User manual

DA14580 External processor interface over SPI

UM-B-013

Abstract

This document describes the configuration of the external processor interface over SPI for the DA14580. The SPI is used as transport layer to interface the external processor to the DA14580. A 5-wire protocol is employed and a test bed with two DA14580 boards is described for testing and evaluation purposes.

Contents

Figures

Tables

1 Terms and definitions

ACK	Acknowledgment
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
CS	Chip Select
DK	Development Kit
DREADY	Data Ready
GPIO	General Purpose Input/Output
LED	Light-Emitting Diode
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
ROM	Read-Only Memory
SCK	Serial Peripheral Interface Clock
SDK	Software Development Kit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transceiver

2 References

1. UM-B-010, DA14580 Proximity application, User manual, Dialog Semiconductor
2. DA14580 Datasheet, Dialog Semiconductor
3. RW-BLE-HOST-IS, RW BLE Host Interface Specification, Riviera Waves
4. UM-B-014, DA14580 Development Kit, User manual, Dialog Semiconductor

3 Introduction

This document describes the configuration of the external processor interface over SPI for the DA14580 (see Ref. [2]). In this configuration the host application runs on the external processor, the BLE stack is implemented in the DA14580 and the SPI is used as transport layer. The SPI in the external processor is configured as a master and in the DA14580 as a slave device. The 5-wire transport protocol supports the exchange of commands, events and indication messages. The host application also serves as an external SPI master boot device, which downloads the application image into the DA14580's SRAM.

In this document the Proximity Reporter application as described in Ref. [1] is used as an example. One DA14580 is used as SPI slave device with Bluetooth stack and radio active. Another DA14580 is used as SPI master, where only the ARM M0 microcontroller is active. Figure 1 shows the configuration used.

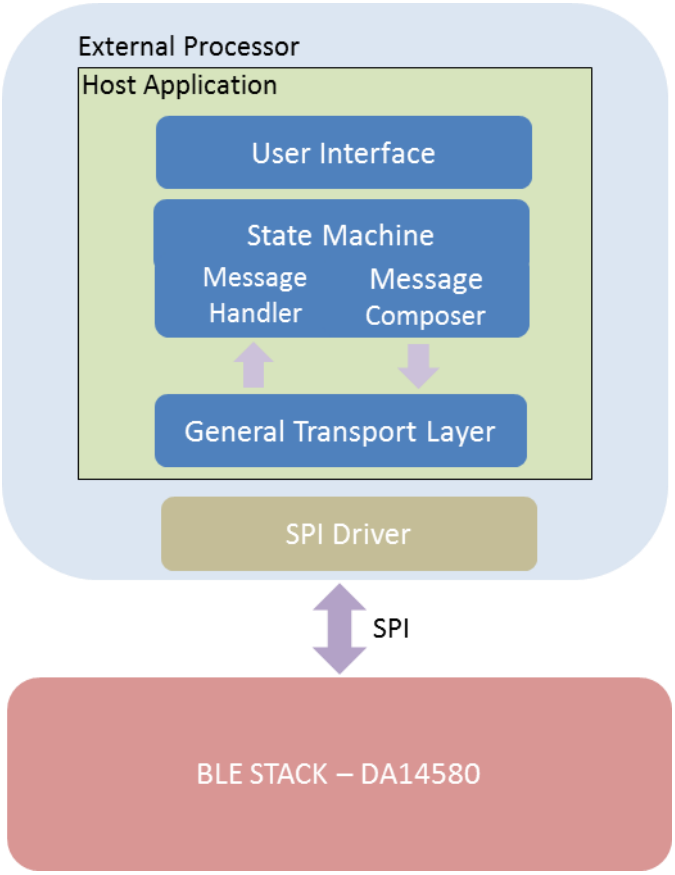

Figure 1: External processor configuration over SPI

4 Software description

The DA14580 Software Development Kit (SDK) includes examples of two Keil projects that implement the host application and the BLE software stack of the external processor configuration via SPI.

