CSR μEnergy®

Keyboard

Application Note

Issue 6

Document History

Revision	Date	History
1	09 FEB 13	Original publication of this document
2	20 DEC 12	Updated to reference CSR1010 and CSR1011 devices
3	11 FEB 13	Updated for SDK 2.1
4	11 FEB 13	Corrections to Table 8.1
5	20 FEB 13	Updated current consumption values
6	10 MAY 13	Updated for SDK 2.2; connection parameter update

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

Contents

Tables, Figures and Equations

1. Introduction

This document describes the Keyboard application supplied with the CSR µEnergy® Software Development kit (SDK) and provides guidance to developers on how to customise the on-chip application.

The application demonstrates the HID over GATT profile which is specified by the Bluetooth SIG.

1.1. Application Overview

1.1.1. Profiles Supported

The Keyboard application supports the HID over GATT Profile.

1.1.1.1. HID over GATT Profile

The HID over GATT profile defines protocols, procedures and features to be used by HID devices and HID Hosts using GATT profile:

Role	Description
HID Device	A HID device is a device that sends Input reports and associated HID data to a HID host.
HID Host	A HID host is a device that receives Input reports from HID devices.

Table 1.1: HID Profile Roles

1.1.2. Application Topology

The Keyboard application implements the HID over GATT profile in HID Device role, see Table 1.2.

Role	HID over GATT Profile	GAP Service	GATT Service	Device Information Service	Battery Service	Scan Parameters Service
GATT Role	GATT Server	GATT Server	GATT Server	GATT Server	GATT Server	GATT Server
GAP Role	Peripheral	Peripheral	Peripheral	Peripheral	Peripheral	Peripheral

Table 1.2: Application Topology

Role	Responsibility
GATT Server	It accepts incoming commands and requests from the client and sends responses, indications and notifications to the client.
GAP Peripheral	It accepts connection requests from the remote device and acts as a slave in the connection.

Table 1.3: Responsibilities

For more information on the GATT server and GAP peripheral, see *Bluetooth Core Specification Version 4.0*.

1.1.3. Services

This application exposes the following services:

- HID (Version 1.0)
- Device Information (Version 1.1)
- Battery (Version 1.0)
- Scan Parameters (Version 1.0)
- GAP
- GATT

In addition to the above services, the Keyboard application optionally supports a proprietary HID boot service to remain compatible with early CSR HID implementations. See section 7.8 for details.

The HID over GATT profile mandates three services: Human Interface Device (HID), Battery and Device Information. GAP and GATT services are mandated by *Bluetooth Core Specification Version 4.0*. Scan Parameter services and HID boot Service are optional as shown in Figure 1.1.

For more information on HID, Device Information, Battery and Scan parameters services, see *HID Service Specification Version 1.0*, *Device Information Service Specification Version 1.1*, *Battery Service Specification Version 1.0* and *Scan Parameters Service Specification Version 1.0* respectively. For more information on GATT and GAP services, see *Bluetooth Core Specification Version 4.0*.

See Appendix B for information on characteristics supported by each service.

Note:

The Keyboard application does not support any characteristic for the GATT service. See *Bluetooth Core Specification Version 4.0* for more information.

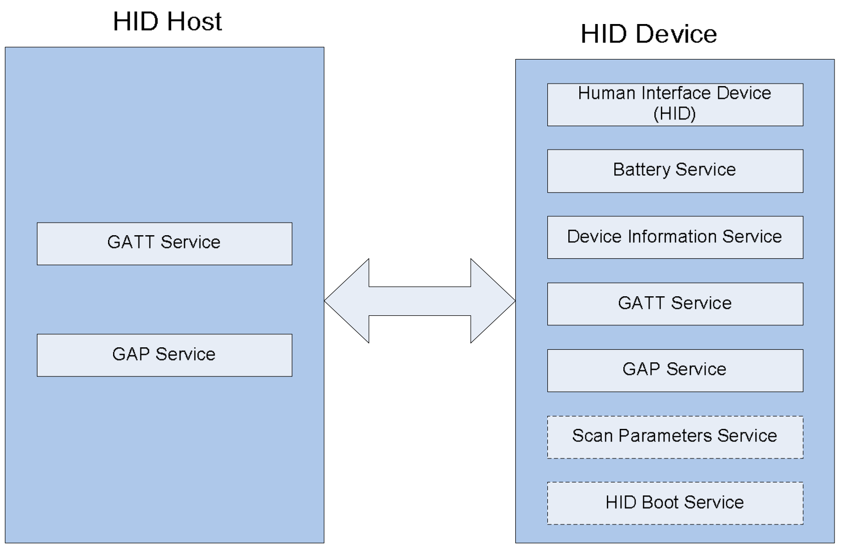


Figure 1.1: Primary Services

2. Using the Application

This section describes how the Keyboard application can be used with the CSR µEnergy Profile Demonstrator application.

2.1. Demonstration Kit

The application can be demonstrated using the following components:

Component	Hardware	Application
HID Device	CSR10x1 Development Board	Keyboard Application
HID Host	CSR µEnergy BT4.0 USB dongle	CSR µEnergy Profile Demonstrator Application

Table 2.1: Demonstration Components

The CSR µEnergy Profile Demonstrator application, CSR device driver and installation guide are included in the SDK.

2.1.1. HID Device

The SDK is used to build and download the Keyboard application to the development board. See the *CSR µEnergy xIDE User Guide* for further information.

Ensure the development board is switched on using the Power slider switch. During normal use, set the switch to the **Batt** position to use the coin cell battery. Figure 2.1 shows a CSR1001 development board with the switch set to receive power from the mini-USB connector when debugging and downloading the firmware image. Figure 2.2 shows a CSR1011 board with the switch set to receive power from the battery.

Note:

When the mini-USB cable has been disconnected, wait at least 1 minute before switching the board to receive power from the coin cell. This allows any residual charge received from the SPI connector to be dissipated.

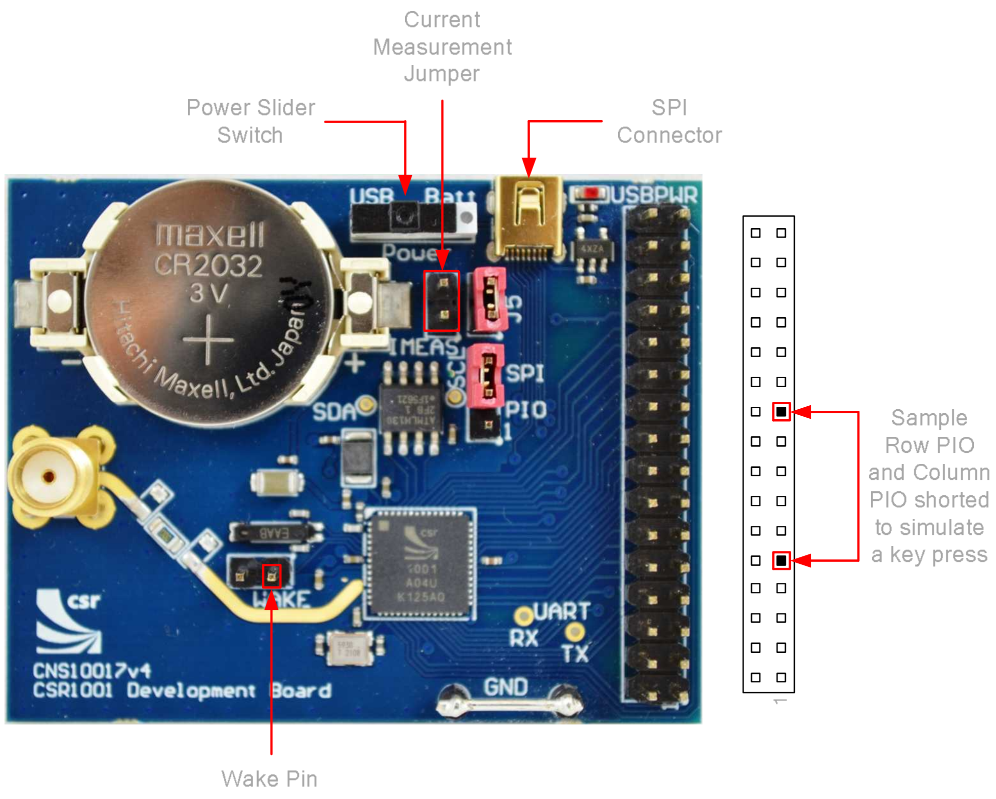

Current Measurement Jumper
Power Slider Switch
SPI Connector
Sample Row PIO and Column PIO shorted to simulate a key press
Wake Pin
maxell CR2032 3 V Hitachi Maxell, Ltd. Japan
USB Batt
Power
USBPWR
IMEAS
SPI
PIO
SDA
WAKE
UART RX TX
GND
CNS10017v4 CSR1001 Development Board

Figure 2.1: CSR1001 Development Board

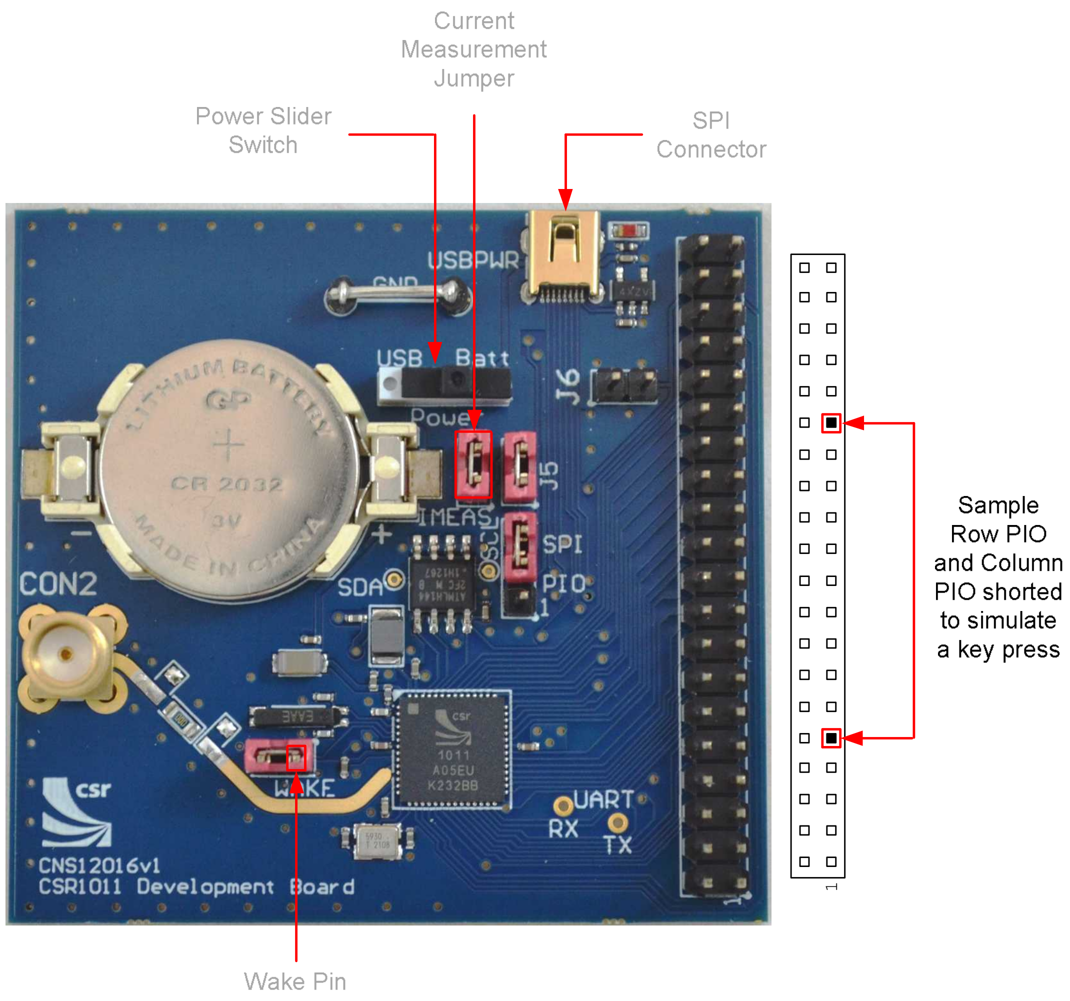

Current Measurement Jumper
Power Slider Switch
SPI Connector
Sample Row PIO and Column PIO shorted to simulate a key press
Wake Pin

Figure 2.2: CSR1011 Development Board

2.1.1.1. User Interface

This application uses the PIOs exposed on the development board as the rows and columns of the keyboard device. Figure 2.3 details the hardware configuration used in the Keyboard application.

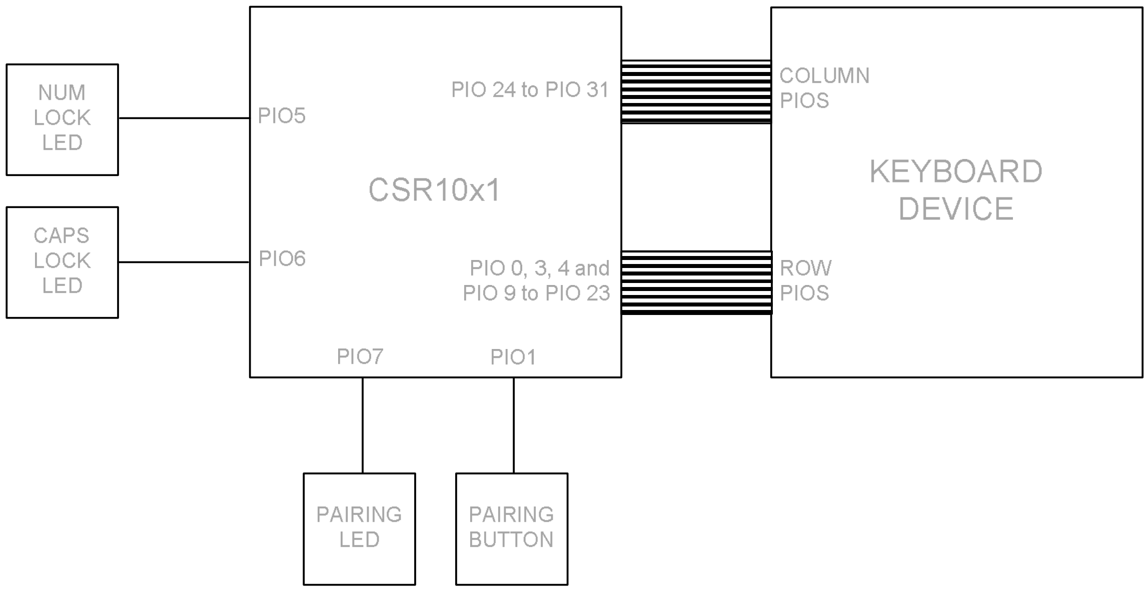


Figure 2.3: Hardware Configuration in Keyboard

Table 2.2 maps the PIOs exposed on the CSR1001 development board to rows and columns as configured in the Keyboard application. Table 2.3 maps the rows, columns, and additional buttons and LEDs, to PIOs exposed on the CSR1011 development board.

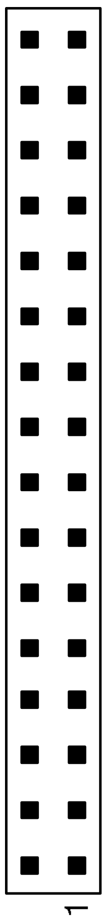
CSR1001 PIO Pinout	Usage in Keyboard	Dev Board Pinout		Dev Board Pinout	Usage in Keyboard	CSR1001 PIO Pinout
						
GND		32		31		AIO2
GND		30		29		AIO1
VDD_PAD		28		27		AIO0
VDD_PAD		26		25	COLUMN 7	PIO31
PIO30	COLUMN 6	24		23	COLUMN 5	PIO29
PIO28	COLUMN 4	22		21	COLUMN 3	PIO27
PIO26	COLUMN 2	20		19	COLUMN 1	PIO25
PIO24	COLUMN 0	18		17	ROW 17	PIO23
PIO22	ROW 16	16		15	ROW 15	PIO21
PIO20	ROW 14	14		13	ROW 13	PIO19
PIO18	ROW 12	12		11	ROW 11	PIO17
PIO16	ROW 10	10		9	ROW 9	PIO15
PIO14	ROW 8	8		7	ROW 7	PIO13
PIO12	ROW 6	6		5	ROW 5	PIO11
PIO10	ROW 4	4		3	ROW 3	PIO9
PIO4	ROW 2	2		1	ROW 1	PIO3
				UART Tx test point	ROW 0	PIO0

Table 2.2: 18 x 8 Keypad Mapping to Pins on CSR1001 Development Board

CSR1011 PIO Pinout	Usage in Keyboard	Dev Board Pinout		Dev Board Pinout	Usage in Keyboard	CSR1011 PIO Pinout
GND		40	■ ■	39		GND
VDD_PAD		38	■ ■	37		AIO2
I2C_SDA		36	■ ■	35		AIO1
I2C_SCL		34	■ ■	33		AIO0
PIO2	EEPROM PWR	32	■ ■	31	COLUMN 7	PIO31
PIO30	COLUMN 6	30	■ ■	29	COLUMN 5	PIO29
PIO28	COLUMN 4	28	■ ■	27	COLUMN 3	PIO27
PIO26	COLUMN 2	26	■ ■	25	COLUMN 1	PIO25
PIO24	COLUMN 0	24	■ ■	23	ROW 17	PIO23
PIO22	ROW 16	22	■ ■	21	ROW 15	PIO21
PIO8/SPI_MISO	LOW PWR LED	20	■ ■	19	PAIR LED	PIO7/SPI_MOSI
PIO20	ROW 14	18	■ ■	17	ROW 13	PIO19
PIO6/SPI_CSB	CAPS LOCK LED	16	■ ■	15	ROW 12	PIO18
PIO17	ROW 11	14	■ ■	13	NUM LOCK LED	PIO5/SPI_CLK
PIO16	ROW 10	12	■ ■	11	ROW 9	PIO15
PIO14	ROW 8	10	■ ■	9	ROW 7	PIO13
PIO1/UART_RX	PAIR BUTTON	8	■ ■	7	ROW 0	PIO0/UART_TX
PIO12	ROW 6	6	■ ■	5	ROW 5	PIO11
PIO10	ROW 4	4	■ ■	3	ROW 3	PIO9
PIO4/SF_CSB	ROW 2	2	■ ■	1	ROW 1	PIO3/SF_DOUT
			1			

Table 2.3: 18 x 8 Keypad, buttons and LED Mapping to Pins on CSR1011 Development Board

Key Press Behaviour

To simulate a key press, a PIO corresponding to a row and a PIO corresponding to a column must be shorted together.

PIO0, PIO3, PIO4 and PIO9 to PIO23 are configured as columns and PIO24 to PIO31 are configured as rows of the keyboard in the PIO controller code (`pio_ctrlr_code.asm`).

When a Key Press is detected, an Input report corresponding to the pressed key is buffered in a circular queue.

- If the Keyboard application is connected to a HID host and notifications are configured on Input reports, then the Input reports held in the circular queue are notified to the remote host.
- If the Keyboard application is not connected to any HID host, it starts advertisements.
- See section 5 for details on sending a key press to the HID host.
- See Figure 2.4 for the state transitions resulting from a Key Press.

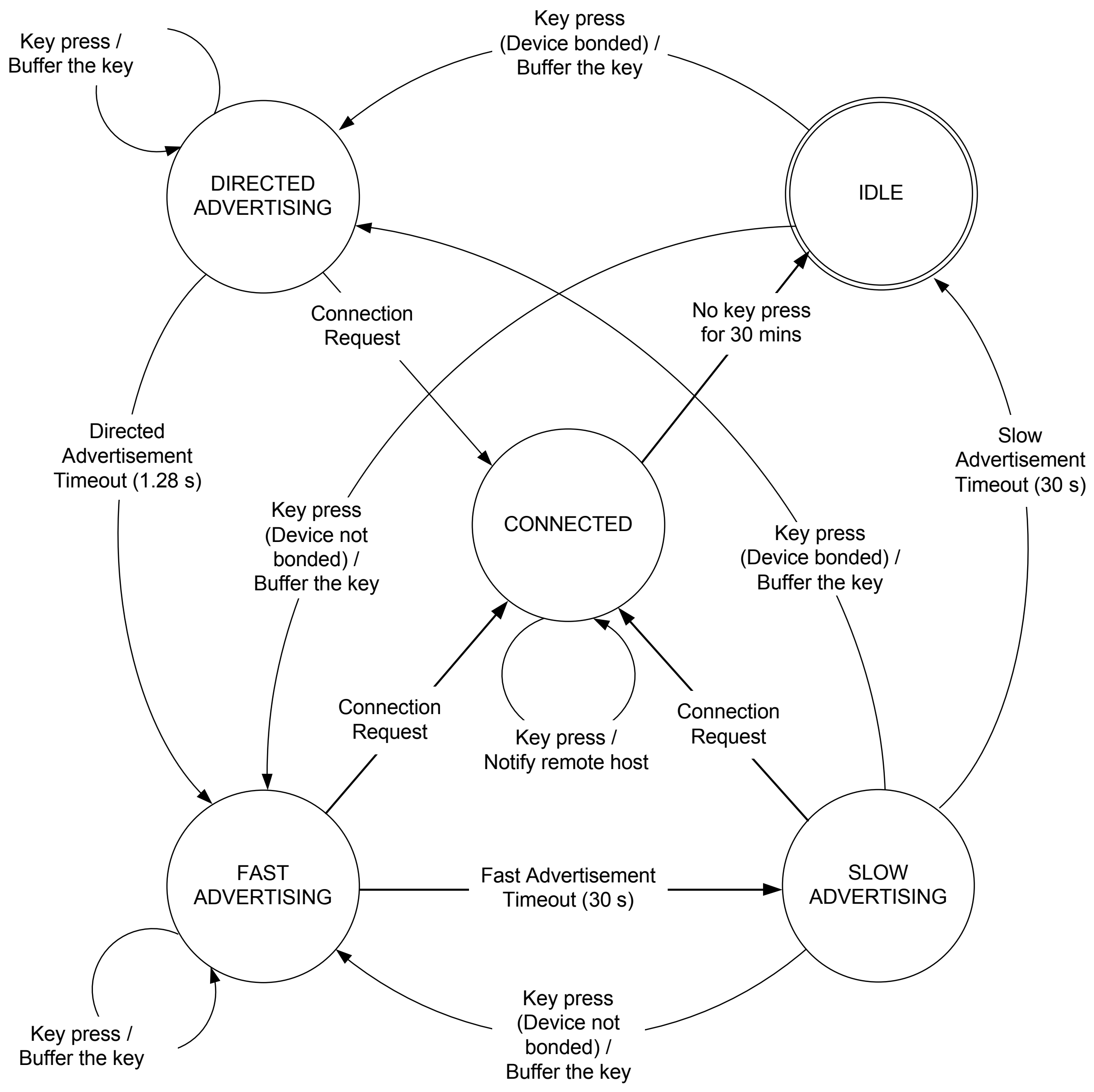

Key press /
Buffer the key
Key press
(Device bonded) /
Buffer the key
DIRECTED ADVERTISING
IDLE
Connection Request
No key press for 30 mins
Directed Advertisement Timeout (1.28 s)
Slow Advertisement Timeout (30 s)
Key press (Device not bonded) / Buffer the key
CONNECTED
Key press (Device bonded) / Buffer the key
Connection Request
Key press / Notify remote host
Connection Request
FAST ADVERTISING
Fast Advertisement Timeout (30 s)
SLOW ADVERTISING
Key press / Buffer the key
Key press (Device not bonded) / Buffer the key

Figure 2.4: Key Press Behaviour

Note:

If privacy is enabled (see section 7.7) and the bonded HID host has written to the re-connection address characteristic, then the `DIRECTED ADVERTISING` state is entered before entering `FAST ADVERTISING` state.

Pairing Button Press Behaviour

The Keyboard application configures PIO1 as the button used to remove the pairing information. PIO1 is visible as the **UART_RX** test point on the CSR1001 development board. To simulate the Pairing button press on the CSR10x1 development board, PIO1 must be connected to ground for more than a second.

The Pairing Button Press event is registered when the pairing button is pressed and held down for a period of 1 second. On registering this event,

- If `CONNECTED`, the Keyboard application removes the bonding information and triggers a disconnection with the remote HID Host.
- If `IDLE`, the Keyboard application removes the bonding information and triggers fast advertisements.
- If `ADVERTISING`, the Keyboard application removes the bonding information and starts fast advertisements.

The pairing LED starts blinking once the bonding is removed and the advertisements are started.

See Figure 2.5 for the pairing removal button press behaviour.

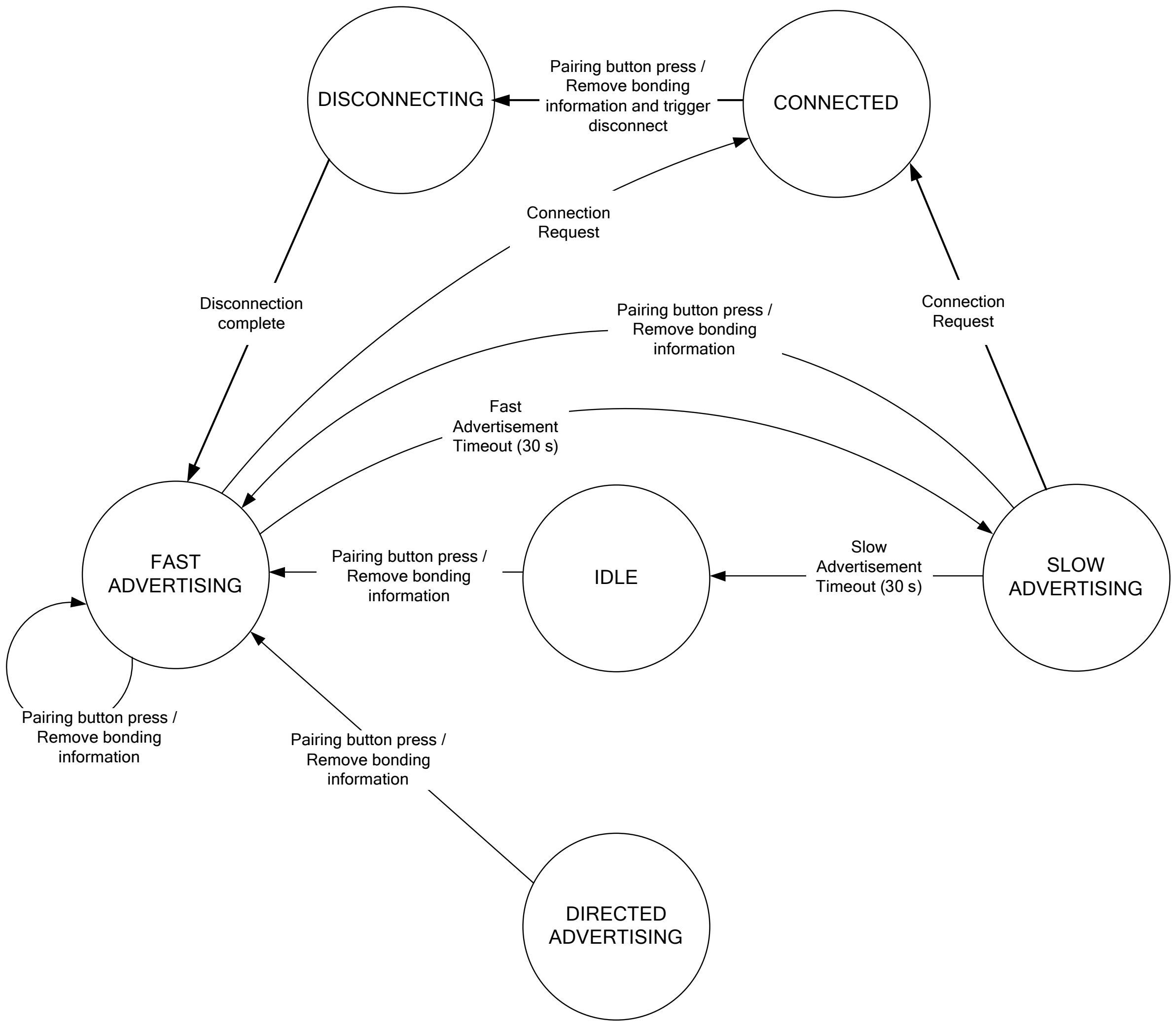

DISCONNECTING
Pairing button press / Remove bonding information and trigger disconnect
CONNECTED
Connection Request
Disconnection complete
Pairing button press / Remove bonding information
Connection Request
Fast Advertisement Timeout (30 s)
FAST ADVERTISING
Pairing button press / Remove bonding information
IDLE
Slow Advertisement Timeout (30 s)
SLOW ADVERTISING
Pairing button press / Remove bonding information
Pairing button press / Remove bonding information
DIRECTED ADVERTISING

Figure 2.5: Pairing Button Press Behaviour

2.1.2. HID Host

2.1.2.1. CSR µEnergy BT4.0 USB Dongle

The CSR µEnergy BT4.0 USB dongle shown in Figure 2.6 can be used with the CSR µEnergy Profile Demonstrator application to complete the Bluetooth Smart link between two devices. To use the USB dongle, the default Bluetooth Windows device driver must be replaced with the CSR BlueCore device driver as described in the *Installing the CSR Driver for the Profile Demonstrator Application* user guide.

