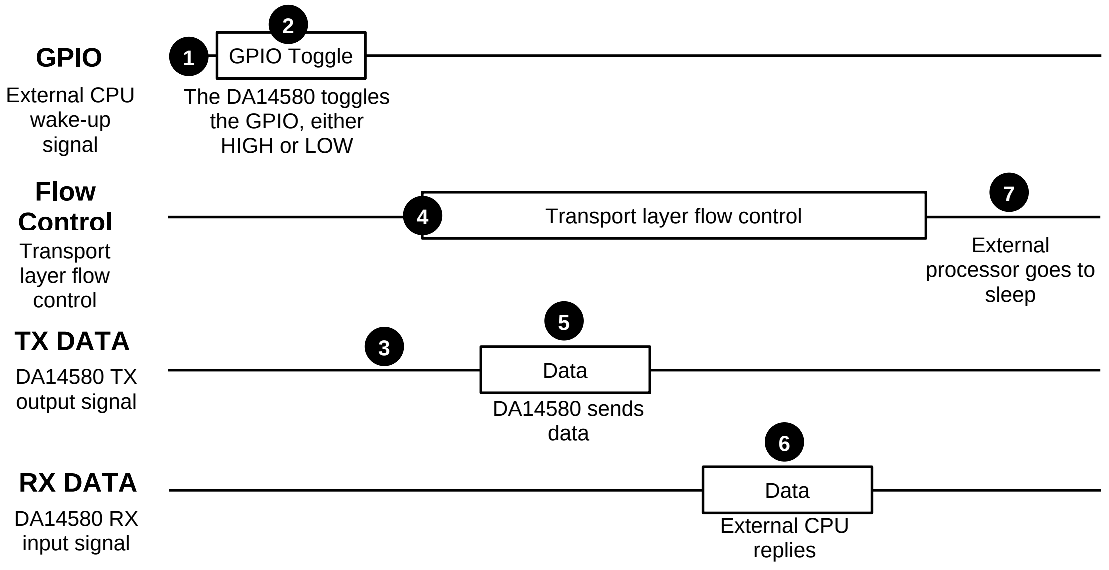
GPIO
External CPU wake-up signal
Flow Control
Transport layer flow control
TX DATA
DA14580 TX output signal
RX DATA
DA14580 RX input signal
1
2
GPIO Toggle
The DA14580 toggles the GPIO, either HIGH or LOW
4
Transport layer flow control
7
External processor goes to sleep
3
5
Data
DA14580 sends data
6
Data
External CPU replies

Figure 8: DA14580 to external processor wake-up process

Details:

1. The external processor is in sleep mode. The DA14580 GPIO signal, configured to act as the external processor wake-up signal, is set to output. It is initialised either to LOW or HIGH depending on the external processor wake-up signal requirements.
2. The DA14580 wants to send a message to the external processor. The DA14580 should toggle the GPIO to wake up the external processor.
3. The DA14580 cannot yet send any data. The external processor is still in the wake-up process and therefore the DA14580 should wait for the flow control signalling.

4. The external processor has woken up and it asserts or de-asserts (depending on the interface used) the flow control signal. It is ready to receive messages.
5. The DA14580 detects the level change of the flow control signal and sends the data.
6. The external processor replies.
7. Actions on the external processor are performed, according to the received message and when finished it goes to sleep by appropriate flow control signalling.

8 Sleep limitation

A sleep limitation currently exists in the software stack, which prevents the DA14580 to sleep forever in the external processor configuration and when the GTL interface is used. This limitation does not exist in the integrated processor configuration software.

The DA14580 must automatically wake itself up at least every 10 s, even when no external wake-up interrupt is triggered. This automatic wake up does not prevent the external wake-up process to be fully functional as described.

A workaround for this limitation will be given in a future release of this application note, as alternatives are still under investigation.

Appendix A Software changes

The software changes needed to implement the external wake-up processes will be outlined next.

A.1 External processor to DA14580 wake-up process software

To enable the external processor to DA14580 wake-up process one should include the following definition:

```
/*External wake up mechanism*/
#define EXTERNAL_WAKEUP     1
```

This definition can be found in the following two files:

```
dk_apps\keil_projects\proximity\monitor_fe\da14580_config.h
dk_apps\keil_projects\proximity\reporter_fe_spi\da14580_config.h
```

The external wake-up GPIO assignment can be passed as a function parameter when the function `ext_wakeup_enable` is called.