The project for the host application running on the external processor is stored in the folder:
host_apps\da14580\proximity\reporter\host_proxr.uvproj

The project for the BLE software stack over SPI is stored in the folder:
dk_apps\keil_projects\proximity\reporter_fe_spi\fe_proxr_spi.uvproj

The host application can also be used to download the image to the DA14580 over SPI. This image could for instance be stored in the non-volatile memory of the host device.

This document explains the options to build and run both software projects on two DA14580 devices, or to build and run the host application project only and use it to download the included pre-built image to the slave device. Section 7.2.1 describes in detail how to select each option before running the host application project.

5 External CPU interface message format

Before going into the details of the 5-wire protocol used to implement the external CPU interface over SPI, one has to be familiar with the basics of the message format of the external CPU interface commands, events and indications used by the two devices to communicate.

Note: The interface described here is proprietary. Please refer to [3] for further details.

Every valid external CPU interface message consists of a start byte (0x05), an 8-byte header that holds the message ID, the source ID, the destination ID, the parameter length and the data payload, which holds the message parameters. The external CPU interface protocol message format is depicted in Figure 2.

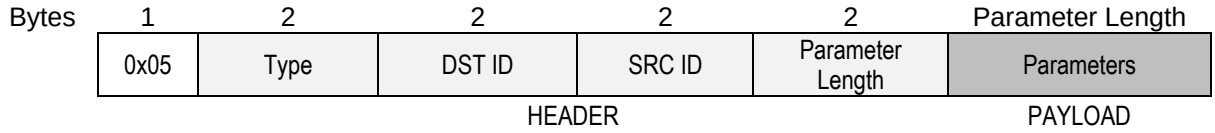

Figure 2: Packet format

For any given application based on the external processor concept, two devices are needed: one CPU that will run the application layer software and a device that will run the Bluetooth stack up to the profile. As explained in the introduction this concept will be demonstrated using two DA14580 devices, where one is only used as an SPI master microcontroller. The sample application project for the SPI master can be ported to any other microcontroller with SPI. Since the SPI master provides the SPI slave with the clock signal and initiates all SPI transfers, the master-to-slave communication is straightforward.

SPI transactions are synchronous; that means that if the SPI slave device is not “listening” while the master is transmitting, the message might be lost or partially received. The SPI driver of the slave device makes sure that all the data sent from the SPI master device are received in-time and properly by the SPI slave device. This driver enables SPI FIFO mode (see [2]) which however can only store 5 bytes at most. So, it serves as a small safety net but cannot be used to store a whole message (just a message’s header is 8 bytes long). The implementation of the external CPU interface over SPI ensures that the aforementioned requirements are met.

However, for the slave device to transmit data, the master device has to initiate it by reading an amount of bytes equal to the incoming message length. Upon reception of the message initiator (0x05), the master has to get to message reception state and to read eight more bytes, which constitute the message’s header. As soon as the two bytes that state the parameter length (or payload size in bytes) are received, the master must read as many bytes as the parameter length. Upon completion it should return to an idle state.

The protocol described in the following sections eliminates issues that arise from the synchronous nature of the SPI. For further information on the DA14580’s SPI module, please refer to [2].

6 The 5-wire protocol

6.1 SPI configuration

The slave DA14580 device running the BLE software stack configures its SPI controller with the following parameters:

- 8-bit mode
- Slave role
- Mode 0: SPI clock is initially expected to be low and SPI phase is zero.

The master device has to be configured as an SPI master with the same parameters. The SPI master must provide the clock (SCK) and chip select (CS) inputs for the DA14580 configured as a slave.

The SPI master must provide a clock with a frequency of maximum 512 kHz. This is due to the driver implementation on the SPI slave: in order to avoid having the processor busy during the entire reception of a message over SPI, the 5-byte SPI FIFO has been used on the SPI driver of the slave device to allow the processor to be used for other tasks between consecutive byte receptions over SPI.