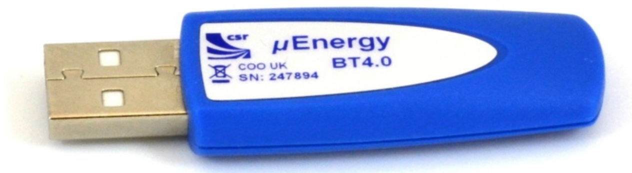
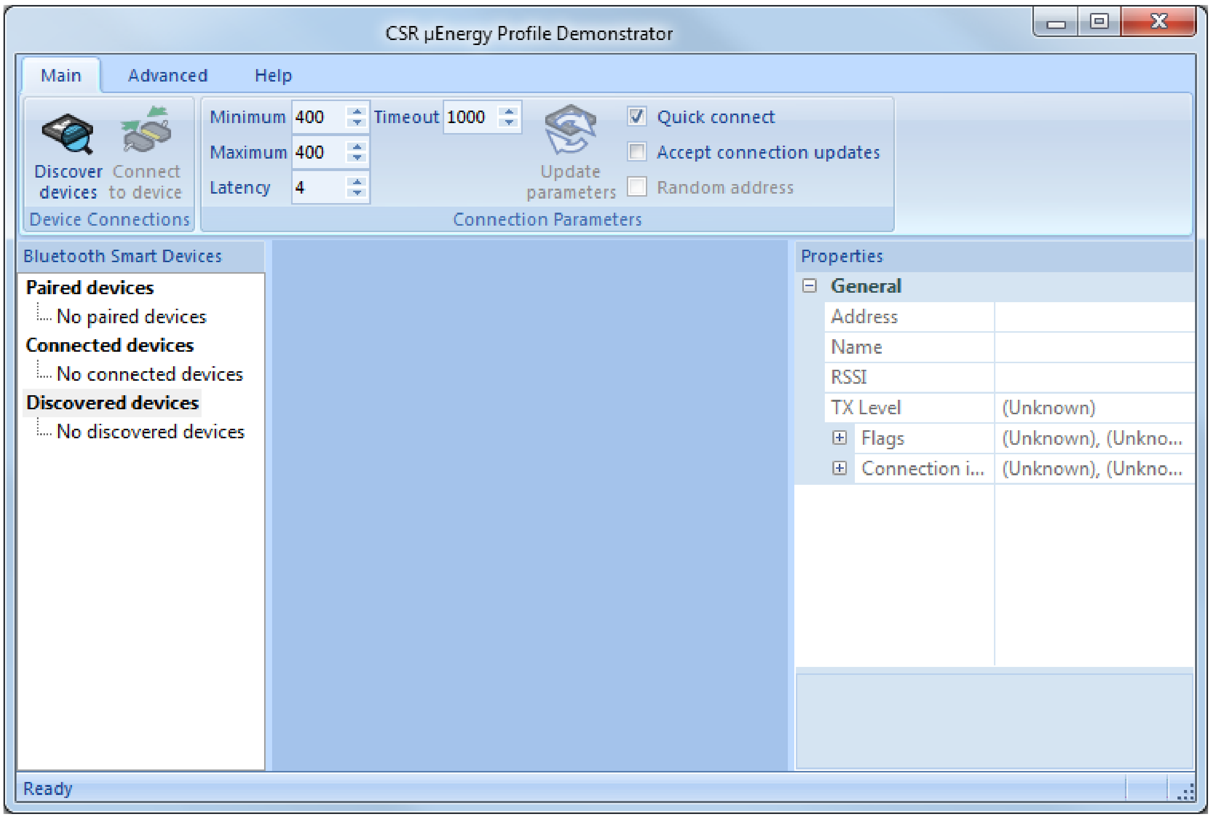


Figure 2.6: CSR µEnergy BT4.0 USB Dongle

2.1.2.2. CSR µEnergy Profile Demonstrator Application

The CSR µEnergy Profile Demonstration application is compatible with a PC running Windows XP, Windows 7 (32-bit and 64-bit) or Windows 8 (32-bit and 64-bit). Launch the application once the USB dongle is attached to the PC and the driver has been loaded.


CSR µEnergy Profile Demonstrator

Main | Advanced | Help

Device Connections: Discover devices, Connect to device

Connection Parameters:
Minimum 400
Maximum 400
Latency 4
Timeout 1000
Update parameters
☑ Quick connect
☐ Accept connection updates
☐ Random address

Bluetooth Smart Devices
- Paired devices
  - No paired devices
- Connected devices
  - No connected devices
- Discovered devices
  - No discovered devices

Properties

General	
Address	
Name	
RSSI	
TX Level	(Unknown)
Flags	(Unknown), (Unkno...
Connection i...	(Unknown), (Unkno...

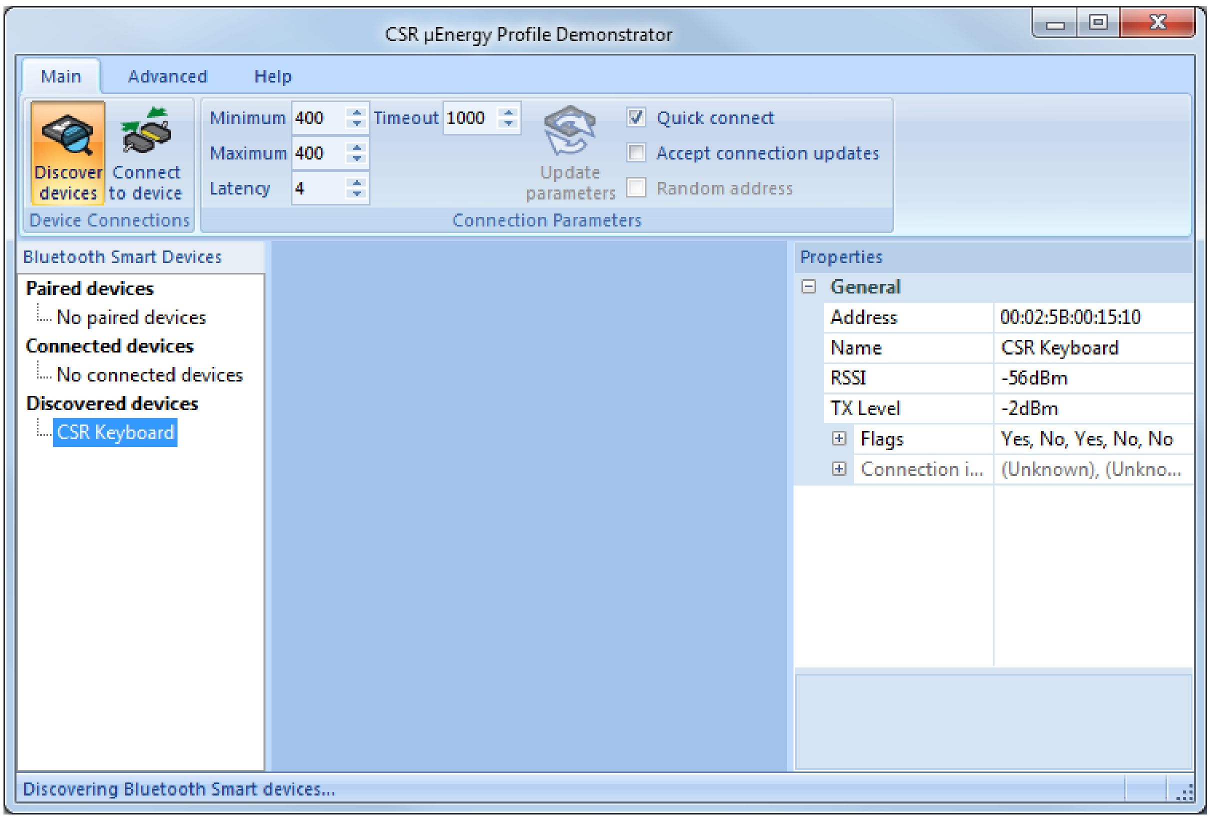
Ready

Figure 2.7: CSR µEnergy Profile Demonstrator

2.2. Demonstration Procedure

This section describes how to use the keyboard demonstration application:

1. Switch on the development board to trigger advertisements. If the advertisements have stopped without any connections being made, advertising can be restarted by shorting a row PIO with a column PIO.
2. Click on the **Discover devices** button in the **CSR µEnergy Profile Demonstrator** application window. The software searches for Bluetooth Smart devices and lists all the discovered devices on the left hand side of the application window, see Figure 2.8.


CSR µEnergy Profile Demonstrator

Main | Advanced | Help

Minimum 400 | Timeout 1000
Maximum 400
Latency 4

[✓] Quick connect
[ ] Accept connection updates
[ ] Random address

Bluetooth Smart Devices

- **Paired devices**
  - No paired devices
- **Connected devices**
  - No connected devices
- **Discovered devices**
  - CSR Keyboard

Properties

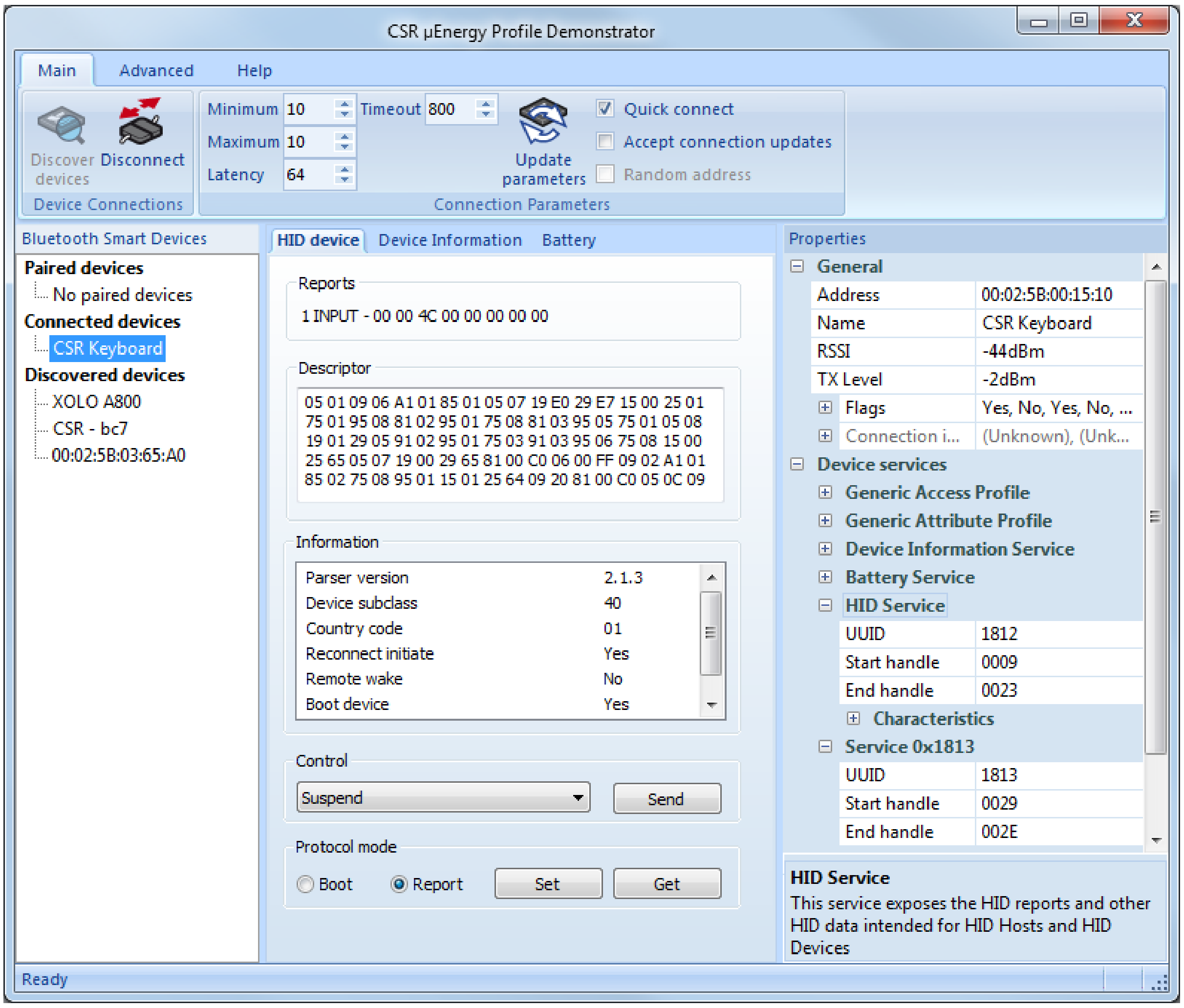
General

Address	00:02:5B:00:15:10
Name	CSR Keyboard
RSSI	-56dBm
TX Level	-2dBm
Flags	Yes, No, Yes, No, No
Connection i...	(Unknown), (Unkno...

Discovering Bluetooth Smart devices...

Figure 2.8: Keyboard Device Discovered

3. When the device labelled **CSR Keyboard** appears, select it and its device address appears on the right hand side of the screen. To connect to this device, click on the **Connect to device** button, see Figure 2.8. The CSR µEnergy Profile Demonstrator application displays a tabbed pane corresponding to different services supported by the device, see Figure 2.9.


CSR µEnergy Profile Demonstrator

Main | Advanced | Help

Device Connections: Discover devices | Disconnect

Connection Parameters: Minimum 10 | Maximum 10 | Latency 64 | Timeout 800 | Update parameters | [✓] Quick connect | [ ] Accept connection updates | [ ] Random address

Bluetooth Smart Devices
- Paired devices
  - No paired devices
- Connected devices
  - CSR Keyboard
- Discovered devices
  - XOLO A800
  - CSR - bc7
  - 00:02:5B:03:65:A0

HID device | Device Information | Battery

Reports

```
1 INPUT - 00 00 4C 00 00 00 00 00
```

Descriptor

```
05 01 09 06 A1 01 85 01 05 07 19 E0 29 E7 15 00 25 01
75 01 95 08 81 02 95 01 75 08 81 03 95 05 75 01 05 08
19 01 29 05 91 02 95 01 75 03 91 03 95 06 75 08 15 00
25 65 05 07 19 00 29 65 81 00 C0 06 00 FF 09 02 A1 01
85 02 75 08 95 01 15 01 25 64 09 20 81 00 C0 05 0C 09
```

Information

Parser version	2.1.3
Device subclass	40
Country code	01
Reconnect initiate	Yes
Remote wake	No
Boot device	Yes

Control: Suspend | Send

Protocol mode: Boot [✓ Report] | Set | Get

Properties

General	
Address	00:02:5B:00:15:10
Name	CSR Keyboard
RSSI	-44dBm
TX Level	-2dBm
Flags	Yes, No, Yes, No, ...
Connection i...	(Unknown), (Unk...
Device services	
Generic Access Profile	
Generic Attribute Profile	
Device Information Service	
Battery Service	
HID Service	
UUID	1812
Start handle	0009
End handle	0023
Characteristics	
Service 0x1813	
UUID	1813
Start handle	0029
End handle	002E

HID Service
This service exposes the HID reports and other HID data intended for HID Hosts and HID Devices

Ready

Figure 2.9: Device Connected

4. The HID device tab in **CSR µEnergy Profile Demonstrator** application window shows five sections:

   4.1. Reports

   This section shows the latest HID Input report received from the Keyboard application.

   4.2. Descriptor

   This section shows the HID Descriptor used by the Keyboard application.

   4.3. Information

   This section shows the HID Information supported by the Keyboard application.