This is done inside the following file:

```
dk_apps\src\plf\refip\src\arch\main\ble\arch_main.c
```

```
#if ((EXTERNAL_WAKEUP) && (!BLE_APP_PRESENT)) // external wake up, only in full embedded designs
#ifdef CFG_HCI_SPI
      // Enable external wakeup interrupt at SPI_EN, active low.
      ext_wakeup_enable(SPI_GPIO_PORT, SPI_CS_PIN, 0);
#else
      // Enable external wakeup interrupt at UART CTS, active high.
      ext_wakeup_enable(UART1_CTSN_GPIO_PORT, UART1_CTSN_GPIO_PIN, 1);
#endif
#endif
```

The above gives default GPIO wake-up configurations for either UART CTS or SPI_EN signals, depending on the communication transport layer used. One could modify the parameters of the `ext_wakeup_enable` function as desired and set any GPIO port and pin as the wake-up signal, irrespectively of the interface in use.

A.2 DA14580 to external processor wake-up process software

An example implementation exists in the throughput evaluation project. One could edit the following definitions to change the GPIO used as the wake-up signal.

```
#define EXT_WAKEUP_PORT     GPIO_PORT_0
#define EXT_WAKEUP_PIN      GPIO_PIN_7
```

These definitions can be found in the following file:

```
sdk_ref\dk_apps\src\modules\app\src\app_project\throughput_eval_peripheral_fh\system\p
eriph_setup.h
```

The activation of the wake-up process is enabled by defining the `CFG_EXT_PROCESSOR_WKUP` in the `da14580_config.h` configuration file.

Appendix B DA14580 wake-up timing

The wake-up delay of the DA14580 is 4.5 ms to 5 ms. This delay is the same, irrespectively of the transport layer or the wake-up signal being used.

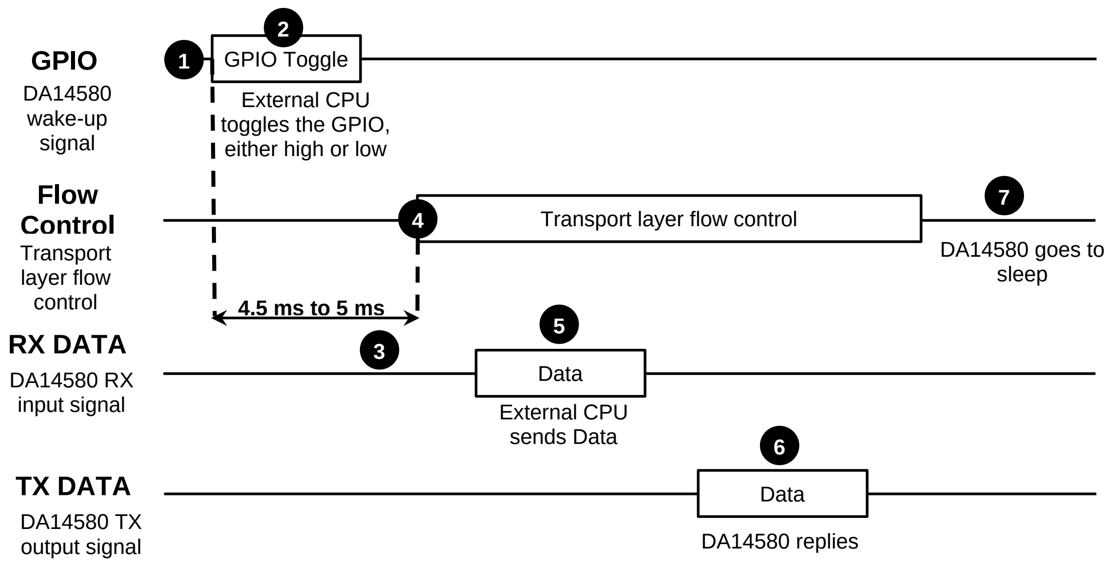

Figure 9: DA14580 wake up timing

Figure 9 shows the wake-up timing. From the start of the wake-up signal toggle (GPIO, UART CTS or SPI_EN signal) until the DA14580 is ready to receive messages, indicated by the flow control process, 4.5 ms to 5 ms are needed.

9 Revision history

Revision	Date	Description
1.1	23-Dec-2021	Updated logo, disclaimer, copyright.
1.0	15-Aug-2014	Initial version.

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance