Application Note

HID over GATT Profile (HOGP) Bluetooth® Low Energy

Jan Iglesias Morales

ABSTRACT

This application report provides detailed documentation and showcases the HID Over GATT Profile (HOGP) *Bluetooth*® Low Energy example. A thorough description of the project is provided within this document, which was developed using the TI SimpleLink™ Bluetooth Low Energy software stack. Custom HOGP projects may use the information contained within this document and the HOGP BLE Project as a starting point.

The accompanying software example can be found on the Texas Instruments GitHub.

Table of Contents

Trademarks

1 Introduction

Human Interface Devices (HID) are devices whose primary purpose is to allow users to interface with computers. HID is based on a specification that is maintained by the USB Implementers Forum's Device Working Group. HIDs cover a variety of devices from monitors, keyboards, webcams, and mice all the way to VR headsets, gamepads, and touch screens. HIDs have traditionally connected to computers through the use of a physical USB connection. However, HID devices are also able to interface with a computer through the use of wireless technology. Through the use of the HID specification, one is able to develop computer peripheral devices that follow a general standard, which enables compatibility with most devices.

2 Bluetooth Low Energy Introduction

Bluetooth Low Energy is a personal local network technology that is designed and driven by Bluetooth Special Interest Group (Bluetooth SIG). Bluetooth Low Energy is primarily designed for enabling connectivity for applications with very low power requirements and has been used across a broad range of applications such as personal electronics, wearables, medical, building automation and industrial automation.

The TI SimpleLink Bluetooth Low Energy CC26x2 family of wireless microcontrollers can implement a low-latency BLE peripheral role which has been used to facilitate the HOGP functionality in this project. This application report describes our implementation of the HOGP (HID Over GATT Profile) project on the CC26x2 family of devices and how the project functions in detail.

3 HOGP Introduction

The HID Over GATT Profile (HOGP) is a Bluetooth profile specification that is maintained by the Bluetooth Special Interest Group. The HOGP is a profile that implements HID over the Bluetooth Low Energy technology. By using Bluetooth Low Energy as the interface between the HID and the computer device, the HOGP is able to eliminate the need for wires or a physical connection in the HIDs. This application report covers the HOGP example available in the TI GitHub for the CC26x2 devices.

3.1 HID Roles

In HID, and by extension in HOGP, there are two defined roles. These roles are the HID Host and the HID device. An HID device can only be actively connected to a single HID host at time. An HID host can be connected to many HID devices. For more details on how the Host and Device interact with each other, see Chapter 3 *HID Device Requirements*, and Chapter 4 *HID Host Requirements and Behaviors* in the HOGP specification.

3.2 HID Host

The HID Host implements the Bluetooth central role. The HID Host is the device that receives the input information and does further processing with the provided data. For instance, a computer would be an HID Host since this is the device that receives the HID data. The HID Host has many responsibilities and requirements for the HID functionality to properly occur, but the example that this application report is covering implements the HID Device, so these responsibilities will not be covered in this document. To see what responsibilities and requirements a HID Host has, see Chapter 2 *Configuration*, Chapter 4 *HID Host Requirements and Behavior HOGP specification*, Chapter 5 *Connection Establishment* in the HOGP specification.

3.3 HID Device

The HID device implements the Bluetooth peripheral role. The HID device transmits input information to a central computer. For instance, a mouse or keyboard would be HID devices because they send input data to the computer device.

4 Project Description and Walkthrough

The following sections provide an in-depth description of all the major components of this project.