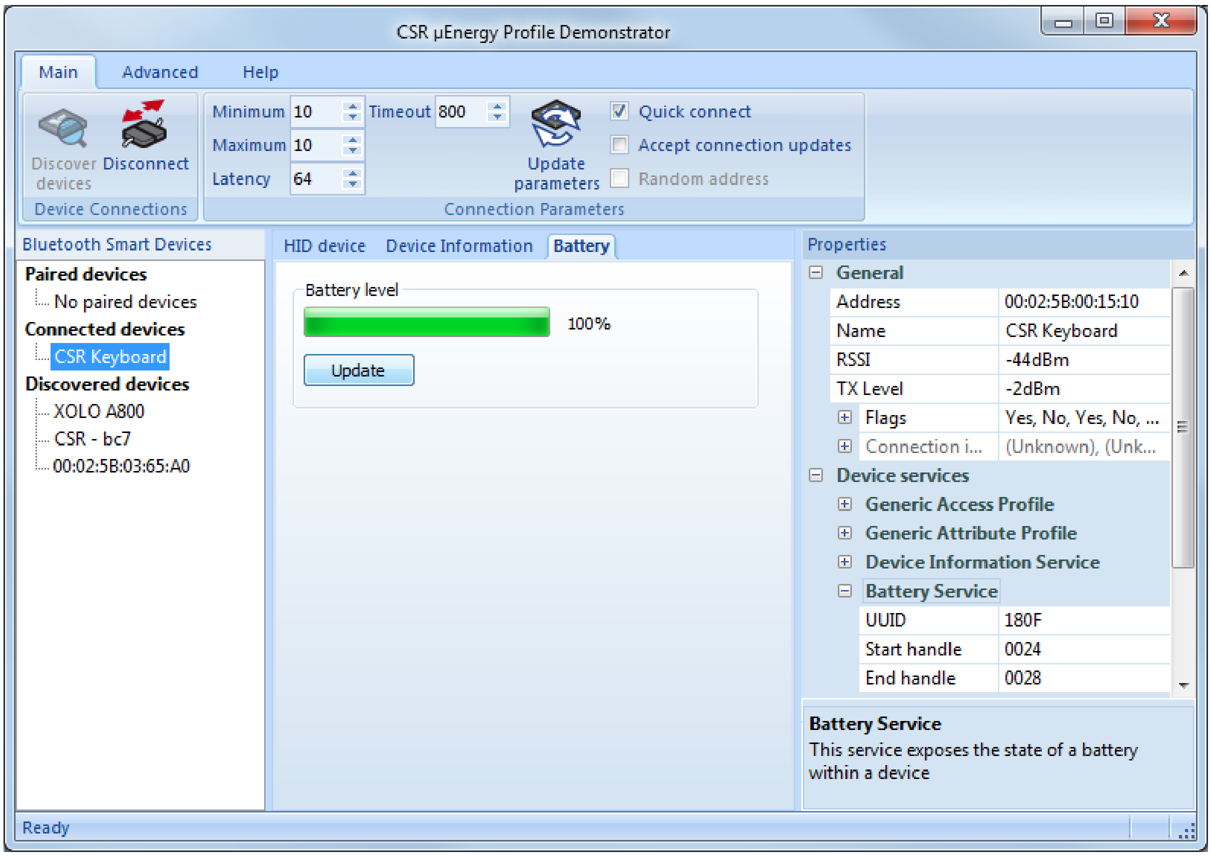
   4.4. Control

   This section is used to trigger Control Point operations on the connected Keyboard application by selecting the required operation from the drop down box. See *HID Service Specification Version 1.0* for more information on HID Control Point operations.

4.5. Protocol Mode

This section is used to change the protocol mode of the connected Keyboard application by selecting the appropriate radio button. See *HID Service Specification Version 1.0* for more information on HID Protocol modes.

5. The **Battery** tab shown in Figure 2.10 displays the current state of battery and the **Device Information** tab shown in Figure 2.11 displays the Device information characteristics of the application.



Minimum 10
Maximum 10
Latency 64
Timeout 800
[✓] Quick connect
[ ] Accept connection updates
[ ] Random address

Paired devices
- No paired devices

Connected devices
- CSR Keyboard

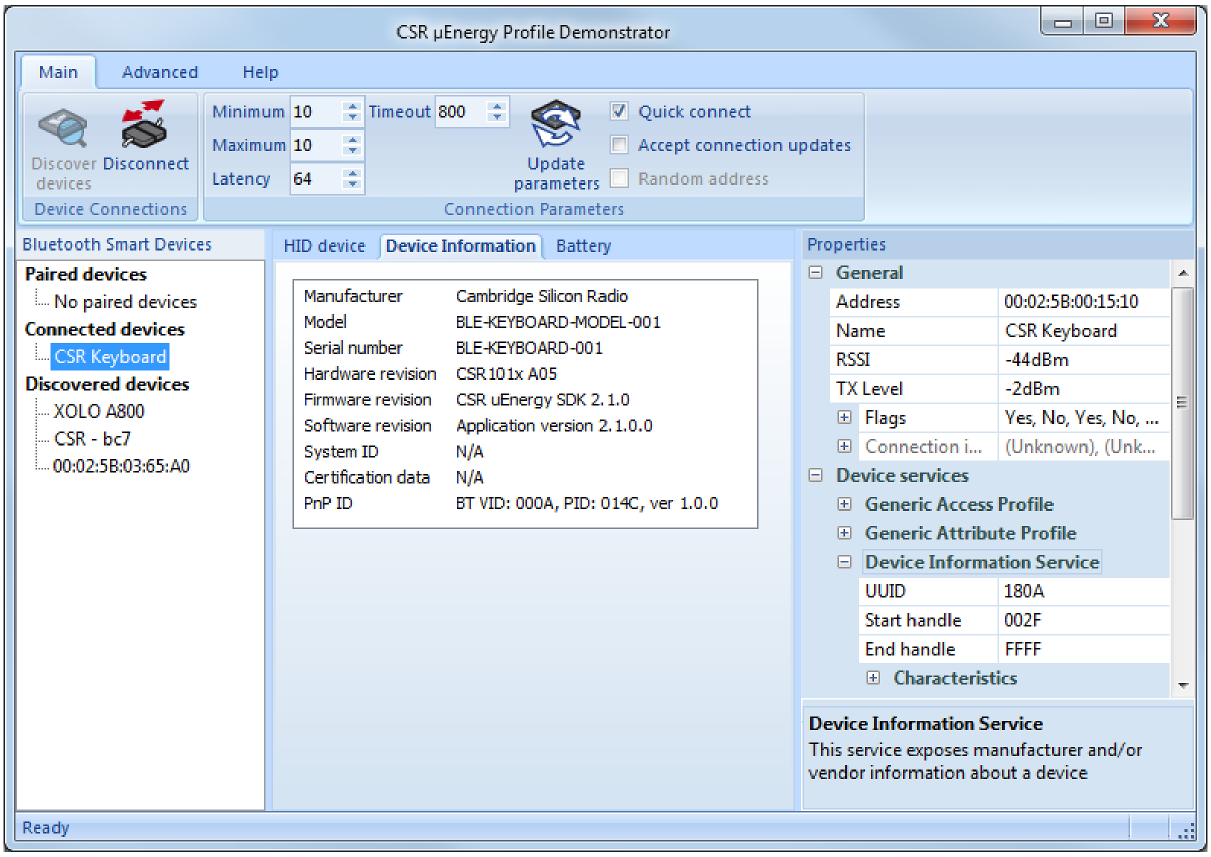
Discovered devices
- XOLO A800
- CSR - bc7
- 00:02:5B:03:65:A0

Battery level
100%

General	
Address	00:02:5B:00:15:10
Name	CSR Keyboard
RSSI	-44dBm
TX Level	-2dBm
Flags	Yes, No, Yes, No, ...
Connection i...	(Unknown), (Unk...
Device services	
Generic Access Profile	
Generic Attribute Profile	
Device Information Service	
Battery Service	
UUID	180F
Start handle	0024
End handle	0028

Battery Service
This service exposes the state of a battery within a device

Figure 2.10: Battery Level



Minimum 10 Timeout 800
Maximum 10
Latency 64
[✓] Quick connect
[ ] Accept connection updates
[ ] Random address

Paired devices
- No paired devices

Connected devices
- CSR Keyboard

Discovered devices
- XOLO A800
- CSR - bc7
- 00:02:5B:03:65:A0

Manufacturer	Cambridge Silicon Radio
Model	BLE-KEYBOARD-MODEL-001
Serial number	BLE-KEYBOARD-001
Hardware revision	CSR101x A05
Firmware revision	CSR uEnergy SDK 2.1.0
Software revision	Application version 2.1.0.0
System ID	N/A
Certification data	N/A
PnP ID	BT VID: 000A, PID: 014C, ver 1.0.0

General

Address	00:02:5B:00:15:10
Name	CSR Keyboard
RSSI	-44dBm
TX Level	-2dBm
Flags	Yes, No, Yes, No, ...
Connection i...	(Unknown), (Unk...

Device services
- Generic Access Profile
- Generic Attribute Profile
- Device Information Service

UUID	180A
Start handle	002F
End handle	FFFF

- Characteristics

Device Information Service
This service exposes manufacturer and/or vendor information about a device

Figure 2.11: Device Information Service Information

6. Click the **Disconnect** button to disconnect the Bluetooth Smart link between the device and the USB dongle shown in Figure 2.11.

3. Application Structure

3.1. Source Files

Table 3.1 lists the source files and their purpose.

File name	Purpose
`keyboard.c`	Implements all the entry functions e.g. `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Events received from the hardware and firmware are first handled here. This file contains handling functions for all the LM and system events.
`keyboard_gatt.c`	Implements routines for triggering advertisement procedures.
`keyboard_hw.c`	Implements routines for hardware initialisation, key press, pairing button press handling and glowing LEDs when output reports are written.
`hid_service.c`	Implements routines required for HID service e.g. Input report notifications corresponding to key press/release data to the remote device, and handling read/write access indications on the HID service specific ATT attributes.
`hid_boot_service.c`	Implements routines required for sending Input reports corresponding to key press/release data using CSR proprietary HID service.
`scan_param_service.c`	Implements routines required for storing the scan interval and scan window used by the HID host.
`battery_service.c`	Implements routines required for the Battery service e.g. reading Battery Level, notifying it to the remote device and handling access indications on the Battery service specific ATT attributes.
`gap_service.c`	Implements routines for the GAP service e.g. handling read/write access indication on the GAP service characteristics, reading/writing device name on NVM etc.
`nvm_access.c`	Implements the NVM read/write routines.

Table 3.1: Source Files

3.2. Header Files

Table 3.2 lists the header files and their purpose.

File name	Purpose
`keyboard.h`	Contains application data structure definition and prototypes of externally referred functions defined in `keyboard.c` file.
`keyboard_gatt.h`	Contains timeout values for fast/slow advertising and prototypes of externally referred functions defined in `keyboard_gatt.c` file.
`keyboard_hw.h`	Contains prototypes of externally referred hardware routines of `keyboard_hw.c` file.

File name	Purpose
`app_gatt.h`	Contains macro definitions, user defined data type definitions and function prototypes which are being used across the application.
`appearance.h`	Contains the appearance value macro of the Keyboard application.
`battery_service.h`	Contains prototypes of externally referred functions defined in `battery_service.c` file.
`battery_uuids.h`	Contains macro definitions for UUIDs of the Battery service and related characteristics.
`dev_info_uuids.h`	Contains macros for UUID values of the Device Information service.
`gap_conn_params.h`	Contains macro definitions for fast/slow advertising, preferred connection parameters, maximum number of connection parameter update requests etc.
`gap_service.h`	Contains prototypes of the externally referred functions defined in `gap_service.c` file.
`gap_uuids.h`	Contains macro definitions for UUIDs of the GAP service and related characteristics.
`gatt_serv_uuid.h`	Contains macros for UUID values of the GATT service.
`hid_service.h`	Contains macro values for the values that can be taken by specific characteristics of HID service and prototypes of externally referred functions defined in `hid_service.c` file.
`hid_uuids.h`	Contains macro definitions for UUIDs of the HID service and related characteristics.
`hid_boot_service.h`	Contains prototypes of externally referred functions defined in `hid_boot_service.c` file.
`hid_boot_service_uuids.h`	Contains macro definitions for UUIDs of the proprietary HID service and related characteristics.
`nvm_access.h`	Contains prototypes of externally referred NVM read/write functions defined in `nvm_access.c` file.
`scan_param_service.h`	Contains macro values for the values that can be taken by specific characteristics of scan parameters service and prototypes of externally referred functions defined in `scan_param_service.c` file.
`scan_parameters_uuids.h`	Contains macro definitions for UUIDs of the scan parameters service and related characteristics.

Table 3.2: Header Files

3.3. Database Files

The SDK uses database files to generate attribute database for the application. For more information on how to write database files, see the *GATT Database Generator User Guide*.

Table 3.3 lists the database files and their purpose.

File name	Purpose
app_gatt_db.db	Master database file which includes all service specific database files. This file is imported by the GATT Database Generator.
battery_service_db.db	Contains information related to Battery service characteristics, their descriptors and values. See Table B.1 for more information on Battery service characteristics.
dev_info_service_db.db	Contains information related to Device Information service characteristics, their descriptors and values. See Table B.2 for Device Information service characteristics.
gap_service_db.db	Contains information related to GAP service characteristics, their descriptors and values. See Table B.3 for GAP characteristics.
gatt_service_db.db	Contains information related to GATT service characteristics, their descriptors and values.
hid_boot_service_db.db	Contains information related to CSR proprietary HID service characteristics, their descriptors and values. The support for CSR proprietary HID service is optional. See section 7.8 for details.
hid_service_db.db	Contains information related to HID service characteristics, their descriptors and values. See Table B.4 for HID service characteristics.
scan_parameters_db.db	Contains information related to Scan Parameters service characteristics, their descriptors and values. See Table B.5 for Scan Parameters service characteristics.

Table 3.3: Database Files

3.4. Assembler code

3.4.1. Pio_ctrlr_code.asm

The PIO controller code implements the keyboard matrix scanning and generates a PIO controller event at the end of each scan cycle when at least one key press has been detected. See section 5 for an overview on the keyboard matrix.

Figure 3.1 shows the scanning algorithm used by the PIO controller.

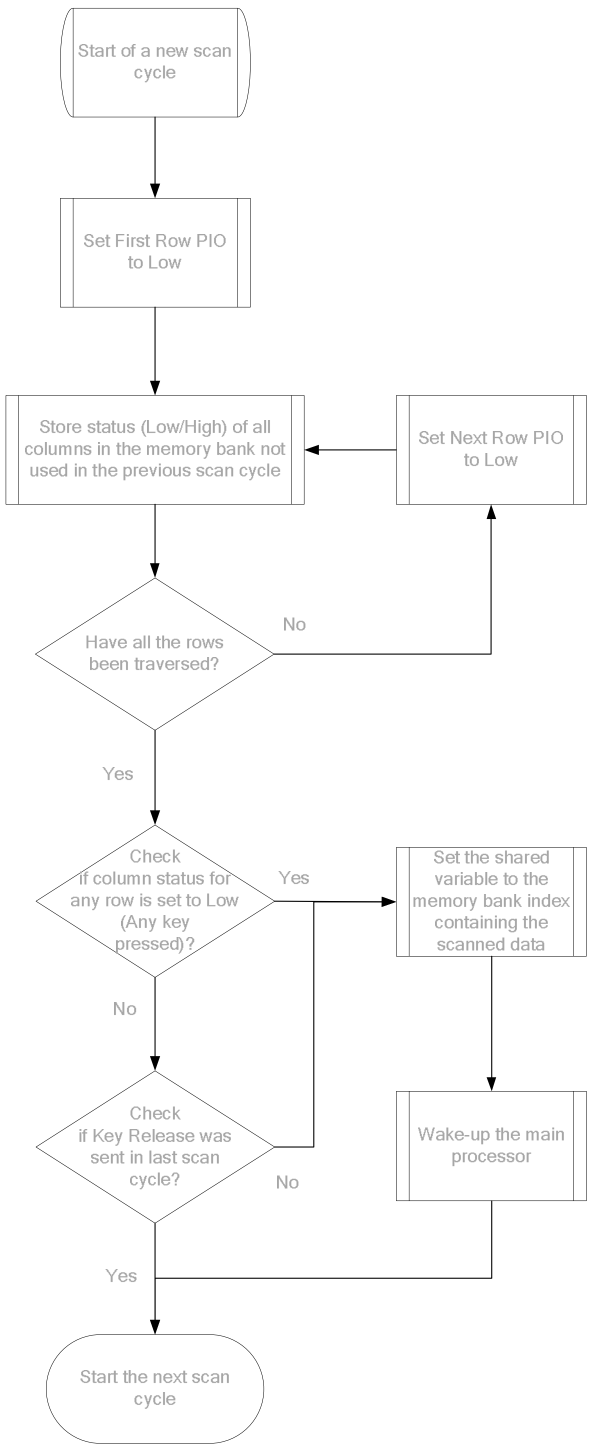

Start of a new scan cycle
Set First Row PIO to Low
Store status (Low/High) of all columns in the memory bank not used in the previous scan cycle
Set Next Row PIO to Low
Have all the rows been traversed?
No
Yes
Check if column status for any row is set to Low (Any key pressed)?
Yes
Set the shared variable to the memory bank index containing the scanned data
No
Check if Key Release was sent in last scan cycle?
No
Wake-up the main processor
Yes
Start the next scan cycle

Figure 3.1: Scanning Algorithm Used by the PIO Controller

All PIOs corresponding to rows and columns of the keyboard are pulled high by default. Key presses are detected by setting a single row to Low and checking the status of each of the columns, repeating for each row in turn. The column result for each row is stored in memory buffers accessible to both the main processor and the PIO controller.

