User manual

DA14580 GTL interface in integrated processor applications

UM-B-017

Abstract

This document describes the communication of the application task running in a DA14580 integrated processor application with an application running on an external processor over GTL/UART interface.

Contents

Figures

Tables

1 Terms and definitions

API	Application Programming Interface
BLE	Bluetooth Low Energy
CTS	Clear To Send
GPIO	General Purpose Input/Output
GTL	Generic Transport Layer
RTS	Request To Send
SDK	Software Development Kit
UART	Universal Asynchronous Receiver/Transmitter

2 References

1. UM-B-010, DA14580 Proximity application example in an integrated- and external-processor solution, Dialog Semiconductor.
2. UM-B-004, DA14580 Peripheral Drivers, Dialog Semiconductor.
3. UM-B-015, DA14580 Software architecture, Dialog Semiconductor.

3 Introduction

The Generic Transport Layer (GTL) protocol facilitates using the UART interface to implement the communication between the BLE or profile tasks running on the DA14580 and an external processor application. The usage of the GTL in external processor mode is given in [1].

The implementation details enabling the message exchange between the application task running in a DA14580 application in integrated processor mode and an application running on an external processor over the existing GTL/UART interface are described.

4 Implementation overview

The application task sends messages to the GTL task through the kernel message routing mechanism and the GTL forwards the messages over the UART interface to the external processor.

Both DA14580 and external processor can enter sleep mode. The UART interface of DA14580 supports an RTS/CTS flow control mechanism for synchronisation of the devices. Besides the UART interface an additional general purpose signal can optionally be used to wake up the external processor.

On the reception path the GTL task receives messages from the UART interface and forwards them to the application task.

The aforementioned message flow is outlined in Figure 1.

External processor application must be able to send and receive messages in GTL format.

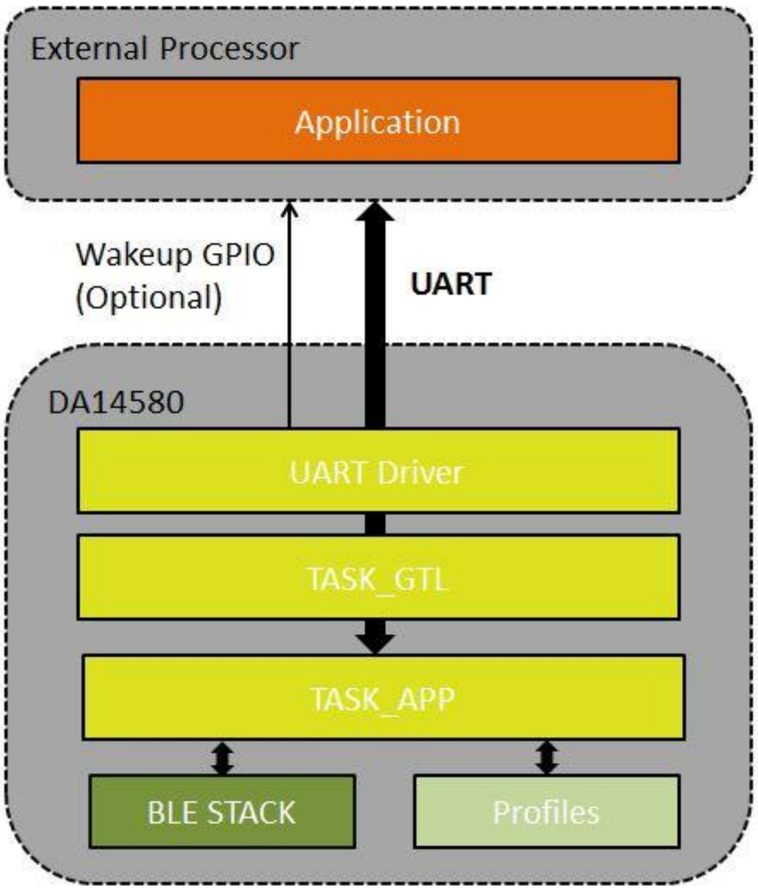

Figure 1: Implementation overview

The following directives are added in project configuration header file (da14580_config.h) for the control of the supported functionality.

Table 1: Configuration directives

Directive	Description
CFG_INTEGRATED_HOST_GTL	If defined, enables the communication over GTL/UART interface of application task in integrated processor application with an external processor.
CFG_WKUP_EXT_PROCESSOR	If defined, enables the wakeup GPIO output signal for external processor. The *dk_apps\src\plf\refip\src\driver\uart\uart_ext_wkup.c* file must be included in project.