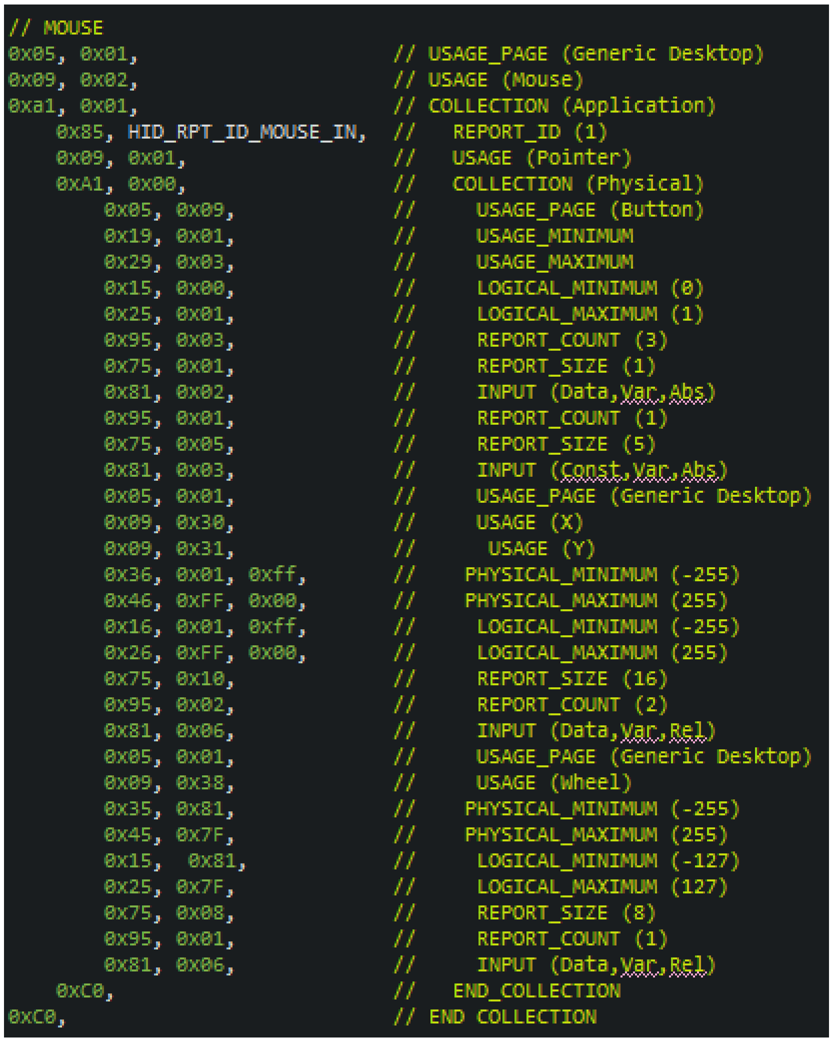
4.1 General Project Discussion

The hid_emu_kbd project is a Bluetooth Low Energy project that implements the HOGP specification on the CC26x2 devices. The project uses the Bluetooth peripheral role as specified by the HOGP specification. The project demos the mouse, keyboard, and consumer report portions of the HOGP. Through the use of notifications and the GATT profile, the project is able to implement HID over Bluetooth Low Energy and properly interface with an HID Host. The hid_emu_kbd project has several complex parts that are discussed in this document. These parts come together in order to implement the overall functionality.

4.2 Report Map Discussion

The hid_emu_kbd project relies on report maps. Report maps are a structure used by the HOGP to implement the HID over Bluetooth Low Energy functionality. Report maps are used to tell the HID Host what functionality and what kind of data to expect from the HID Device. In the original HID USB specification, report maps are called report descriptors. Report maps and report descriptors are identical. The only difference between them is that report maps are used in HOGP and report descriptors are used in HID USB.

The report map used by this project can be found in the hidkbdservice.c file and is contained within the hidReportMap variable. The report map in this project contains all of the input information for the keyboard, mouse, and consumer reports. The structure is divided into sections and the application as well as the HID Host is able to find the desired section through the use of the Report ID field. The report map can be easily modified to include different inputs or to modify the inputs that are already there. The USB Implementers Forum's Device Working Group has a Report Descriptor builder tool that can be used to create report maps easily.