A key press results in a row PIO being shorted with a column PIO, causing the column PIO to be low when the row PIO is driven low by the scanning algorithm. At the end of the scan cycle, the PIO controller checks the results buffers and wakes the main processor if a key press is detected, i.e. at least one row-column combination has been detected as low. This entire scan cycle takes approximately 5.5 ms.

If a key is held down across multiple scan cycles, the main processor is awoken for each of these scan cycles.

When a key is released, the PIO controller checks the results buffers at the end of the scan cycle to determine if ALL previously pressed keys have been released and proceeds to wake the main processor before sending the Key Release notification.

Two banks of memory buffers, defined as 0 and 1, are used alternately to avoid the application reading an incomplete buffer during a scan cycle. Each bank is 18 bytes in size which corresponds to the 18 row x 8 column keyboard matrix. If a scan cycle uses bank 0 to store the scan results, then the next scan cycle uses bank 1 to store the scan results, and vice versa.

A single memory location at address `0x40` is shared between the application and the PIO controller and indicates which buffer bank currently has valid data for reading by the application. For example a value of 0 in this location indicates that memory buffer bank 0 currently has valid data for the main processor to read.

4. Code Overview

Section 4.1 describes significant functions of the Keyboard application.

4.1. Application Entry Points

4.1.1. AppInit()

This function is invoked when the application is powered on or the chip resets. It performs the following initialisation steps:

1. Initialises the application timers, application data structures and hardware
2. Configures the GATT entity for the server role and installs support for optional GATT procedures such as write long characteristic value
3. Configures the NVM manager to use the I^{2}C EEPROM
4. Initialises all the services
5. Reads the persistent store
6. Registers the ATT database with the firmware

4.1.2. AppProcessLmEvent()

This function is invoked whenever a LM-specific event is received by the system. Sections 4.1.2.1 to 4.1.2.4 describe the events handled in this function:

4.1.2.1. Database Access

- `GATT_ADD_DB_CFM`: This confirmation event marks the completion of the database registration with the firmware. On receiving this event, the Keyboard application starts advertising.
- `GATT_ACCESS_IND`: This indication event is received when the remote HID host tries to access an ATT characteristic managed by the application.

4.1.2.2. LS Events

- `LS_CONNECTION_PARAM_UPDATE_CFM`: This confirmation event is received in response to the connection parameter update request by the application. The connection parameter update request from the application triggers the L2CAP connection parameter update signalling procedure. See Volume 3, Part A, Section 4.20 of Bluetooth Core Specification Version 4.0.
- `LS_CONNECTION_PARAM_UPDATE_IND`: This indication event is received when the remote central device updates the connection parameters. On receiving this event, the application validates the new connection parameters against the preferred connection parameters and triggers a connection parameter update request if the new connection parameters do not comply with the preferred connection parameters.
- `LS_RADIO_EVENT_IND`: The Keyboard application registers for this indication when it finds that firmware has no buffers to accept key press or release data from the application. The application uses the `radio_event_tx_data` option to receive the `LS_RADIO_EVENT_IND` event from firmware when data is transmitted successfully to the connected remote device, freeing up buffers in the firmware. The Keyboard application transmits buffered key press or release data to the connected host on receiving this event.

4.1.2.3. SMP Events

- `SM_PASSKEY_INPUT_IND`: This indication event is received when the application uses Security Mode 1 Level 3 (i.e. MITM) for pairing and the remote side has displayed a number to be keyed in from the Keyboard application.
- `SM_KEYS_IND`: This indication event is received on completion of the bonding procedure. It contains keys and security information used on a connection that has completed the short term key generation. The application stores the received diversifier (DIV) and Identity Resolving Key (IRK) (if collector device supports resolvable random addresses) to NVM. See Volume 3, Part H, section 2.1 of the *Bluetooth Core Specification Version 4.0*.
- `SM_SIMPLE_PAIRING_COMPLETE_IND`: This indication event indicates that the pairing has completed successfully or otherwise. See Volume 3, Part H, Section 2.3 of *Bluetooth Core Specification Version 4.0*. In the case of a successful completion of the pairing procedure, the Keyboard application is bonded with a HID host and the bonding information is stored in the NVM. The bonded device address will be added to the white list, if it is not a resolvable random address.
- `SM_DIV_APPROVE_IND`: This indication event is received when the remote connected device re-encrypts the link or triggers encryption at the time of reconnection. The firmware sends the diversifier in this event and waits for the application to approve or disapprove the encryption. The application shall disapprove the encryption if the bond has been removed by the user. The application compares this diversifier with the one it had received in `SM_KEYS_IND` at the time of the first encryption. If similar, the application approves the encryption, otherwise it disapproves it.
- `SM_PAIRING_AUTH_IND`: This indication is received when the remote connected device initiates pairing. The application can either accept or reject the pairing request from the peer device. The application shall reject the pairing request if it is already bonded to a HID host to prevent any new device disguising itself as one previously bonded to the keyboard.

4.1.2.4. Connection Events

- `GATT_CONNECT_CFM`: This confirmation event indicates that the connection procedure has completed. If it has not successfully completed, the application restarts advertising. If the application is bonded to a device with resolvable random address and connection is established, the application tries to resolve the connected device address using the IRK stored in NVM. If the application fails to resolve the address, it disconnects the link and restarts advertising.
- `GATT_CANCEL_CONNECT_CFM`: This confirmation event confirms the cancellation of the connection procedure. When the application stops advertisements to change the advertising parameters or to save power, this signal confirms the successful stopping of advertisements by the Keyboard application.
- `LM_EV_CONNECTION_COMPLETE`: This event is received when the connection with the master is considered to be complete and includes the new connection parameters.
- `LM_EV_DISCONNECT_COMPLETE`: This event is received on link disconnection. Disconnection could be due to link loss, locally triggered or triggered by the remote connected device.
- `LM_EV_ENCRYPTION_CHANGE`: This event indicates a change in the link encryption.
- `GATT_CHAR_VAL_NOT_CFM`: This confirmation event confirms that the firmware has received the application request to send a notification (e.g. for data corresponding to key press/release). The confirmation includes a status indicating whether the notification was successfully queued by the firmware to be sent to the remote device or not.
- `LM_EV_CONNECTION_UPDATE`: This event indicates that the connection parameters have been updated to a new set of values and is generated when the connection parameter update procedure is either initiated by the master or the slave. These new values are stored by the application for comparison against the preferred connection parameter, see section 7.5.

4.1.3. AppProcessSystemEvent()

This function handles the system events such as a PIO change. It currently handles the following two system events:

- `sys_event_pio_ctrlr`: This event is received when the PIO controller detects any key being pressed or ALL keys being released following an earlier key press by the Keyboard application.
- `sys_event_pio_changed`: This event indicates a change in PIO status. Whenever the user presses or releases the pairing removal button, the corresponding PIO status changes and the application receives a PIO changed event and takes the appropriate action.

4.2. Internal State Machine

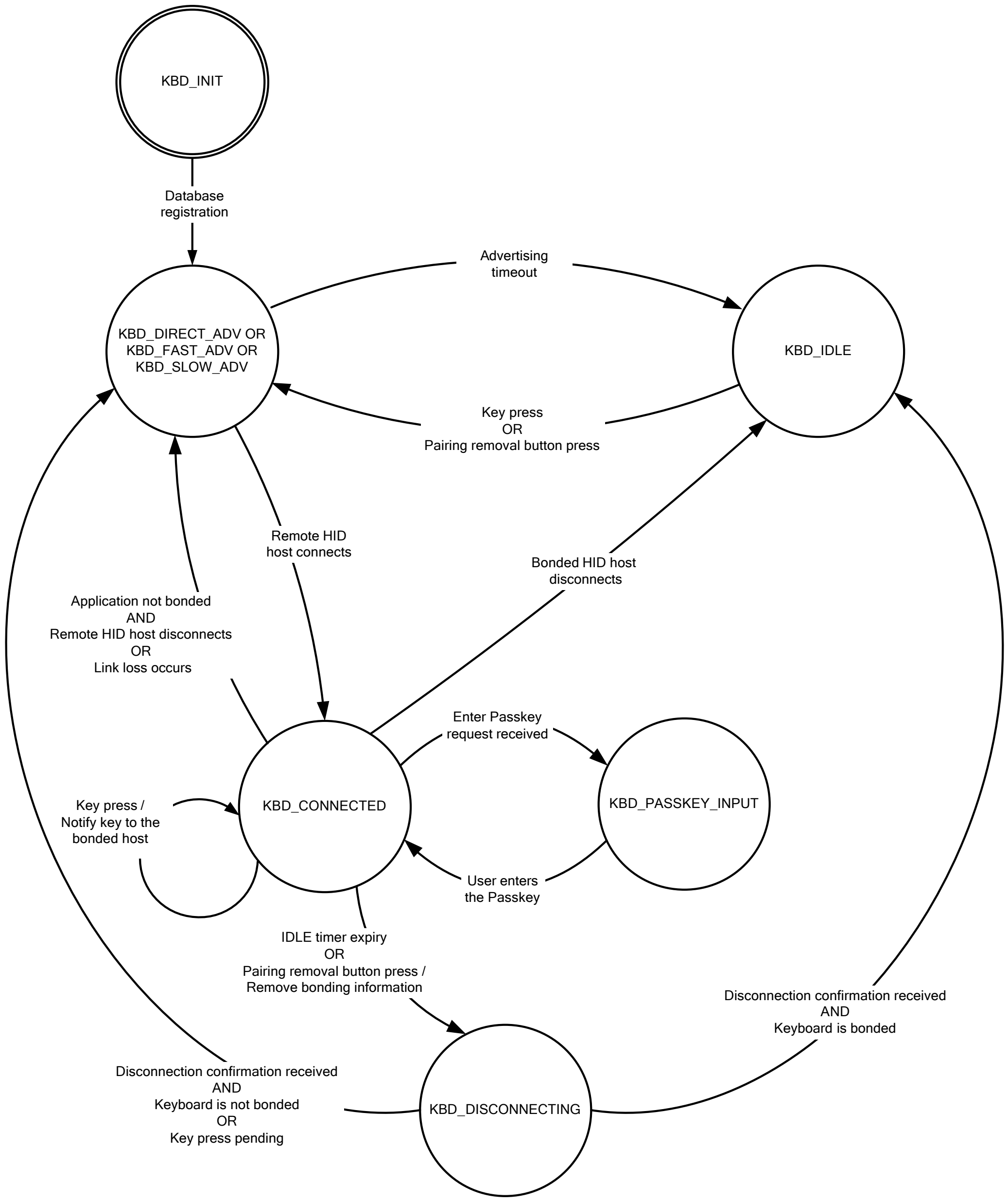

KBD_INIT
Database registration
KBD_DIRECT_ADV OR KBD_FAST_ADV OR KBD_SLOW_ADV
Advertising timeout
KBD_IDLE
Key press OR Pairing removal button press
Remote HID host connects
Bonded HID host disconnects
Application not bonded AND Remote HID host disconnects OR Link loss occurs
Enter Passkey request received
KBD_CONNECTED
KBD_PASSKEY_INPUT
Key press / Notify key to the bonded host
User enters the Passkey
IDLE timer expiry OR Pairing removal button press / Remove bonding information
Disconnection confirmation received AND Keyboard is bonded
Disconnection confirmation received AND Keyboard is not bonded OR Key press pending
KBD_DISCONNECTING

Figure 4.1: Internal State Machine Diagram

The Keyboard application has eight internal states, see Figure 4.1.

4.2.1. KBD_INIT

When the Keyboard application is powered on or the chip resets, it initialises in this state and registers the database with the firmware. When the application receives a successful confirmation for the database registration it enters either the `KBD_FAST_ADVERTISING` state or the `KBD_DIRECT_ADVERT` state.

4.2.2. KBD_IDLE

The Keyboard application is not connected to any HID host.

- On a key press, the application triggers advertisements and enters the `KBD_FAST_ADVERTISING` or `KBD_DIRECT_ADVERT` state.
- When the user performs the Pairing button press, the application removes the bonding information, clears the white list and starts advertising. See *Bluetooth Core specification Version 4.0* for more information on white list.

4.2.3. KBD_ADVERTISING

The Keyboard application enters limited discoverable mode in this state. The transitions between the sub states are depicted in Figure 2.4 and Figure 2.5.

- Sub state `KBD_DIRECT_ADVERT`: The `KBD_DIRECT_ADVERT` sub state is used only when the Keyboard application supports privacy, and triggers directed advertisements in this state. The application starts in this sub state if bonded to the HID host as given below:
    - If privacy is disabled and Keyboard application is bonded to the HID host
    - If privacy is enabled and a valid re-connection address has been written to the Reconnection address characteristic of GAP service (see Appendix B) by the bonded HID host. The application triggers directed advertisements using the re-connection address written by the bonded HID host.
    - If the bonded host re-connects, the application enters the `KBD_CONNECTED` state. If the bonded host does not connect before the directed advertisement timeout period ends, the application enters the `KBD_FAST_ADVERTISING` state.
- Sub state `KBD_FAST_ADVERTISING`: The application triggers un-directed advertisements using fast advertisement parameters in this sub state. The application starts in this sub state if
    - Privacy is enabled and a valid re-connection address has not been written by the bonded device.
    - The Keyboard application is not bonded to any HID host. In this case, the advertisements are accompanied with a blinking LED, indicating to the user that the keyboard is now connectable to any HID host.
    - The application enters the `KBD_CONNECTED` state if a remote HID host connects to the application. The application enters the `KBD_SLOW_ADVERTISING` sub state if the fast advertising timer expires. See section 7.2 for more information on advertisement timers.
- Sub state `KBD_SLOW_ADVERTISING`: The application triggers un-directed advertisements using slow advertisement parameters in this sub state. The application enters the `KBD_CONNECTED` state if a remote HID host connects to the HID device and enters the `KBD_IDLE` state if the slow advertising timer expires. On a key press, the application enters the `KBD_DIRECT_ADVERT` or the `KBD_FAST_ADVERTISING` sub states.