5 GTL Interface

The transport layer interface defines the protocol and the packets exchanged between the external processor application and the BLE stack in DA14580.

The packet format is illustrated in Figure 2. A single byte identifier (**ID**) at the beginning of the packet is used to identify the packet. The value of the identifier (ID) is 0x5 for both incoming and outgoing packets. The structure of the **packet data** is given in Table 1. The packet format is the same for sending and receiving data. The protocol only uses commands for sending data from host application to BLE stack and events in the opposite direction.

The field **Type,** in the packet format, is 2 bytes long and defines a valid message type in application task for the communication with the external processor application. Fields **SRC_ID** and **DST_ID** define the source and destination task of the message respectively. The external processor application can only communicate with application task for this implementation; hence these fields must always have the following values:

- DA14580 Outgoing Messages: DST_ID = TASK_GTL, SRC_ID = TASK_APP
- DA14580 Incoming Messages: DST_ID = TASK_APP, SRC_ID = TASK_GTL

The field **PAR_LEN** is 2 bytes long and defines the size, in bytes, of the message **parameters**. The content and the size of the message parameters depend on the specific message and must be defined in application task source code.

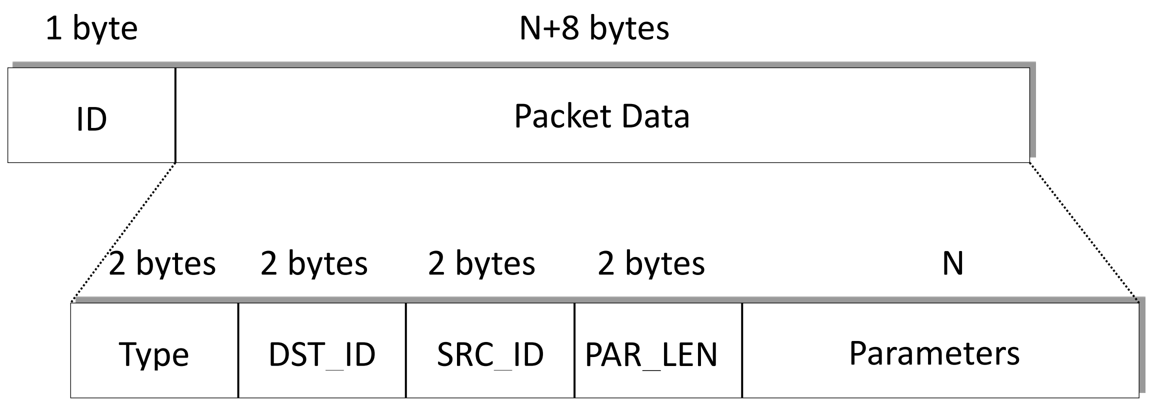

Figure 2: Packet format

Table 2: Message fields

Field	Size	Description
Type	2	Message Identification. A valid Message ID of application task API.
DST_ID	2	Destination Task Identification. Incoming: TASK_APP Outgoing: TASK_GTL
SRC_ID	2	Source Task Identification. Incoming: TASK_GTL Outgoing: TASK_APP
PAR_LEN	2	Message Parameter Length. The size, in bytes, of the Parameters field.
Parameters	N	Message Parameters. Format of this field depends on the message Type (MSG_ID).

6 Application messages

Application task uses the message exchanging mechanism of the kernel for the communication with external processor application. Message types, formats and handlers must be defined in application task source code. Number and format of messages are application specific and vary depending on the specification of the communication protocol.

An example of an external application which sets DA14580 application in scan mode and receives the information of the discovered devices will be used in the next chapters. For this example it is assumed that messages in Table 3 will be used.

Table 3: Scan mode example messages

Message	Direction	Data
Start scanning command	External app to DA14580	No Data
Discovered device info indication	DA14580 to external app	Address type Device address
Scan procedure completed indication	DA14580 to external app	No Data

6.1.1 Types

In DA14580 SDK distribution types of application task specific messages are defined by the APP_MSG enumeration in app_api.h header file. The developer must add the definition of all additional message types that implement the interaction with external application in this enumeration.