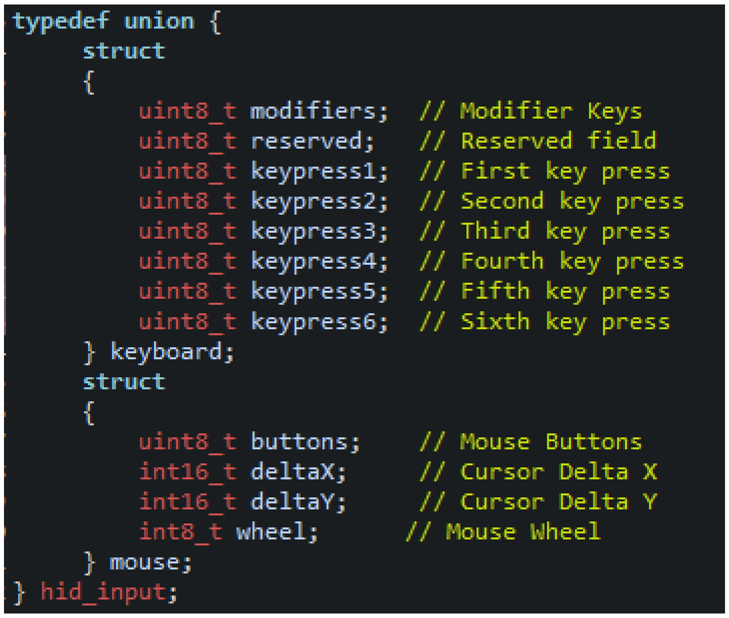
```
// MOUSE
0x05, 0x01,                       // USAGE_PAGE (Generic Desktop)
0x09, 0x02,                       // USAGE (Mouse)
0xa1, 0x01,                       // COLLECTION (Application)
    0x85, HID_RPT_ID_MOUSE_IN,    //   REPORT_ID (1)
    0x09, 0x01,                   //   USAGE (Pointer)
    0xA1, 0x00,                   //   COLLECTION (Physical)
        0x05, 0x09,               //     USAGE_PAGE (Button)
        0x19, 0x01,               //     USAGE_MINIMUM
        0x29, 0x03,               //     USAGE_MAXIMUM
        0x15, 0x00,               //     LOGICAL_MINIMUM (0)
        0x25, 0x01,               //     LOGICAL_MAXIMUM (1)
        0x95, 0x03,               //     REPORT_COUNT (3)
        0x75, 0x01,               //     REPORT_SIZE (1)
        0x81, 0x02,               //     INPUT (Data,Var,Abs)
        0x95, 0x01,               //     REPORT_COUNT (1)
        0x75, 0x05,               //     REPORT_SIZE (5)
        0x81, 0x03,               //     INPUT (Const,Var,Abs)
        0x05, 0x01,               //     USAGE_PAGE (Generic Desktop)
        0x09, 0x30,               //     USAGE (X)
        0x09, 0x31,               //      USAGE (Y)
        0x36, 0x01, 0xff,         //    PHYSICAL_MINIMUM (-255)
        0x46, 0xFF, 0x00,         //    PHYSICAL_MAXIMUM (255)
        0x16, 0x01, 0xff,         //     LOGICAL_MINIMUM (-255)
        0x26, 0xFF, 0x00,         //     LOGICAL_MAXIMUM (255)
        0x75, 0x10,               //     REPORT_SIZE (16)
        0x95, 0x02,               //     REPORT_COUNT (2)
        0x81, 0x06,               //     INPUT (Data,Var,Rel)
        0x05, 0x01,               //     USAGE_PAGE (Generic Desktop)
        0x09, 0x38,               //     USAGE (Wheel)
        0x35, 0x81,               //    PHYSICAL_MINIMUM (-255)
        0x45, 0x7F,               //    PHYSICAL_MAXIMUM (255)
        0x15,  0x81,              //     LOGICAL_MINIMUM (-127)
        0x25, 0x7F,               //     LOGICAL_MAXIMUM (127)
        0x75, 0x08,               //     REPORT_SIZE (8)
        0x95, 0x01,               //     REPORT_COUNT (1)
        0x81, 0x06,               //     INPUT (Data,Var,Rel)
    0xC0,                         //   END_COLLECTION
0xC0,                             // END COLLECTION
```

Figure 4-1. The Mouse Section of the Report Map

Figure 4-1 shows a snippet of the report map structure. All fields are labeled and can easily be modified. For example, if a different amount of mouse buttons is desired, then the relevant fields can be easily updated.

4.3 Hid_input struct/union Discussion

To facilitate development, an HID input union was created. The union, named hid_input, holds a keyboard structure and mouse structure. Through the use of the hid_input, an hid input can be easily packaged for transmission. The union definition is shown in Figure 4-2.



```
typedef union {
     struct
     {
         uint8_t modifiers;  // Modifier Keys
         uint8_t reserved;   // Reserved field
         uint8_t keypress1;  // First key press
         uint8_t keypress2;  // Second key press
         uint8_t keypress3;  // Third key press
         uint8_t keypress4;  // Fourth key press
         uint8_t keypress5;  // Fifth key press
         uint8_t keypress6;  // Sixth key press
     } keyboard;
     struct
     {
         uint8_t buttons;    // Mouse Buttons
         int16_t deltaX;     // Cursor Delta X
         int16_t deltaY;     // Cursor Delta Y
         int8_t wheel;      // Mouse Wheel
     } mouse;
} hid_input;
```

Figure 4-2. The hid_input Union Definition

As shown in Figure 4-2, a variable of type hid_input contains all of the information necessary for the HID Device to communicate with the HID host the necessary input information. For the keyboard section of the union, the structure is able to send several key presses at a time, if desired. The keyboard section is also used to transmit consumer reports. The mouse section of the union sends relative position data, mouse button presses, and mouse wheel data. This structure can be easily modified to implement a different HID use-case.