Since external CPU interface is a bidirectional interface, the slave DA14580 device must have the ability to initiate an SPI transfer and transmit data to the SPI master. Thus, an additional, flow control, signal has to be used from the slave DA14580 to the master device to indicate when data is ready to be transmitted.. This signal, DREADY, introduces a 5th wire in addition to the standard SPI. The SPI configuration for each end is depicted in Table 1.

Table 1: Pin assignment for host and BLE application

Pin	SPI MASTER (Host Application)	SPI signals	SPI SLAVE (BLE DA14580)
P0_0	SPI_CLK as output	SCK	SPI_CLK as input
P0_1	SPI_EN as output	/CS	SPI_EN as input
P0_2	SPI_DI as input	MISO	SPI_DO as output
P0_3	SPI_DO as output	MOSI	SPI_DI as input
P0_7	GPIO as input	DREADY	GPIO as output

When implementing this example on two DA14580 development kits, the user has to remove the jumpers from pin header J26, which connect the UART CTS/RTS signals and would otherwise conflict with the MISO and MOSI functions.

6.2 Flow control

In the external processor configuration both ends can initiate a message transmission at the same time. In the case of the external processor interface over UART, dedicated hardware flow control signals are used, CTS/RTS, which make sure that each side will not start a transmission once another is in progress in the opposite direction. In the case of SPI, such hardware signals do not exist and data can be transmitted in both directions at the same time. Thus, a software flow control mechanism is needed to make sure that only one side can transmit a message at any given time.

A basic flow control functionality is achieved using the DREADY signal, but since the slave device may enter sleep mode, additional flow control mechanisms are needed to notify the master device external that the slave device can receive a message or not (i.e. is in sleep mode or transmitting a message) or to prevent a slave's attempt to start a message transmission while another message is in progress. The flow control mechanisms that ensure a reliable implementation of the external processor configuration over SPI are described in the next sections.

6.2.1 Additional signal DREADY

The external processor configuration over SPI allows both ends to initiate a message transmission at the same time. Basic flow control functionality is achieved using the DREADY signal. The slave device activates the DREADY line to notify the master device that it needs to transmit a message. The master device on the other side should check the status of the DREADY signal to enable reception of the slave device's messages, or to prevent attempts to send messages to slave while a slave to master transmission is in progress.

However, since the slave device may enter sleep mode, an additional flow control mechanism is needed to inform the external host application that the slave device can receive a message or not (i.e. is in sleep mode or transmitting a message). This flow control mechanism is based on flow on and flow off bytes.

6.2.2 Flow on/flow off bytes

The slave DA14580 device transmits a flow on byte (0x06) when it exits sleep mode to denote an active period (it sends one also after system initialization). When it enters sleep mode, it sends a flow off byte (0x07) to denote an inactive period. The master device cannot transmit any data outside slave's active period and must not transmit any flow on/flow off bytes (it should always be active). On start-up, the master device must wait to receive a flow on byte before transmitting any message. The slave device also transmits a flow off byte prior to sending a message. After the message transmission has been completed, it sends a flow on byte to indicate that it is available to receive messages from the master.

6.2.3 DREADY acknowledgement

To prevent the event of simultaneous transmissions from both master and slave (i.e. the master sends the 0x05 start byte to begin the transmission of a message and the slave at the same time sends a flow off byte to subsequently transmit another message), the master has to acknowledge the DREADY request by the slave. To do so, when the master detects DREADY to be active, it sends an acknowledgement byte (ACK, chosen to be 0x08) to inform the slave that it can go on to send the flow off byte (or any other data). This functionality is shown in Figure 3. On any DREADY rising edge, the master has to acknowledge that it detected it and enable the slave device to send data. If the master does not acknowledge the DREADY rising edge, the slave waits until the Chip Select line is inactive (which will indicate that the master has finished sending the message), activate DREADY again, and wait for the acknowledgement to proceed to transmit.

6.2.4 Chip Select polling

For the master to slave message transmission, the SPI master device must keep the Chip Select signal active (low) throughout the message transmission to prevent the SPI slave device from trying to transmit a message while receiving another from the master (the slave polls the Chip Select signal before activating the DREADY signal). The functionality described is shown in Figure 3 and explained in detail in Section 6.3.1.