See *Bluetooth Core Specification v4.0* for more information on limited discoverable mode.

4.2.4. KBD_PASSKEY_INPUT

The Keyboard application enters this state when it receives a `SM_PASSKEY_INPUT_IND` event from the firmware when in the `KBD_CONNECTED` state. The `SM_PASSKEY_INPUT_IND` event is received when the connected HID host initiates pairing using Security Mode 1 Level 3 (i.e. MITM) and sends a 'Pairing Confirm' command to the HID Device while displaying a number on its user interface. The application uses the numbers entered by the user to send the passkey to the firmware and enters the `KBD_CONNECTED` state. The passkey is used to calculate the confirm value to be sent to the connected HID host.

4.2.5. KBD_CONNECTED

The keyboard is connected to the remote HID host.

- On a key press, the application forms the Input report corresponding to the pressed key and sends it to the bonded HID host.
- On the Pairing button being pressed, the application disconnects the link, removes the bonding information, clears the white list and starts advertising. See *Bluetooth Core specification Version 4.0* for more information on white lists.
- The application disconnects the link if kept idle for some time and enters the `KBD_DISCONNECTING` state. See section 7.4 for more information on the idle timer.

In the case of a link loss, the application enters the `KBD_FAST_ADVERTISING` or `KBD_DIRECT_ADVERT` states.

- In the case of a remote triggered disconnection and if the application is not bonded to any HID host, the application enters the `KBD_FAST_ADVERTISING` state; otherwise it enters the `KBD_IDLE` state.

4.2.6. KBD_DISCONNECTING

The Keyboard application enters this state after initiating the disconnect procedure.

- When it receives the disconnect confirmation, it checks if it is bonded to any HID host.
  - If the application is bonded, it enters the `KBD_IDLE` state and waits for user activity.
  - If the application is not bonded, it enters the `KBD_FAST_ADVERTISING` state.
- On the **Pairing** button being pressed, the application removes the bonding information and clears the white list.

5. Processing of Key Presses

A typical keyboard consists of a matrix of electrical contacts organised into M rows and N columns. When a user presses a key on the keyboard, the electrical contact shorts a row with a column. Detecting which rows and columns have been shorted determines which keys are currently pressed. Figure 5.1 represents a keyboard matrix; when a user presses the key 'A' on the keyboard, row 0 and column 0 are shorted together. Similarly when the key 'D' is pressed, row 1 and column 1 are shorted together.

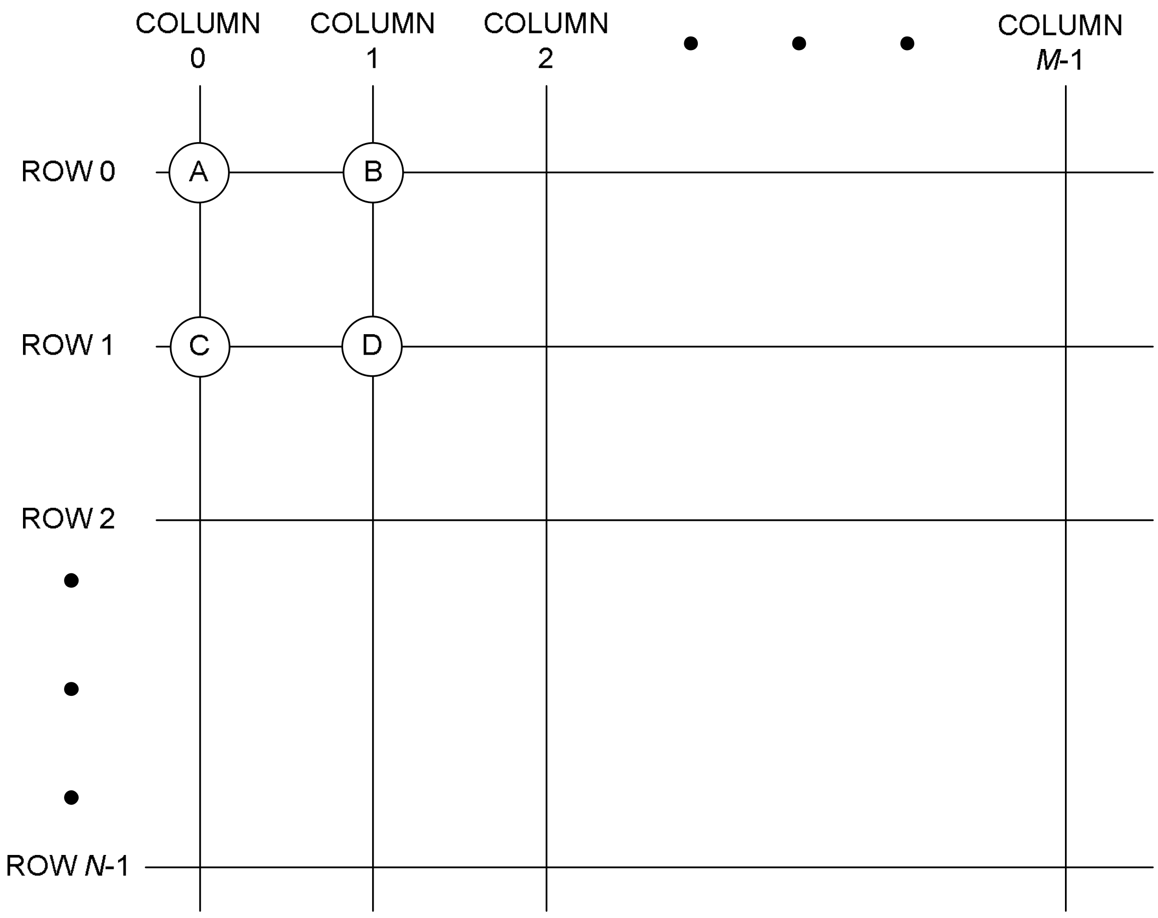


Figure 5.1: Diagrammatic Representation of a Keyboard Matrix

The CSR10x1 serves as the Bluetooth chip on which the PIOs are mapped to the row and column contacts on a keyboard. When a user presses a key on the keyboard, two PIOs corresponding to the row and column are shorted together on the development board. The PIO controller within the chip detects the key press and wakes up the main processor as described in section 3.4.1. The application receives a `sys_event_pio_ctrlr` event that is handled by the function `AppProcessSystemEvent()`. HID reports are formed from the PIO controller data in the following steps:

1. The data scanned by the PIO controller is copied to a local variable as the PIO controller will erase the data in its next scan cycle.
2. The scanned data is checked for ghost keys as described in section 5.1.

3. The scanned data is de-bounced as described in section 5.2.
4. The de-bounced data is checked for ghost keys.
5. A raw report is generated using the scanned data as described in section 5.3.
6. The generated raw report is compared against the previously generated raw report and if different, Input and Consumer reports are created and added to a queue. If the application is in `KBD_PASSKEY_INPUT` state, then the raw report is used to enter the passkey.
7. The Input and Consumer reports are added to a queue if they are different from the previously generated reports of these specific report types.
8. The Input and Consumer reports are transmitted to the connected HID host if encryption is enabled and notifications are enabled for Input reports.

5.1. Ghost Key Filtering

Key ghosting occurs when three keys that form corners of a rectangle are pressed by the user but a fourth key appears to be additionally pressed. This fourth key is termed the ghost key and is caused by multiple shorted paths.

As an example, if the keys 'A', 'B', and 'C' are pressed simultaneously by the user on the keyboard as depicted in Figure 5.2, the row 0 scan results in 'A' and 'B' being detected, and the row 1 scan results in both 'C' and the ghost key 'D' being detected due to the second path being Low.

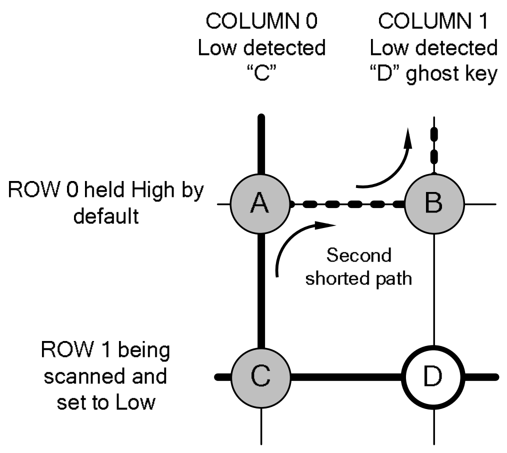


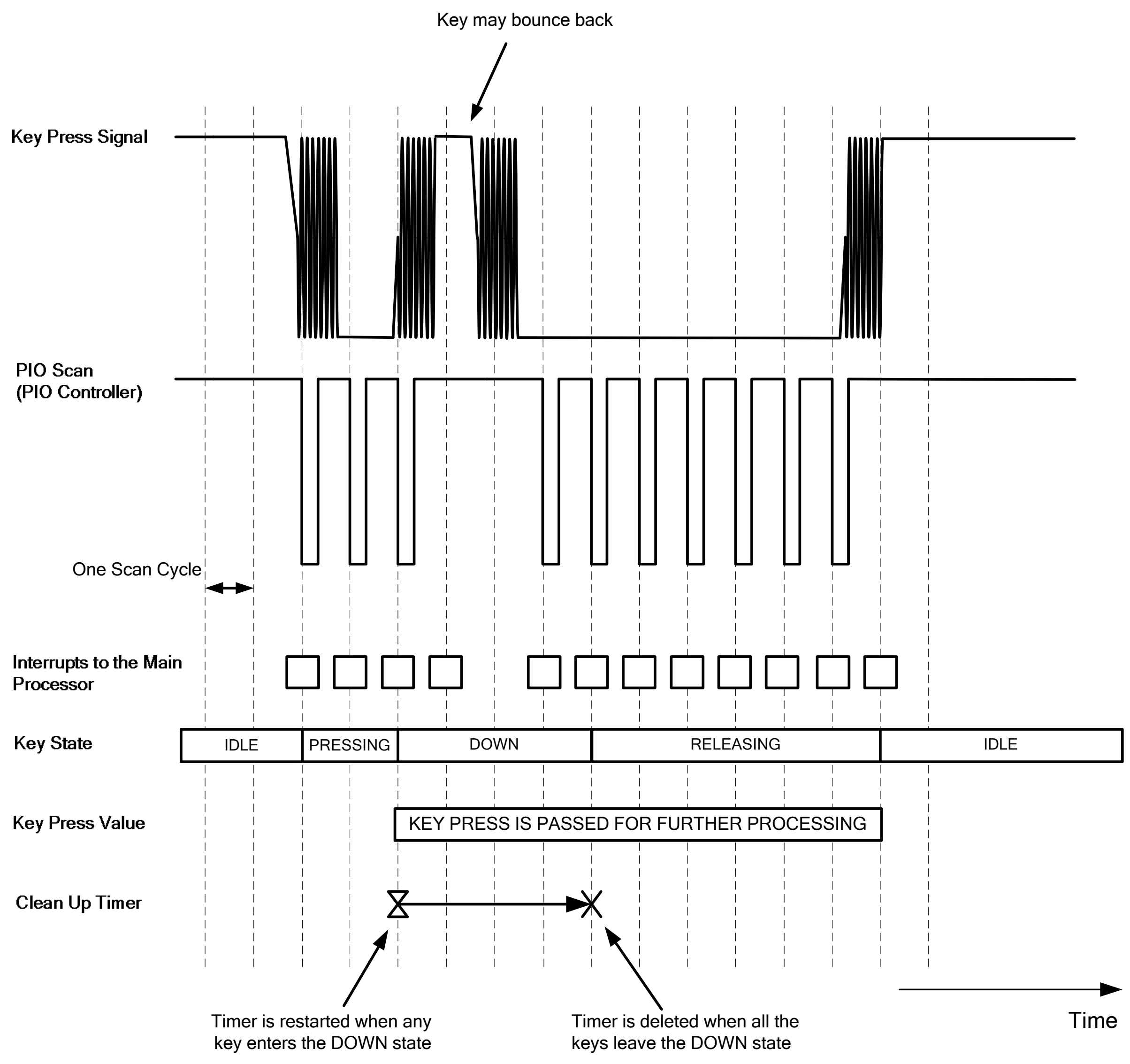
Figure 5.2: Ghost Key Detected on Row 1 Scan

It has been observed that in a few scan cycles, the PIO controller detects the keys 'A', 'B' and 'D' being pressed even though keys 'A', 'B', and 'C' were only pressed by the user.

Ghost keys are filtered out by bit-wise ANDing the detected row result with all other rows whenever two key presses are detected in a particular row. If this bit-wise ANDing operation generates a non-zero result, the resulting pattern is due to ghosting and therefore no key presses are sent to the connected HID host.

5.2. De-bouncing of Keys

The physical action of the electrical contacts making and breaking gives rise to a bouncing input pattern. The process for de-bouncing the keys of the keyboard is shown in Figure 5.3.



Key may bounce back
Key Press Signal
PIO Scan
(PIO Controller)
One Scan Cycle
Interrupts to the Main Processor
Key State
IDLE
PRESSING
DOWN
RELEASING
IDLE
Key Press Value
KEY PRESS IS PASSED FOR FURTHER PROCESSING
Clean Up Timer
Timer is restarted when any key enters the DOWN state
Timer is deleted when all the keys leave the DOWN state
Time

Figure 5.3: Bouncing Pattern Observed Upon a Key Press

A key press resulting from the 'Key Press Signal' is registered only upon receiving the 'Key Press Detected' event over three consecutive scan cycles. The state of each key and the number of times the 'Key Press Detected' event was received for each key ('count' in the state diagram shown in Figure 5.4) is maintained in an array which is updated on receiving interrupts from the PIO controller. Each key press goes through multiple states as illustrated in Figure 5.4.

- A key by default is in the `IDLE` state. On detecting a key press, the key moves to the `PRESSING` state.

- Upon detecting the key pressed event for the key for two more scan cycles, the key moves to the DOWN state.