4.4 Mouse Operation

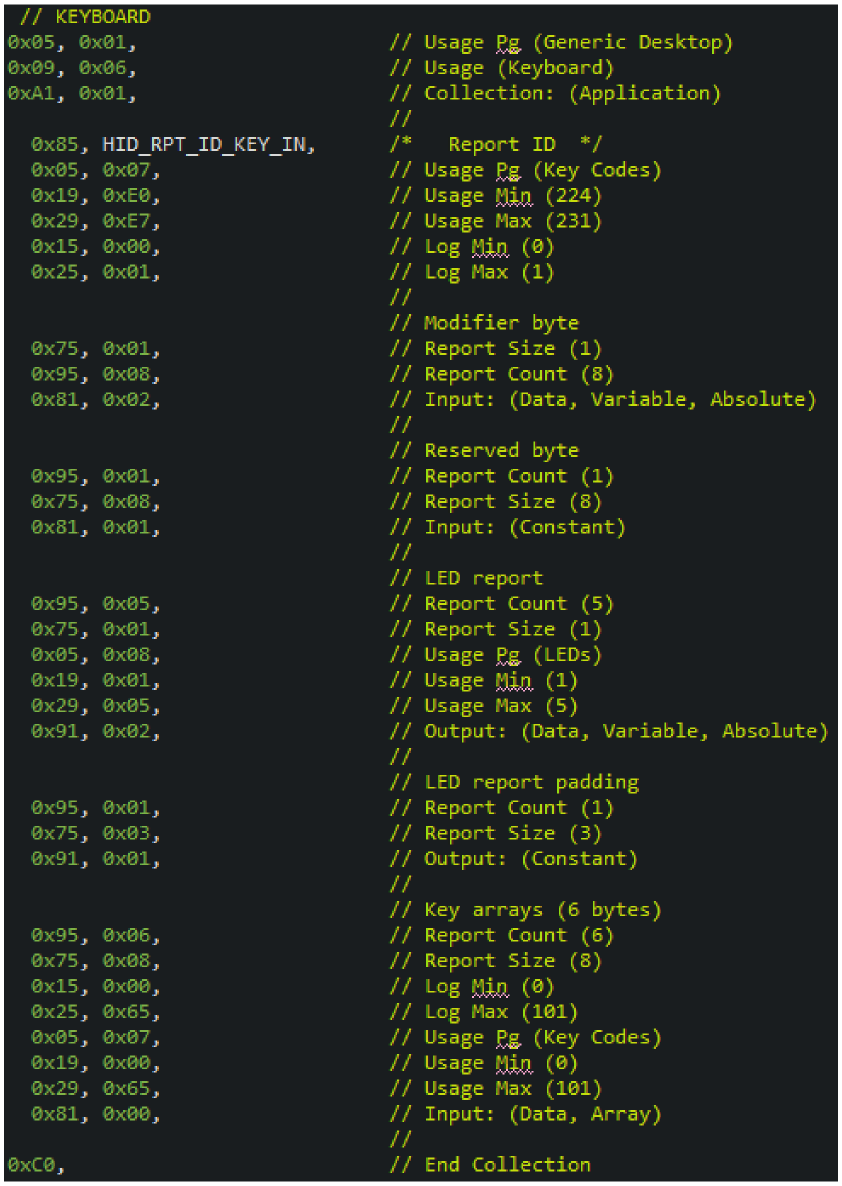
The hid_emu_kbd implements HID mouse operations. The mouse operation is implemented through the use of the hid_input union. The mouse input data that this program is considering is the mouse buttons, the relative horizontal (x) position, the relative vertical (y) position, and the mouse wheel. The buttons member of the mouse structure and it consists of 8 bits. In the report map, the USAGE_MINIMUM and USAGE_MAXIMUM fields mark that the left mouse button, middle mouse button, and right mouse button are the buttons that the mouse will implement. Bit 0 corresponds to the right mouse button, bit 1 corresponds to the middle mouse button, and bit 2 corresponds to the left mouse button. Additional buttons may be added if the use-case requires it. To indicate a button being pressed, the relevant bit should be set and to signify a button not being pressed the relevant bit should be cleared.

The mouse movement portion of the mouse operation is a bit more complicated. A relative movement value is provided to the deltaX and deltaY members. Both of these members are signed 16-bit values each. This can be seen in the report map as well as the data type used for the deltaX and deltaY members. The values accepted range from -255 to 254. The mouse section of the report map is shown in Figure 4-1.

Once the mouse data has been packaged in a hid_input variable within the mouse structure, the HidEmuKbd_sendMouseInput() function may be called to send the actual mouse input to the HID Host. Its parameter would be the hid_input variable. In this project, this is all done in the demo portion within the HidEmuKbd_mouseTaskFxn() function through the use of events, messages, and semaphores. This can be changed and modified as needed.

4.5 Keyboard Operation

The project also implements HID keyboard operation. The keyboard functionality is implemented in a manner that is very similar to how the mouse functionality is implemented. The hid_input union is used to package the keyboard input. Specifically, the keyboard structure is packaged with the desired inputs. Several keypresses can be sent in a single transmission. The hiddev.h file contains all of the keyboard/keypad usage IDs. The aforementioned IDs are used to send a specific key press. The section of the report map that corresponds to the keyboard operation is shown in Figure 4-3.