6.3 Message transmission examples

6.3.1 Slave to master transmission

A message transmission from the slave device running the BLE software stack to the master device requires the use of the additional flow control signal DREADY, which indicates a request from the slave to transmit a message to the master. This procedure is shown in Figure 3 and explained in detail in the following paragraphs:

1. Before message transmission, the slave DA14580 transmits a flow off byte in the same manner described in the previous section (using DREADY to request a transmission and waiting for the ACK byte to send the flow off byte) to prevent the master from sending any messages while slave will be transmitting the message.
2. The slave device activates DREADY again to request data transmission and the master device sends the ACK byte (the master device must send the ACK byte in every rising edge of the DREADY signal).
3. While transmitting the message, the slave device will keep the DREADY signal active. The master must decode the message header to extract the size of the payload and perform the necessary amount of SPI transfers to allow for the transmission of the whole message from the slave. The format of the message and the position of the payload size bytes have been described in detail in Section 5.
4. Upon the completion of the message transmission, the slave device will lower the DREADY signal and will activate it again shortly after to transmit a flow on byte. Again, an ACK byte from the master to indicate acknowledgement of the DREADY rising edge is needed. Every single transmission from the slave will be enclosed between a flow off and a flow on byte, which denotes a slave's inactive period (i.e. a period during which the master cannot transmit any messages to the slave).

The SPI master has to be able to decode messages and provide the SPI slave device with as many SPI accesses as needed to allow for the transmission of the whole message. The master device should also be capable of detecting the slave's active and inactive periods to avoid initiating a master-to-slave transmission after the slave has sent a flow off byte. The software included in the SDK meets the aforementioned requirements.

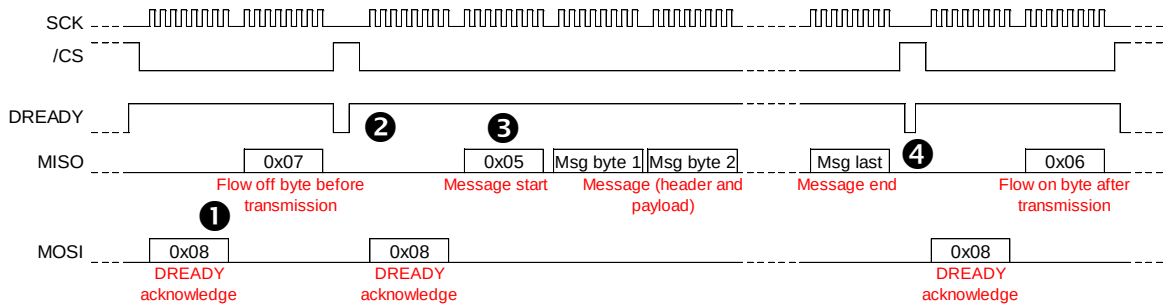

Figure 3: Flow control for slave-to-master transmission

6.3.2 Master-to-slave transmission

A message transmission from the SPI master to the DA14580 SPI slave is much simpler than slave-to-master transmission. A flow on byte must have been sent from the slave (using the DREADY signal). During a slave's active period, the master can initiate a message transmission at any time, keeping the chip select (/CS) signal active (low) throughout the message. After the last byte of the message has been transmitted, the master device must deactivate the chip select signal. This procedure is explained in detail in the following paragraphs and in Figure 4, where flow on and flow off bytes are included to indicate the slave's active period.

1. The SPI slave starts up or wakes from sleep and activates the DREADY signal.
2. The external host application transmits the ACK byte (0x08) to acknowledge the slave that it detected the DREADY high level.
3. The master initiates a transfer to allow the slave device to transmit the flow on byte, which denotes an active period of the slave DA14580.
4. After a flow on byte, the host application can initiate a message transmission at any time. The host application keeps the chip select signal active (low) throughout the entire message transmission.

The master device can send additional messages until the slave sends a flow off byte that indicates the end of the active period and the entrance of sleep mode or the beginning of a message transmission from slave to master.