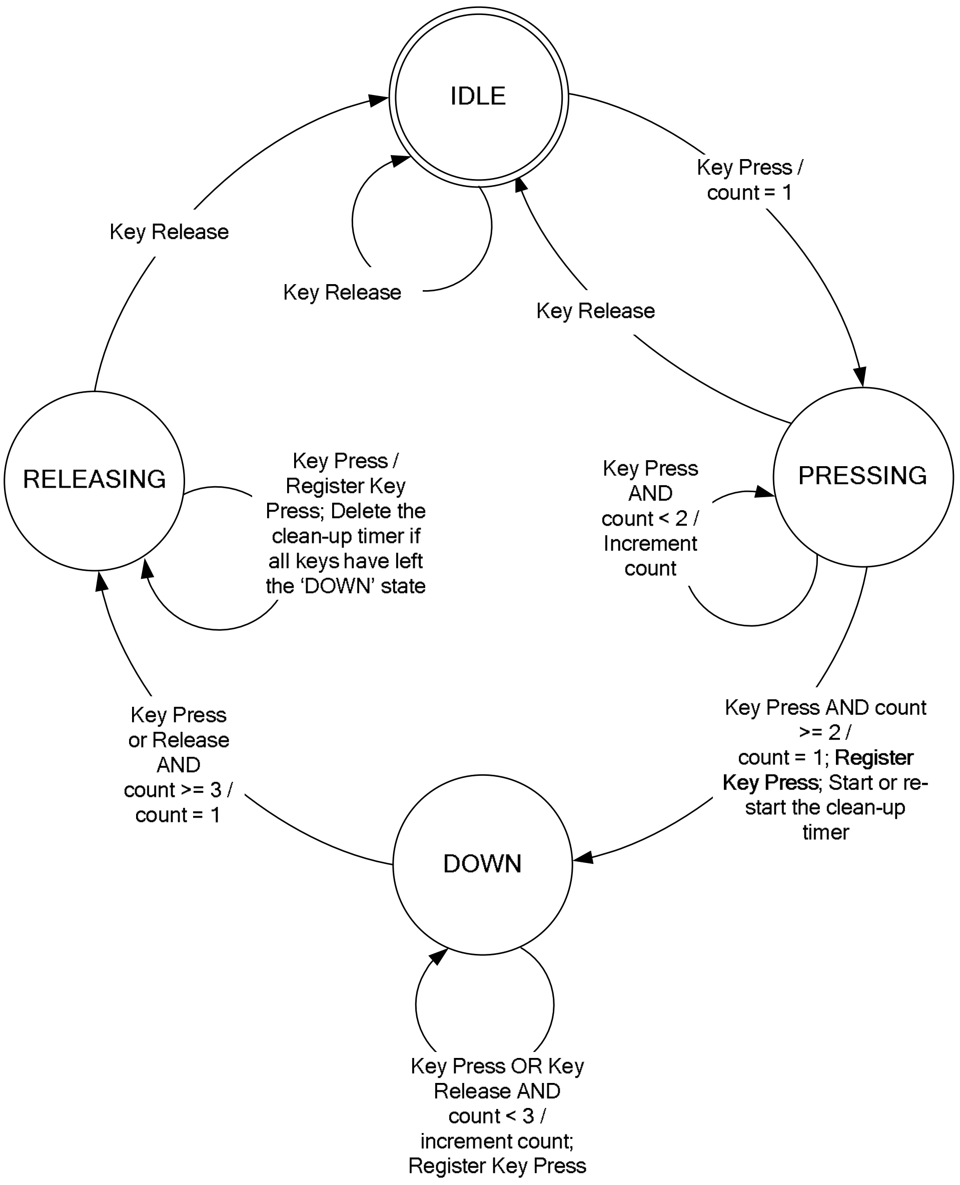

IDLE
Key Press / count = 1
PRESSING
Key Release
Key Release
Key Release
RELEASING
Key Press / Register Key Press; Delete the clean-up timer if all keys have left the 'DOWN' state
Key Press AND count < 2 / Increment count
Key Press AND count >= 2 / count = 1; Register Key Press; Start or re-start the clean-up timer
Key Press or Release AND count >= 3 / count = 1
DOWN
Key Press OR Key Release AND count < 3 / increment count; Register Key Press

Figure 5.4: State Flow for De-bouncing

- A transition to the DOWN state means that the bouncing has been filtered out and the de-bouncing algorithm registers the key to be pressed continuously for the next three scan cycles. While in this state, any additional bouncing that may occur is filtered out.
- It is possible that a valid key release event may be lost by the de-bouncing algorithm during these three scan cycles in this state. To prevent this key release being lost, a clean-up timer is started when first entering the DOWN state. Upon expiry of this timer, the key state returns to the IDLE state.

- After receiving three key press/release events in the `DOWN` state, the key moves to `RELEASING` state.
- If the de-bouncing algorithm receives another key press/release event after moving to the `RELEASING` state, it means that a valid key release event has been detected and the clean-up timer is deleted.
- If a key press event is received in the `RELEASING` state, it means that a key is being held down. Upon receiving an event indicating that the key has been released, the state of the key is updated from `RELEASING` to `IDLE`.

On moving to the `DOWN` state, a HID report corresponding to the key being pressed is generated and sent to the connected HID host as described in the following sections.

5.3. Generation of Input Report from Scanned Data

The scanned data is contained in an array of 18 `uint8` elements, with each element corresponding to a particular row in the keyboard. Each `uint8` element is a snapshot of all column PIO statuses when the row PIO is made low. For the first element of the array (corresponding to the column snapshot when row 0 is shorted), if a bit in position n is set, it means that a key of row 0 and column n has been detected as being pressed.

The report corresponding to this key is fetched from the array `KEYBOARDKEYMATRIX[]` and added to the `raw_report` array. This procedure is followed for all the bits that are set in the scanned data to form an Input report, see Report ID 1 in Appendix C.

5.4. Generation of Consumer Key Report from Input Report

5.4.1. Overlaid Keys

In this implementation of a keyboard, the consumer keys are implemented as overlaid keys. An example of an overlaid key is shown in Figure 5.5.

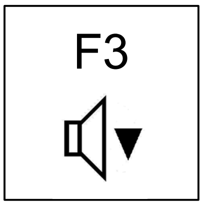


Figure 5.5: An Example of an Overlaid Key

When this example overlaid key is pressed, a HID report containing the standard HID usage ID of the **F3** key is sent to the connected HID host. When the same key is pressed with another key (normally designated as **Fn** on keyboards), a HID report containing the standard HID usage ID of 'Volume down' is sent. Hence, different reports are sent when keys in the same row and column positions are pressed.

The array `KEYBOARDKEYMATRIX[]` holds the reports (usage IDs) of the keys which should be sent when keys are pressed independently. The array `HIDLUT[]` holds the reports (usage IDs) of the keys which should be sent when the keys are pressed along with the **Fn** key.

5.4.2. Consumer Keys

The consumer keys are implemented as overlaid keys in this Keyboard application. For these keys, a set of reserved values of usage IDs in the range `0xf1` to `0xf5` are selected from the standard usage table specified by the *USB HID Specification* and placed in the HIDLUT array at indices corresponding to the consumer key positions.

To detect consumer keys, the raw_report array is parsed for reserved usage IDs, which are used to form the consumer report (Report ID 3 in Appendix C) by referring to the CONSUMERKEYS array.

6. NVM Memory Map

The applications can store data in the NVM to prevent data loss in the event of a power off or chip reset. The Keyboard application uses the memory maps described in Table 6.1 to Table 6.5 for NVM.

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
Sanity Word	uint16	1	0
Bonded Flag	Boolean	1	1
Bonded Device Address	Structure	5	2
Diversifier	uint16	1	7
IRK	uint16 array	8	8

Table 6.1: NVM Memory Map for Application

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
GAP Device Name Length	uint16	1	16
GAP Device Name	uint8 array	20	17
GAP Peripheral Privacy Flag[1]	Boolean	1	37[2]
GAP Reconnection address[1]	Structure	4	38[2]
[1] The NVM value of these characteristics will only be added if privacy is enabled. [2] The NVM Offset values used when privacy is enabled.			

Table 6.2: NVM Memory Map for GAP Service

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
Boot keyboard input report Characteristic Client Configuration Descriptor	uint16	1	37/42[1]
Keyboard input report Characteristic Client Configuration Descriptor	uint16	1	38/43[1]
Consumer report Characteristic Client Configuration Descriptor	uint16	1	39/44[1]
[1] The NVM Offset values used when privacy is enabled.			

Table 6.3: NVM Memory Map for HID Service

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
Battery Level characteristic Client Configuration Descriptor	uint16	1	40/45[1]
[1] The NVM Offset values used when privacy is enabled.			

Table 6.4: NVM Memory Map for Battery Service

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
Scan refresh characteristic Client Configuration Descriptor	uint16	1	41/46[1]
[1] The NVM Offset values used when privacy is enabled.			

Table 6.5: NVM Memory Map for Scan Parameters Service

Note:

The Keyboard application does not pack the data before writing it to the NVM. This means that writing a `uint8` value takes one word of NVM memory.

7. Customising the Application

The developer can easily customise the application by modifying the parameter values described in sections 7.1 to 7.8.

7.1. Advertising Parameters

The Keyboard application uses the parameters in Table 7.1 for fast and slow advertisements. The macros for these values are defined in the `gap_conn_params.h` file These values have been chosen by considering the overall current consumption of the device. See *Bluetooth Core Specification Version 4.0* for the advertising parameter range.

Parameter Name	Slow Advertisements	Fast Advertisements
Minimum Advertising Interval	1280 ms	60 ms
Maximum Advertising Interval	1280 ms	60 ms

Table 7.1: Advertising Parameters

7.2. Advertisement Timers

The Keyboard application enters the appropriate state on expiry of the advertisement timers. See section 4.2 for more information. The macros for these timer values are defined in the `gap_conn_params.h` file.

Timer Name	Timer Values
Fast Advertisement Timer Value	30 s
Slow Advertisement Timer Value	30 s

Table 7.2: Advertisement Timers

7.3. Connection Parameters

The Keyboard application uses the connection parameters listed in Table 7.3. The macros for these values are defined in the `gap_conn_params.h` file These values have been chosen by considering the overall current consumption of the device. See *Bluetooth Core Specification Version 4.0* for the connection parameter range. See section 7.5 for the connection update procedure.

Parameter Name	Parameter Value
Minimum Connection Interval	12.5 ms
Maximum Connection Interval	12.5 ms
Slave Latency	100 intervals
Supervision Timeout	12.5 s

Table 7.3: Connection Parameters

7.4. Idle Connected Timeout

The Keyboard application disconnects the link if no key is pressed for some time. The macro for this idle connected timeout is defined in the `gap_conn_params.h` file.

Parameter Name	Parameter Value
Idle Connected Timeout	30 min

Table 7.4: Idle Connected Timeout

7.5. Connection Parameter Update

The Keyboard application can request the remote HID host to update the connection parameters according to its power requirements. This application currently requests a connection parameter update 30 seconds after an encryption change or 30 seconds after the remote HID host changes the connection parameters. The connection parameter update is initiated by the Keyboard application when the new values are outside the range of the preferred connection parameters. The remote HID host may or may not accept the requested parameters. If the remote HID host rejects the new requested parameters, the application again requests an update after 30 seconds. The macro `GAP_CONN_PARAM_TIMEOUT` for this time value is defined in the `keyboard.c` file and should be greater than 30 seconds due to the restrictions imposed by the core specification. See *Bluetooth Core Specification Version 4.0 [Vol. 3], Part C Section 9.3.9*.

7.6. Device Name

The user can change the device name for the application. By default, it is set to `CSR Keyboard` in the `gap_service.c` file. The maximum length of the device name is 20 octets.

7.7. Privacy

The macro `__GAP_PRIVACY_SUPPORT__` is used to enable or disable privacy support. By default, the support of privacy is disabled. To enable support of privacy, uncomment the definition of the macro `__GAP_PRIVACY_SUPPORT__` in the `user_config.h` file.

If privacy support is enabled:

- The Keyboard application uses a resolvable random address for advertising
- GAP service characteristics 'Peripheral privacy flag' and 'Re-connection address' are supported when

- Directed advertisements use the re-connection address if it has been written by the remote device

7.8. Proprietary HID Service

The Keyboard application optionally supports the proprietary HID boot service to remain compatible with early CSR Smart HID implementations. The default configuration is to exclude the support of this proprietary HID service. To enable support of proprietary HID boot service, uncomment the definition of macro `__PROPRIETARY_HID_SUPPORT__` in the `user_config.h` file.

7.9. Non-Volatile Memory

The Keyboard application uses one of the following macros to store and retrieve persistent data in either the EEPROM or Flash-based memory.

- `NVM_TYPE_EEPROM` for I2C EEPROM.
- `NVM_TYPE_FLASH` for SPI Flash.

Note:

The macros are enabled by selecting the NVM type using the Project Properties in xIDE. This macro is defined during compilation to let the application know which NVM type it is being built for. If EEPROM is selected `NVM_TYPE_EEPROM` will be defined and for SPI Flash the macro `NVM_TYPE_FLASH` will be defined.

8. Current Consumption

The current consumed by the application can be measured by removing the Current Measuring Jumper, see Figure 2.1, which is labelled **IMEAS** and installing an ammeter in its place. The ammeter should be set to DC, measuring current from µA to mA.

The setup used while measuring current consumption is described in section 2.1. Also, the CSR µEnergy Profile Demonstrator application must be configured to accept connection parameter update requests, see Figure 2.7.

Table 8.1 shows the typical current consumption values measured when testing CSR1011 under noisy RF conditions with Channel Map Updates disabled, and with typical connection parameter values. See the Release Notes for the actual current consumption values measured for the application.

Test Scenario	**Description**	**Average Current Consumption**	**Remarks**
Fast Advertisements	1. Switch on the HID device 2. Wait for 5 s 3. Take the measurement	441 µA	▪ Advertisement Interval: 60 ms ▪ Advertisement data length: 25 octets ▪ Measurement Time Duration: 20 s
Slow Advertisements	1. Switch on the HID device 2. Wait for 40 s 3. Take the measurement	24 µA	▪ Advertising Interval: 1.28s ▪ Advertisement Data length: 25 octets ▪ Measurement Time Duration: 20 s
Connected Idle	1. Connect to the HID host 2. Wait for 60 s 3. Take the measurement	20 µA	▪ Connection Parameters ▪ Minimum connection interval: 12.5 ms ▪ Maximum connection interval: 12.5 ms ▪ Slave latency: 64 ▪ Measurement Time Duration: 60 s
Connected Active	1. Connect to the HID host 2. Wait for 60 seconds 3. HID device sends one HID report every second to the HID host by shorting pin 19 to pin17 on the CSR10x1 board 4. Take the measurement	114 µA	▪ Connection Parameters ▪ Minimum connection interval: 12.5 ms ▪ Maximum connection interval: 12.5 ms ▪ Slave latency: 64 ▪ Measurement Time Duration: 60 s

Table 8.1: Current Consumption Values

Appendix A Definitions

Term	Meaning
Input report	Low latency transfer of information from HID device to HID host
Output report	Low latency transfer of information from HID host to HID device
Pairing button press	The pairing button is pressed and held down for a period of 1 second

Table A.1: Definitions

Appendix B GATT Database

B.1 Battery Service Database

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Battery Level	`0x002a`	Read, Notify	Application	Security mode 1 and Security level 2	Current battery level
Battery Level-Client Configuration Descriptor	`0x002b`	Read, Write	Application	Security mode 1 and Security level 2	Current client configuration for “Battery level” characteristic

Table B.1: Battery Service Database

For more information on Battery Service, see *Battery Service Specification Version 1.0*.

B.2 Device Information Service Database

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Serial Number String	`0x0035`	Read	Firmware	Security Mode 1 and Security Level 2	`"BLE-KEYBOARD-001"`
Model Number String	`0x0037`	Read	Firmware	Security Mode 1 and Security Level 2	`"BLE-KEYBOARD-MODEL-001"`
Hardware Revision String	`0x0039`	Read	Firmware	Security Mode 1 and Security Level 2	*<Chip Identifier>*
Firmware Revision String	`0x003b`	Read	Firmware	Security Mode 1 and Security	*<SDK Version>*

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
				Level 2	
Software Revision String	`0x003d`	Read	Firmware	Security Mode 1 and Security Level 2	*<Application Version>*
Manufacturer Name String	`0x003f`	Read	Firmware	Security Mode 1 and Security Level 2	"`Cambridge Silicon Radio`"
PnP ID	`0x0041`	Read	Firmware	Security Mode 1 and Security Level 2	Vendor ID source is BT Vendor ID is `0x000a` Product ID is `0x014c` Product Version is 1.0.0

Table B.2: Device Information Service Database

For more information on Device Information Service, see *Device Information Service Specification Version 1.1*.