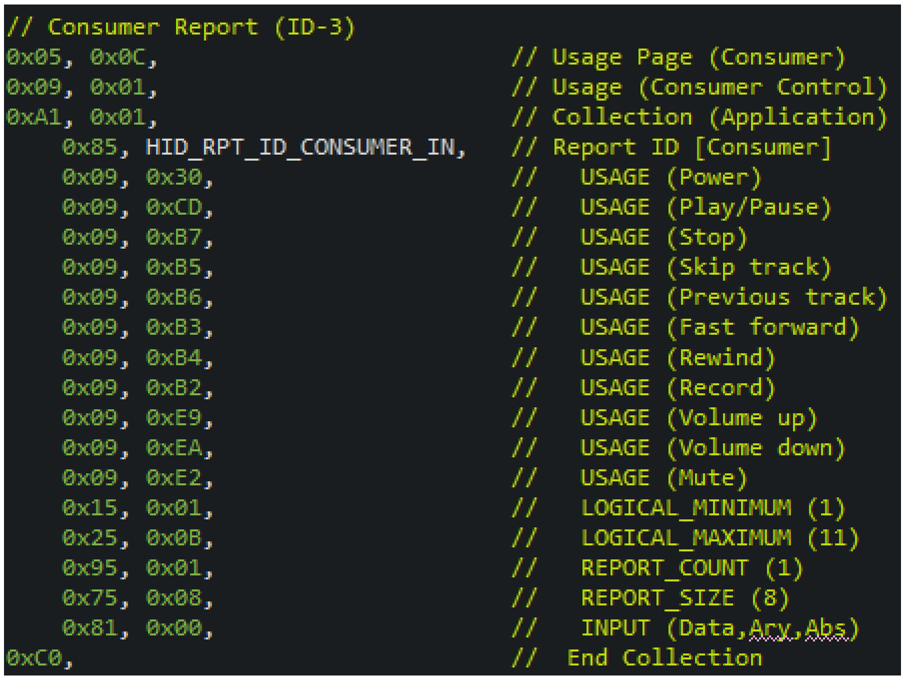
```
 // KEYBOARD
0x05, 0x01,                     // Usage Pg (Generic Desktop)
0x09, 0x06,                     // Usage (Keyboard)
0xA1, 0x01,                     // Collection: (Application)
                                //
  0x85, HID_RPT_ID_KEY_IN,      /*   Report ID  */
  0x05, 0x07,                   // Usage Pg (Key Codes)
  0x19, 0xE0,                   // Usage Min (224)
  0x29, 0xE7,                   // Usage Max (231)
  0x15, 0x00,                   // Log Min (0)
  0x25, 0x01,                   // Log Max (1)
                                //
                                // Modifier byte
  0x75, 0x01,                   // Report Size (1)
  0x95, 0x08,                   // Report Count (8)
  0x81, 0x02,                   // Input: (Data, Variable, Absolute)
                                //
                                // Reserved byte
  0x95, 0x01,                   // Report Count (1)
  0x75, 0x08,                   // Report Size (8)
  0x81, 0x01,                   // Input: (Constant)
                                //
                                // LED report
  0x95, 0x05,                   // Report Count (5)
  0x75, 0x01,                   // Report Size (1)
  0x05, 0x08,                   // Usage Pg (LEDs)
  0x19, 0x01,                   // Usage Min (1)
  0x29, 0x05,                   // Usage Max (5)
  0x91, 0x02,                   // Output: (Data, Variable, Absolute)
                                //
                                // LED report padding
  0x95, 0x01,                   // Report Count (1)
  0x75, 0x03,                   // Report Size (3)
  0x91, 0x01,                   // Output: (Constant)
                                //
                                // Key arrays (6 bytes)
  0x95, 0x06,                   // Report Count (6)
  0x75, 0x08,                   // Report Size (8)
  0x15, 0x00,                   // Log Min (0)
  0x25, 0x65,                   // Log Max (101)
  0x05, 0x07,                   // Usage Pg (Key Codes)
  0x19, 0x00,                   // Usage Min (0)
  0x29, 0x65,                   // Usage Max (101)
  0x81, 0x00,                   // Input: (Data, Array)
                                //
0xC0,                           // End Collection
```

Figure 4-3. The Keyboard Section of the Report Map

The keyboard portion of the hid_input union be filled with several key presses, modifier keys, and a reserved field. Once the keyboard data has been packaged, the HidEmuKbd_sendKeyboardInput() function may be called to send a keyboard input transmission to the HID Host. In the demo portion of this project, keypresses are sent through the use of events that are queued up within the HidEmuKbd_keyPressHandler() function.