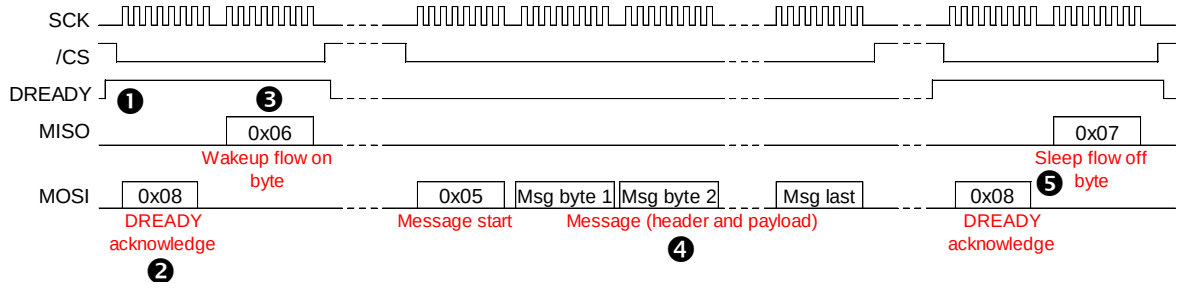

Figure 4: Flow control for master-to-slave transmission

7 Practical example of the external processor configuration

7.1 Hardware configuration

This section describes how to set up one DA14580 development board to run the proximity reporter as host application (external processor) in order to control another DA14580 development board, which implements the proximity profile BLE software stack.

7.1.1 Jumper setup for LED and button interface

Before testing any proximity monitoring alert events, the LED and push button interface needs to be set up properly.

On the DA14580 development board which will be used to run the proximity reporter host application device, PIN3 on connector J15 must be connected to PIN4 and PIN5 must be connected to PIN6. These connections enable the D1 green LED and push button K1 as a user interface.

The proximity reporter will notify the user when an alert indication, link loss and an immediate alert is triggered. The alert notification will be as follows:

- **High level alert:** fast (500 ms) green LED blinking.
- **Mild alert:** slow (1500 ms) green LED blinking.

The user can stop alert notification by pressing the push button K1.

7.1.2 Connection diagram

To enable the SPI communication between the two DA14580 devices, the following connections between the two boards have to be made:

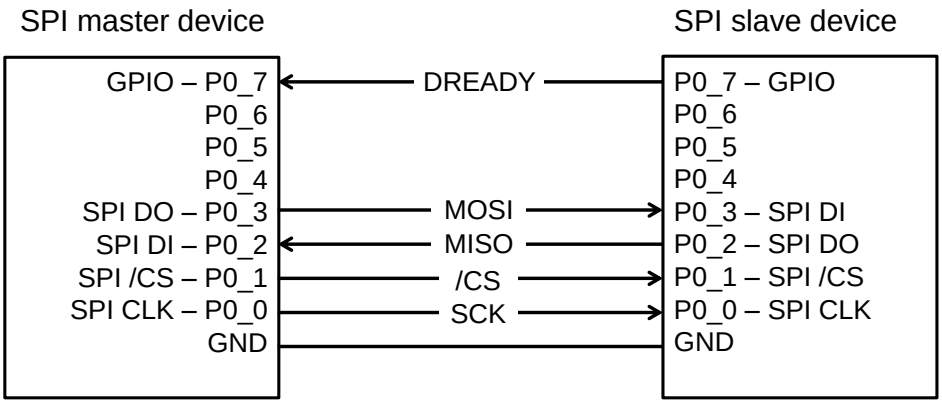

Figure 5: Connections for the external CPU interface over SPI

Note: The wire connections between the two devices need proper termination (especially the clock signal) and shielding to prevent transmission errors caused by signal reflections and electromagnetic interference.

The implemented external processor configuration over SPI employs an additional flow control signal from the DA14580 slave to the DA14580 master, the DREADY signal, which is activated by the slave to indicate a transmit request.