For more information on Security permissions, see *Bluetooth Core Specification Version 4.0*.

B.3 GAP Service Database

Characteristic Name	**Database Handle**	**Access Permissions**	**Managed By**	**Security Permissions**	**Value**
Device Name	`0x0003`	Read, Write	Application	Security Mode 1 and Security Level 2	Device name. Default name: "`CSR Keyboard`"
Appearance	`0x0005`	Read	Firmware	Security Mode 1 and Security Level 1	Keyboard - `0x03c1`
Peripheral preferred connection parameters	`0x0007`	Read	Firmware	Security Mode 1 and Security Level 1	Min connection interval - 12.5ms Max connection interval – 12.5ms Slave latency - 64 Connection timeout- 12.5 s
Peripheral privacy flag[1]	`0x0009`	Read, Write	Application	Security Mode 1 and Security Level 2	`0x0001` (TRUE)
Reconnection address[1]	`0x000b`	Write	Application	Security Mode 1 and Security Level 2	`0x4000 00 000000`
[1] These characteristics will only be added if privacy is enabled.					

Table B.3: GAP Service Database

For more information on GAP service and security permissions, see *Bluetooth Core Specification Version 4.0*.

B.4 HID Service Database

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
HID Information	`0x0011`	Read	Firmware	Security Mode 1 and Security Level 2	Base USB HID Specification Version - `0x0213` Country Code - `0x40` Flags - `0x01` (Remote Wakeup Supported)
Report Map	`0x0013`	Read	Firmware	Security Mode 1 and Security Level 2	See Appendix C for value
Boot Keyboard Input Report	`0x0015`	Read, Notify	Application	Security Mode 1 and Security Level 2	Current key press value in Boot mode
Boot Keyboard Input Report - Client Characteristic Configuration descriptor	`0x0016`	Read, Write	Application	Security Mode 1 and Security Level 2	Current client configuration for the "Boot Keyboard Input Report" characteristic
Boot Keyboard Output Report	`0x0018`	Read, Write, Write without response	Application	Security Mode 1 and Security Level 2	Current value of output report in Boot mode
Keyboard Input Report	`0x001a`	Read, Notify	Application	Security Mode 1 and Security Level 2	Current key press value in Report mode
Keyboard Input Report - Client Characteristic Configuration descriptor	`0x001b`	Read, Write	Application	Security Mode 1 and Security Level 2	Current client configuration for the "Keyboard Input Report" characteristic
Keyboard Input Report - Report Reference Characteristic descriptor	`0x001c`	Read	Firmware	Security Mode 1 and Security Level 2	Report ID - 1 Report Type - 1 (Input Report)
Keyboard Consumer Report	`0x001e`	Read, Notify	Application	Security Mode 1 and Security Level 2	Current consumer key press value in Report mode

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Keyboard Consumer Report - Client Characteristic Configuration descriptor	`0x001f`	Read, Write	Application	Security Mode 1 and Security Level 2	Current client configuration for the "Keyboard Consumer Report" characteristic
Keyboard Consumer Report - Report Reference Characteristic descriptor	`0x0020`	Read	Firmware	Security Mode 1 and Security Level 2	Report ID - 3 Report Type - 1 (Input Report)
Keyboard Output Report	`0x0022`	Read, Write, Write without response	Application	Security Mode 1 and Security Level 2	Current value of output report in report mode
Keyboard Output Report - Report Reference Characteristic descriptor	`0x0023`	Read	Firmware	Security Mode 1 and Security Level 2	Report ID - 1 Report Type - 2 (Output Report)
HID Control Point	`0x0025`	Write without response	Application	Security Mode 1 and Security Level 2	*<Control point value>*
Protocol Mode	`0x0027`	Read, Write without response	Application	Security Mode 1 and Security Level 2	Current protocol mode value (`0x00` - Boot Protocol Mode, `0x01` - Report Protocol Mode)

Table B.4: HID Service Database

See *Bluetooth Core Specification Version 4.0* for more information on security permissions. For more information on HID Service, see *HID Service Specification Version 1.0*.

B.5 Scan Parameter Service Database

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Scan Interval Window	`0x002f`	Write without response	Application	Security mode 1 and Security level 2	Scan interval window value written by the HID host
Scan Refresh	`0x0031`	Notify	Application	Security mode 1 and Security level 2	Server requires refresh - `0x00`
Scan Refresh - Client Configuration Descriptor	`0x0032`	Read, Write	Application	Security mode 1 and Security level 2	Current client configuration for “Scan Refresh” characteristic

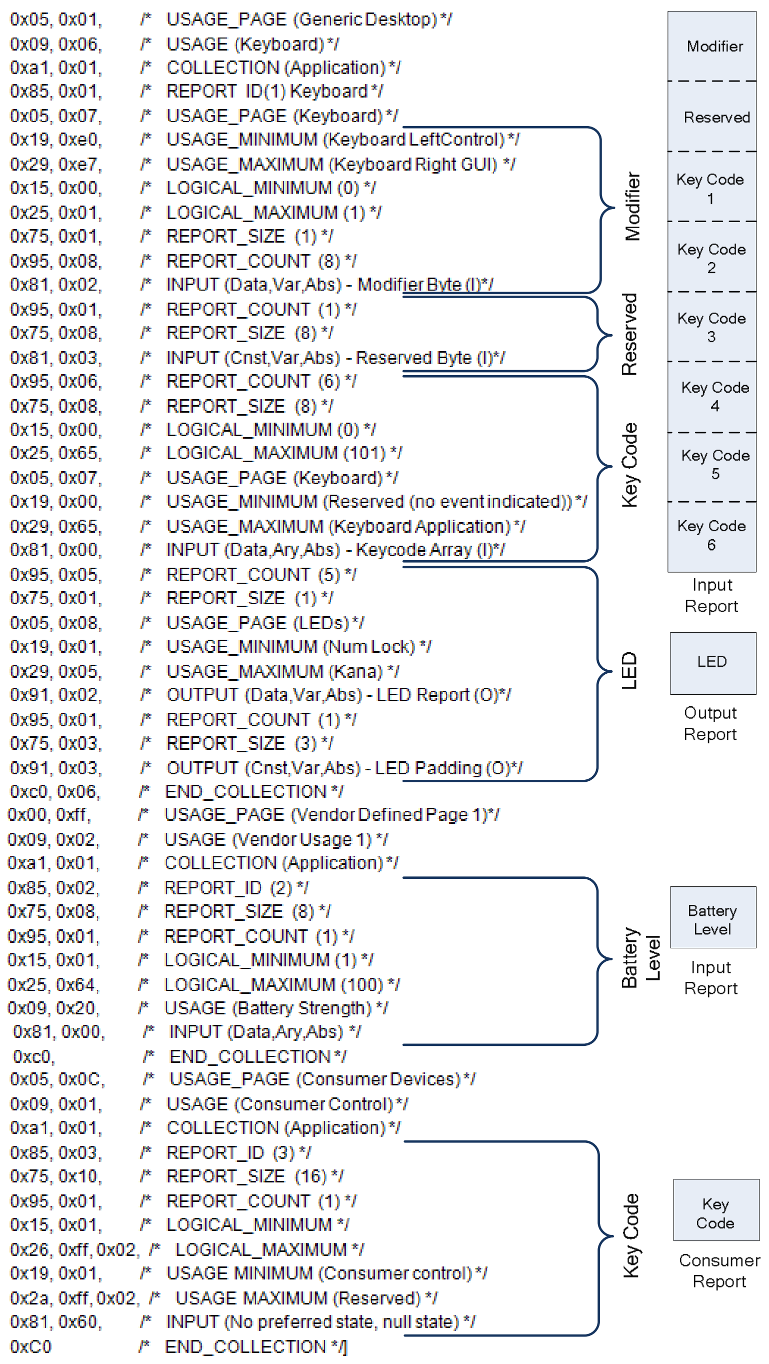
Table B.5: Scan Parameter Service Database

For more information on Scan Parameter Service, see *Scan Parameter Service Specification Version 1.0*.

Note:

Characteristics are managed by either the firmware or the application. The characteristics managed by the application have flags set to `FLAG_IRQ` in the corresponding database file. When the remote connected device accesses that characteristic, the application will receive an `GATT_ACCESS_IND` LM event and will be handled in function `AppProcessLmEvent()` defined in file `keyboard.c`. See section 4.1.2.1 for more information on the handling of the `GATT_ACCESS_IND` LM event. For more information on flags, see the *GATT Database Generator User Guide*.

Appendix C Report Map Value



```
0x05, 0x01,      /*  USAGE_PAGE (Generic Desktop) */
0x09, 0x06,      /*  USAGE (Keyboard) */
0xa1, 0x01,      /*  COLLECTION (Application) */
0x85, 0x01,      /*  REPORT ID(1) Keyboard */
0x05, 0x07,      /*  USAGE_PAGE (Keyboard) */
0x19, 0xe0,      /*  USAGE_MINIMUM (Keyboard LeftControl) */
0x29, 0xe7,      /*  USAGE_MAXIMUM (Keyboard Right GUI) */
0x15, 0x00,      /*  LOGICAL_MINIMUM (0) */
0x25, 0x01,      /*  LOGICAL_MAXIMUM (1) */
0x75, 0x01,      /*  REPORT_SIZE (1) */
0x95, 0x08,      /*  REPORT_COUNT (8) */
0x81, 0x02,      /*  INPUT (Data,Var,Abs) - Modifier Byte (I)*/
0x95, 0x01,      /*  REPORT_COUNT (1) */
0x75, 0x08,      /*  REPORT_SIZE (8) */
0x81, 0x03,      /*  INPUT (Cnst,Var,Abs) - Reserved Byte (I)*/
0x95, 0x06,      /*  REPORT_COUNT (6) */
0x75, 0x08,      /*  REPORT_SIZE (8) */
0x15, 0x00,      /*  LOGICAL_MINIMUM (0) */
0x25, 0x65,      /*  LOGICAL_MAXIMUM (101) */
0x05, 0x07,      /*  USAGE_PAGE (Keyboard) */
0x19, 0x00,      /*  USAGE_MINIMUM (Reserved (no event indicated)) */
0x29, 0x65,      /*  USAGE_MAXIMUM (Keyboard Application) */
0x81, 0x00,      /*  INPUT (Data,Ary,Abs) - Keycode Array (I)*/
0x95, 0x05,      /*  REPORT_COUNT (5) */
0x75, 0x01,      /*  REPORT_SIZE (1) */
0x05, 0x08,      /*  USAGE_PAGE (LEDs) */
0x19, 0x01,      /*  USAGE_MINIMUM (Num Lock) */
0x29, 0x05,      /*  USAGE_MAXIMUM (Kana) */
0x91, 0x02,      /*  OUTPUT (Data,Var,Abs) - LED Report (O)*/
0x95, 0x01,      /*  REPORT_COUNT (1) */
0x75, 0x03,      /*  REPORT_SIZE (3) */
0x91, 0x03,      /*  OUTPUT (Cnst,Var,Abs) - LED Padding (O)*/
0xc0, 0x06,      /*  END_COLLECTION */
0x00, 0xff,      /*  USAGE_PAGE (Vendor Defined Page 1)*/
0x09, 0x02,      /*  USAGE (Vendor Usage 1) */
0xa1, 0x01,      /*  COLLECTION (Application) */
0x85, 0x02,      /*  REPORT_ID (2) */
0x75, 0x08,      /*  REPORT_SIZE (8) */
0x95, 0x01,      /*  REPORT_COUNT (1) */
0x15, 0x01,      /*  LOGICAL_MINIMUM (1) */
0x25, 0x64,      /*  LOGICAL_MAXIMUM (100) */
0x09, 0x20,      /*  USAGE (Battery Strength) */
 0x81, 0x00,     /*  INPUT (Data,Ary,Abs) */
 0xc0,           /*  END_COLLECTION */
0x05, 0x0C,      /*  USAGE_PAGE (Consumer Devices) */
0x09, 0x01,      /*  USAGE (Consumer Control) */
0xa1, 0x01,      /*  COLLECTION (Application) */
0x85, 0x03,      /*  REPORT_ID (3) */
0x75, 0x10,      /*  REPORT_SIZE (16) */
0x95, 0x01,      /*  REPORT_COUNT (1) */
0x15, 0x01,      /*  LOGICAL_MINIMUM */
0x26, 0xff, 0x02, /*  LOGICAL_MAXIMUM */
0x19, 0x01,      /*  USAGE MINIMUM (Consumer control) */
0x2a, 0xff, 0x02, /*  USAGE MAXIMUM (Reserved) */
0x81, 0x60,      /*  INPUT (No preferred state, null state) */
0xC0             /*  END_COLLECTION */]
```

Figure C.1: Report Map Value

Appendix D Advertising and Scan Response Data

The Keyboard application adds the following fields to the Advertising data:

Advertising Data Field	Contents
Flags	The Keyboard application sets the General Discoverable Mode bit. See Section 11, Part C of Volume 3 in *Bluetooth Core Specification Version 4.0* for more information.
Service UUIDs	The Keyboard application adds a 16-bit UUID for the following service: HID
Device Appearance	Keyboard: `0x03c1`
Tx Power	Current Tx Power level
Device Name[1]	Device name (Default value : “CSR Keyboard”)
[1] The application adds the Device Name to the Scan Response data if the Device Name length is greater than the space left in the Advertising data field.	

Table D.1: Advertising Data Field

Appendix E Known Issues or Limitations

See the Keyboard application and SDK Release Notes.

Document References

Document	Reference
Bluetooth Core Specification Version 4.0	https://www.bluetooth.org/Technical/Specifications/adopted.htm
HID over GATT Profile Specification Version 1.0	https://www.bluetooth.org/Technical/Specifications/adopted.htm
HID Service Specification Version 1.0	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Battery Service Specification Version 1.0	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Device Information Service Specification Version 1.1	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Scan Parameter Service Specification Version 1.0	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Service Characteristics And Descriptions	http://developer.bluetooth.org/gatt/characteristics/Pages/default.aspx
GATT Database Generator	CS-219225-UG
CSR μEnergy xIDE User Guide	CS-212742-UG
Installing the CSR Driver for the Profile Demonstrator Application	CS-235358-UG

Terms and Definitions

ATT	Attribute
Batt	Battery
Bluetooth®	Set of wireless technologies providing audio and data transfer over short-range radio connections
Bluetooth Smart	Formerly known as Bluetooth Low Energy
BT	Bluetooth
CFM	Confirmation
CSR®	Cambridge Silicon Radio
DIV	Diversifier
ID	IDentifier
EEPROM	Electrically Erasable Programmable Read Only Memory
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GND	Ground
HID	Human Interface Device
i.e.	*Id est*, that is
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IND	Indication
IRK	Identity Resolving Key
L2CAP	Logical Link Control and Adaptation Protocol
LM	Link Manager
NVM	Non Volatile Memory
PC	Personal Computer
PIO	Programmable Input Output
PnP	Plug and Play
SDK	Software Development Kit
SMP	Security Manager Protocol

SPI	Serial Peripheral Interface
Tx	Transmit
USB	Universal Serial Bus
UUID	Universally Unique Identifier
xIDE	CSR's Integrated Development Environment
CSR10x1	CSR1001 and CSR1011 devices