4.6 Consumer Report Operation

HID consumer reports are a special type of HID input which enable many additional HID features such as volume control, media playback control, and power controls. These inputs are commonly used in media keys on keyboards. HID Consumer reports are implemented in project. The functionality of the consumer report operation is nearly identical to that of the keyboard operation. The keyboard structure within the hid_input union is used for packaging the input. However, instead of using the keypress members to package the input, the consumer reports use the reserved member of the structure. The consumer report IDs that are supported by the project can be found in the hidemukbd.c file. The section of the report map that corresponds to the consumer report operation is shown in Figure 4-4.



```
// Consumer Report (ID-3)
0x05, 0x0C,                       // Usage Page (Consumer)
0x09, 0x01,                       // Usage (Consumer Control)
0xA1, 0x01,                       // Collection (Application)
    0x85, HID_RPT_ID_CONSUMER_IN, // Report ID [Consumer]
    0x09, 0x30,                   //   USAGE (Power)
    0x09, 0xCD,                   //   USAGE (Play/Pause)
    0x09, 0xB7,                   //   USAGE (Stop)
    0x09, 0xB5,                   //   USAGE (Skip track)
    0x09, 0xB6,                   //   USAGE (Previous track)
    0x09, 0xB3,                   //   USAGE (Fast forward)
    0x09, 0xB4,                   //   USAGE (Rewind)
    0x09, 0xB2,                   //   USAGE (Record)
    0x09, 0xE9,                   //   USAGE (Volume up)
    0x09, 0xEA,                   //   USAGE (Volume down)
    0x09, 0xE2,                   //   USAGE (Mute)
    0x15, 0x01,                   //   LOGICAL_MINIMUM (1)
    0x25, 0x0B,                   //   LOGICAL_MAXIMUM (11)
    0x95, 0x01,                   //   REPORT_COUNT (1)
    0x75, 0x08,                   //   REPORT_SIZE (8)
    0x81, 0x00,                   //   INPUT (Data,Ary,Abs)
0xC0,                             //  End Collection
```

Figure 4-4. The Consumer Report Section of the Report Map

Once the consumer report input has been packaged in the hid_input variable, then the input transmission may be sent to the HID Host by calling the HidEmuKbd_sendConsumerInput() function. The demo portion of this project, sends consumer reports alongside keyboard button presses when a button on the LaunchPad is pressed.

4.7 Connection Interval

The HOGP specification states that the connection interval for device that implements the HOGP must be in the range of 7.5 ms to 50 ms. This project implements a 7.5 ms connection interval by default, but this can be changed to another value if desired. For more information on how to change the connection interval, please refer to the SDK User's Guide. A connection interval of 7.5 ms provides the fastest response time and lowest latency for the hid_emu_kbd. Figure 4-5

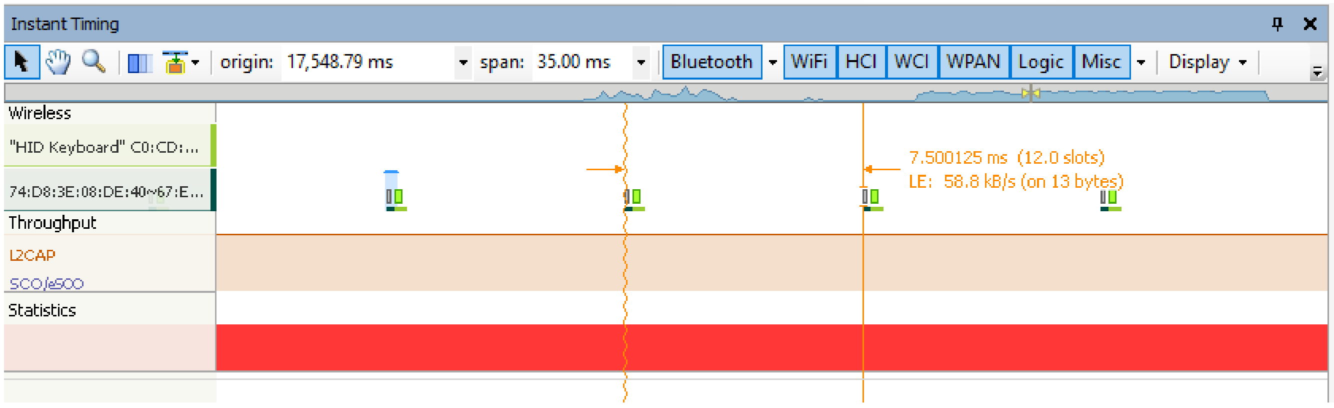


Figure 4-5. Bluetooth Traffic Log Snippet Showing 7.5 ms Connection Interval

The connection interval is very important to performance because the HID input data is sent to the HID Host via notifications that can only occur during a connection event.