For the example of scan procedure, the developer must apply the following changes in APP_MSG enumeration:

```
enum APP_MSG
{
        APP_MODULE_INIT_CMP_EVT = KE_FIRST_MSG(TASK_APP),
        …
        APP_EXT_SCAN_CMD,
        APP_EXT_ADV_REPORT_IND,
        APP_EXT_SCAN_CMP_IND,
}
```

6.1.2 Format

The structure of data format must be defined for each of the messages. In the example of scan procedure only the APP_EXT_ADV_REPORT_IND message contains data; hence only one data structure is necessary to be defined:

```
struct app_ext_adv_report_ind
{
    uint8_t addr_type;
    uint8_t addr[6];
};
```

6.1.3 Handlers

A message handler function must be defined for each incoming application task message. A sample handler of APP_EXT_SCAN_CMD is demonstrated bellow. In this handler, the application task builds a GAPM_START_SCAN_CMD with the desired scan procedure parameters and sends it to TASK_GAPM:

```
int app_ext_scan_cmd_handler(ke_msg_id_t const msgid,
                                         void const *param,
                                         ke_task_id_t const dest_id,
                                         ke_task_id_t const src_id)
{

    struct gapm_start_scan_cmd *msg = KE_MSG_ALLOC(GAPM_START_SCAN_CMD,
                                 TASK_GAPM, TASK_APP, gapm_start_scan_cmd);


    msg->mode = GAP_GEN_DISCOVERY;
    msg->op.code = GAPM_SCAN_ACTIVE;
    msg->op.addr_src = GAPM_PUBLIC_ADDR;
    msg->filter_duplic = SCAN_FILT_DUPLIC_EN;
    msg->interval = 10;
    msg->window = 5;

    ke_msg_send(msg);

return (KE_MSG_CONSUMED);
}
```

The APP_EXT_SCAN_CMD message handler must be added in app_default_state array in app_task_handlers.h.

```
EXTERN const struct ke_msg_handler app_default_state[] =
{
{GAPM_DEVICE_READY_IND,           (ke_msg_func_t)gapm_device_ready_ind_handler},
{GAPM_CMP_EVT,                    (ke_msg_func_t)gapm_cmp_evt_handler},
       ...
{APP_EXT_SCAN_CMD,                (ke_msg_func_t)app_ext_scan_cmd_handler},
}
```

6.1.4 Message transmission

Messages oriented to external application must be sent through the message exchanging mechanism of the kernel to TASK_GTL.

In the example of scanning procedure, application sends to the external application an APP_EXT_ADV_REPORT_IND message upon each reception of a GAPM_ADV_REPORT_IND and an APP_EXT_SCAN_CMP_IND upon the completion of scan operation:

```
int gapm_adv_report_ind_handler(ke_msg_id_t const msgid,
                                struct gapm_adv_report_ind const *param,
                                ke_task_id_t const dest_id,
                                ke_task_id_t const src_id)
{

       struct app_ext_adv_report_ind *cmd = KE_MSG_ALLOC(APP_EXT_ADV_REPORT_IND,
TASK_GTL,TASK_APP,app_ext_adv_report_ind);
       cmd->addr_type = param-> adv_addr_type;
       memcpy ((void *)cmd->addr, (void *)&param->report.adv_addr, BD_ADDR_LEN);
       // Send the message
ke_msg_send(cmd);
       return (KE_MSG_CONSUMED);
}

int gapm_cmp_evt_handler(ke_msg_id_t const msgid,
                         struct gapm_cmp_evt const *param,
                         ke_task_id_t const dest_id,
                         ke_task_id_t const src_id)
{
       switch(param->operation)
       {
              …
              case GAPM_SCAN_ACTIVE:
              case GAPM_SCAN_PASSIVE:
              struct app_ext_scan_cmp_ind *cmd = ke_msg_alloc(APP_EXT_SCAN_CMP_IND,
                                                   TASK_GTL, TASK_APP, 0);
              // Send the message
              ke_msg_send(cmd);
              break;
              …
       }
       return (KE_MSG_CONSUMED);
}
```

6.1.5 Example project

The developer can use as a reference of usage of GTL interface in an integrated processor application, the central role throughput evaluation project included in SDK (release version 3.0.4 or later). This project uses an extended list of messages exchanged between external application and DA14580 application task, to create an application for the evaluation of max data throughput mechanism.

The project file can be found under the dk_apps\keil_projects\throughput_eval\throughput_eval_central_fh directory.

7 Sleep architecture

7.1.1 DA14580 sleep

DA14580 supports sleep mode in order to reduce power consumption during inactive periods. During sleep periods DA14580 application cannot receive messages on UART/GTL interface. System wakes up to service BLE events or other kernel events, e.g. kernel timer expiration. At system wakeup UART/GTL interface is activated automatically and communication over GTL is enabled. RTS/CTS handshaking is used to control the message flow on UART port.

If GTL interface is enabled there is a maximum sleep period that DA14580 can remain in sleep mode. If the time to next scheduled active period is above a defined threshold, DA14580 will wake up to service GTL interface and receive external processors messages. This threshold is configurable and controlled by the value of MAX_SLEEP_DURATION_PERIODIC_WAKEUP directive in rwip.c. The default value is 500 ms.