The SPI MOSI signal is mapped on pin P0_3, so the jumpers on connector J26 that connect the UART RTS/CTS signals to pins P0_2 and P0_3, have to be removed.

7.2 Running the projects

7.2.1 Starting the external processor device

The proximity reporter host application project is stored in the folder:
`host_apps\da14580\proximity\reporter\host_proxr.uvproj`

The project options are defined in file
`host_apps\da14580\proximity\reporter\da14580_config.h`

The project should be built (F7) and the steps in the DA14580 user guide to load the executable to the SDK must be followed. Then the executable needs to be started.

7.2.2 Starting the BLE device

There are two ways to enable the slave device for external CPU interface over the SPI:

1. Boot from the external SPI master device:

   The proximity reporter host application project includes an SPI booter functionality, which provides the slave device with an image of an integrated proximity reporter with external CPU interface over the SPI driver.

   If the flag **SPI_BOOTER** is defined in file
   `host_apps\da14580\proximity\reporter\da14580_config.h`
   the SPI booter will be enabled and the slave device will boot from the external SPI master device.

2. Load proximity reporter profile:

   If the flag **SPI_BOOTER** is not defined, then the SPI booter will be disabled and the user will be able to load the proximity reporter BLE software stack from another interface like UART or SWD. If this is the case, the host application needs to be started prior to connecting the two boards with a cable.

   In this case the Keil environment should be started and the following project should be opened and executed to download the firmware using the JTAG interface:
   `dk_apps\keil_projects\proximity\reporter_fe_spi\fe_proxr.uvproj`

More information how to build, open and execute a Keil project is given in Ref. [4].

In either case, when the above steps are followed, the proximity reporter host application device should end up in advertising mode.

Note: The master device should be started prior to starting the slave device.

Appendix A Replacing external CPU interface over UART by SPI

The API for the external CPU interface over the SPI driver included in the SDK is the same as the external CPU interface over the UART API, except for the function names. The equivalence is as follows:

uart_read ⇔ spi_read
uart_write ⇔ spi_write
uart_flow_on ⇔ spi_flow_on
uart_flow_off ⇔ spi_flow_off

The external CPU interface over the SPI driver's functions are called with the same arguments and return the same values as the external CPU interface over the UART functions.

To replace the built-in driver for the external CPU interface over the UART that resides in the DA14580 ROM, the following steps were taken:

1. In file *rom_symdef_hci_spi.txt* (`dk_apps/misc/rom_symdef_hci_spi.txt`) the symbol definitions that refer to functions

   ```
   SPI_Handler
   rwip_eif_get_func
   ```

   were commented out.

2. The file *arch_hci_spi.c* was added, which contains:
   a. The SPI interface structure definition:

   ```
   // Creation of SPI external interface api
   const struct rwip_eif_api spi_api =
   {
       spi_read_func,
       spi_write_func,
       spi_flow_on_func,
       spi_flow_off_func,
   };
   ```

   b. The association of the external interface with the SPI API defined previously:

   ```
   const struct rwip_eif_api* rwip_eif_get_func(uint8_t type)
   {
       const struct rwip_eif_api* ret = NULL;
       switch(type)
       {
           case RWIP_EIF_AHI:
           case RWIP_EIF_HCIC:
           {
               ret = &spi_api;
           }
           break;
           default:
           {
               ASSERT_ERR(0);
           }
           break;
       }
       return ret;
   }
   ```

3. A call to function *spi_init_func* was added in function *periph_init*, to enable the SPI module during system initialisation or upon wake-up.

8 Revision history

Revision	Date	Description
1.0	05-Jun-2014	Initial version.
1.1	24-Jun-2014	Corrected DI and DO pins as input and output respectively in Table 1.

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.

Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, unless otherwise stated.

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

Germany Headquarters
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

United Kingdom
Dialog Semiconductor (UK) Ltd
Phone: +44 1793 757700

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5425 4567

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Singapore
Dialog Semiconductor Singapore
Phone: +65 64 849929

China
Dialog Semiconductor China
Phone: +86 21 5178 2561

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8291