4.8 Notification System

As mentioned previously, the HID inputs are sent through BLE notifications. When the HidEmuKbd_sendKeyboardInput(), HidEmuKbd_sendMouseInput(), and HidEmuKbd_sendConsumerInput() functions are called, they take the hid input, process the input, package the input in a notification format, and then queue up a notification with the payload. In the demo portion of this program, notifications are sent routinely. When the buttons on the LaunchPad are pressed, several notifications are sent. When the mouse demo is started, mouse input notifications will be queued up repeatedly. The mouse repeated notification enqueuing occurs in the HidEmuKbd_mouseTaskFxn() function. A small delay between enqueuing notifications is added to make the mouse movement smoother, but this can be removed, if desired. It is critical that notifications are sent reliably and with minimal latency in order for the HOGP performance to be maximized.

4.9 PDU Size and Number of PDUs per Connection Event

The PDU size and PDU amount have a direct impact on notifications. The max size of PDUs is set to 69 bytes in the project by default. This is the minimum size that should be used for this project because this is the minimum size required by the LE Secured Connections feature. The number of PDUs per connection event has a more direct impact on the project. The mouse demo portion of the project constantly enqueues notifications. If a project is enqueuing notifications at a rapid pace, then a large number of PDUs is required per connection event in order to keep up with the stream of notifications. If the amount of PDUs is set lower than the notification stream requires, then data loss may occur. The project has the PDU amount per connection event set to the maximum of 255 PDUs. This setting should be minimized in a final implementation of this project to a value that offers minimal data loss while also offering minimal heap consumption. For additional information on PDUs and their impact, see the Link Layer (LL) chapter of the SDK User's Guide.

4.10 Notification Payload Discussion

The notification payload is essentially the HID data that the HID Host requires to perform any HID operation. The following sections describe the notifications generated by each type of HID input that this program generates as well as their payloads.

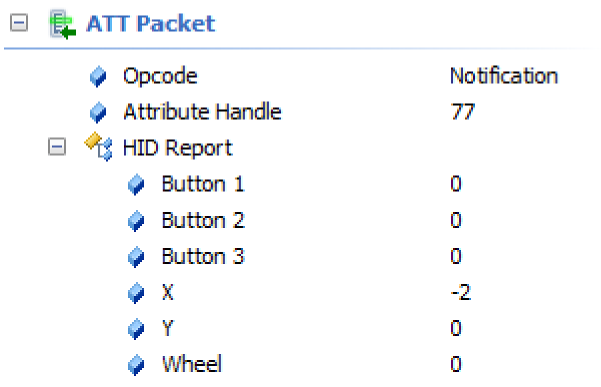
4.10.1 Mouse Notification

The mouse notifications have a payload that is comprised of 6 bytes. Each byte corresponds to a specific piece of HID mouse input information. Table 4-1 describes what each byte of the mouse notification payload corresponds to.

Table 4-1. Mouse Notification Payload Fields

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Mouse Buttons	Horizontal (X) Relative Position		Vertical (Y) Relative Position		Mouse Wheel

An example of a mouse notification payload that is generated by this project is “00 FE FF 00 00 00”, which would be parsed by the HID Host as the shown in Figure 4-6.


ATT Packet

Opcode	Notification
Attribute Handle	77
HID Report	
Button 1	0
Button 2	0
Button 3	0
X	-2
Y	0
Wheel	0

Figure 4-6. Parsed Mouse Notification Payload

For more details on how the HOGP specification makes use of notification, see Chapter 4 *HID Host Requirements and Behaviors* and Chapter 5 *Connection Establishment* in the HOGP specification.

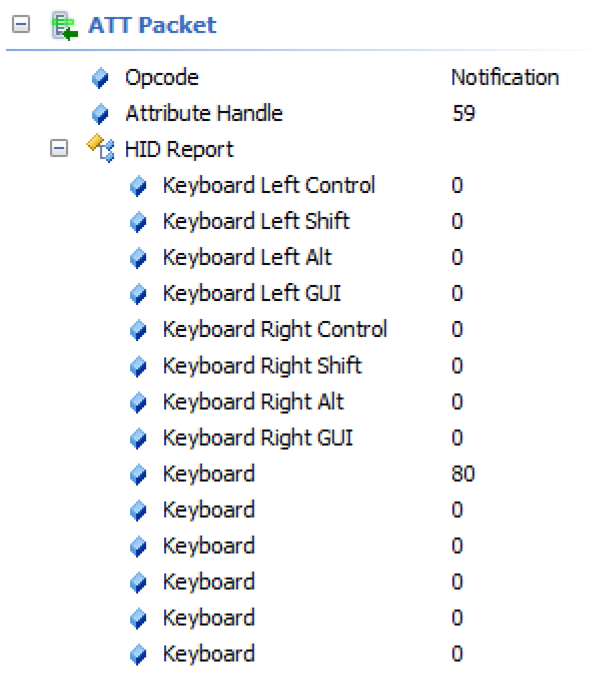
4.10.2 Keyboard Notification

The keyboard notifications have a payload that is comprised of 8 bytes. Each byte corresponds to a specific piece of HID keyboard input information. Table 4-2 describes what each byte of the keyboard notification payload corresponds to.