7.1.2 External processor wakeup

Additionally to the UART, a GPIO line in output mode can be used by DA14580 application to send a wakeup signal to external processor. The idle state of wakeup GPIO is low and DA14580 toggles the state whenever there is a transmission ready on UART port. Wakeup GPIO toggling is displayed on Figure [3].

GPIO port used for this wakeup signal is configurable and controlled by the following directives in periph_setup.h header file:

```
#define EXT_WAKEUP_PORT     0
#define EXT_WAKEUP_PIN      7
```

The activation of the wakeup signal is optional and is enabled by defining CFG_EXT_PROCESSOR_WKUP directive in da14580_config.h configuration file.

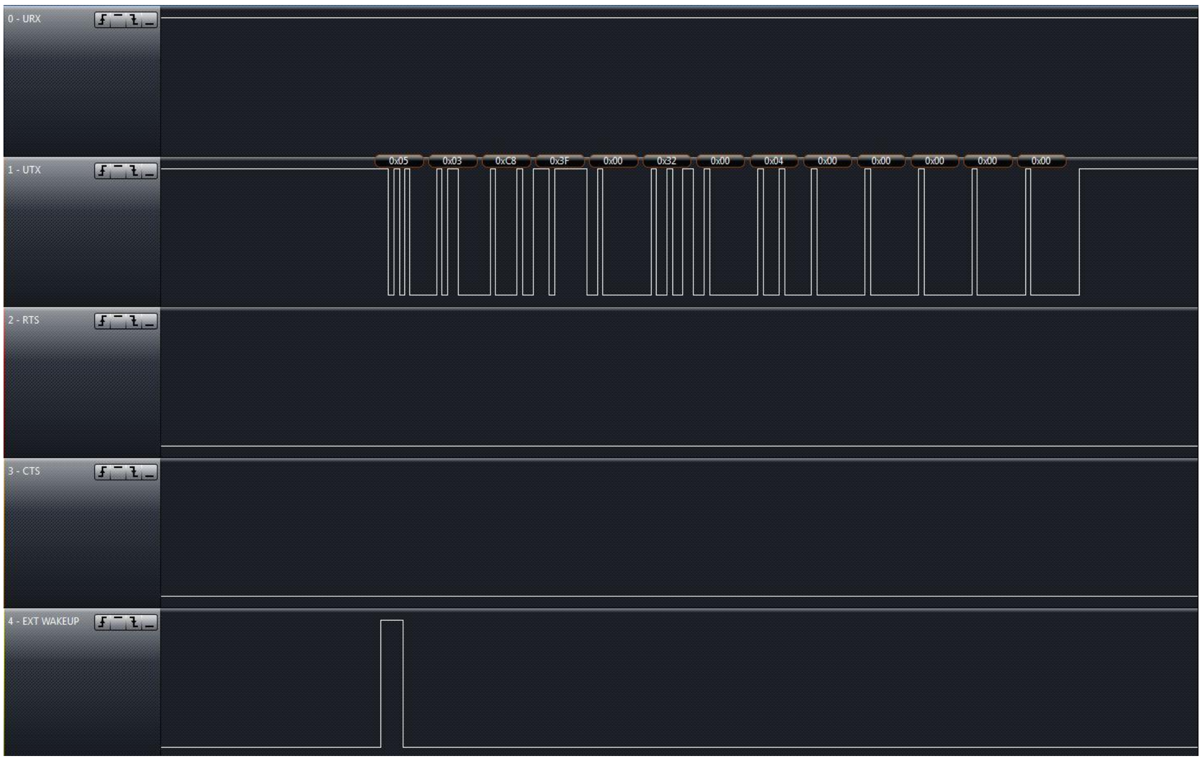

Figure 3: External processor wakeup signal

8 Revision history

Revision	Date	Description
1.0	17-Jul-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

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	North America *Dialog Semiconductor Inc.* Phone: +1 408 845 8500	Singapore *Dialog Semiconductor Singapore* Phone: +65 64 849929
United Kingdom *Dialog Semiconductor (UK) Ltd* Phone: +44 1793 757700	Japan *Dialog Semiconductor K. K.* Phone: +81 3 5425 4567	China *Dialog Semiconductor China* Phone: +86 215178 2561
The Netherlands *Dialog Semiconductor B.V.* Phone: +31 73 640 8822	Taiwan *Dialog Semiconductor Taiwan* Phone: +886 281 786 222	Korea *Dialog Semiconductor Korea* Phone: +82 2 3469 8291
Email: enquiry@diasemi.com	Web site: www.dialog-semiconductor.com	