Table 4-2. Keyboard Notification Payload Fields

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Modifier Key	Reserved	Keypress 1	Keypress 2	Keypress 3	Keypress 4	Keypress 5	Keypress 6

An example of a keyboard notification payload that is generated by this project is “00 00 50 00 00 00 00 00”, which would be parsed by the HID Host as shown in Figure 4-7.



ATT Packet

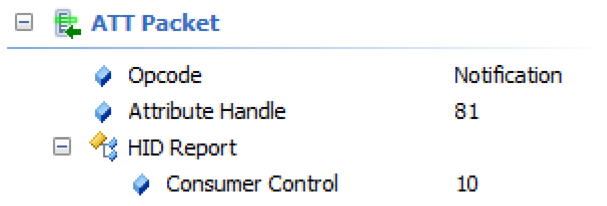
Opcode	Notification
Attribute Handle	59
HID Report	
Keyboard Left Control	0
Keyboard Left Shift	0
Keyboard Left Alt	0
Keyboard Left GUI	0
Keyboard Right Control	0
Keyboard Right Shift	0
Keyboard Right Alt	0
Keyboard Right GUI	0
Keyboard	80
Keyboard	0
Keyboard	0
Keyboard	0
Keyboard	0
Keyboard	0

Figure 4-7. Parsed Keyboard Notification Payload

For more details on how the HOGP specification makes use of notification, see Chapter 4 *HID Host Requirements and Behaviors* and Chapter 5 *Connection Establishment* in the HOGP specification.

4.10.3 Consumer Report Notification

The consumer report notifications have a payload that is comprised of a single byte. The byte corresponds to the Consumer Control ID that is being sent. An example of a consumer report notification payload that is generated by this project is “0A” which would be parsed by the HID Host as shown in Figure 4-8.



ATT Packet

Opcode	Notification
Attribute Handle	81
HID Report	
Consumer Control	10

Figure 4-8. Parsed Consumer Report Notification Payload

4.11 Throughput Discussion

When sending constant data to the HID Host, the performance of the notification stream becomes critical. If the HID mouse requires very low latency and sends frequent HID inputs, then good throughput is essential. This project has been observed to have a peak throughput of approximately 1.6 KBps during the mouse demo. Figure 4-9 illustrates this throughput.

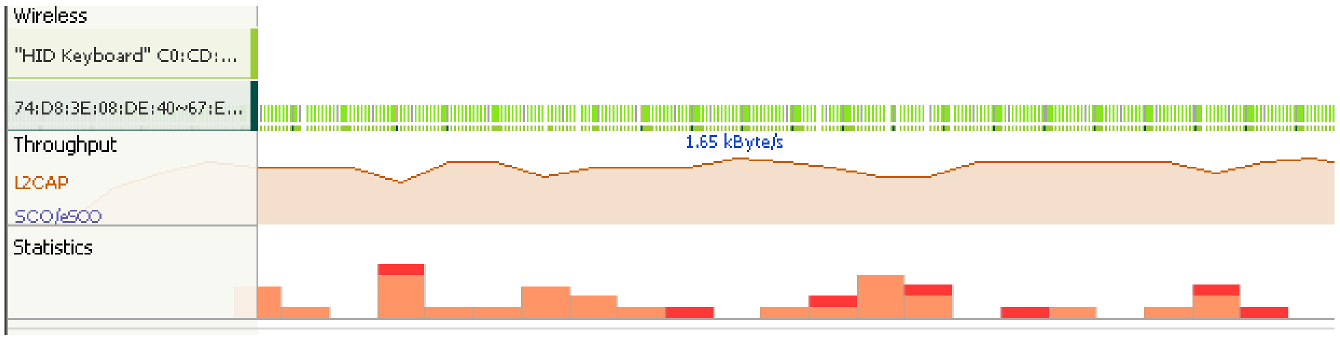


Figure 4-9. Throughput Measured During Mouse Demo

A 1.6kBps throughput is sufficient for enabling a low latency experience. However, this is not the maximum throughput achievable by this device. The Bluetooth 5 Throughput Demo showcases what a high (near theoretical max) throughput can be expected from this device.

4.12 Overall Block Diagrams

The processes and components discussed in the previous sections come together to implement the HOGP project. Figure 4-10 shows the process in which the report map is shared with the HID Host device.

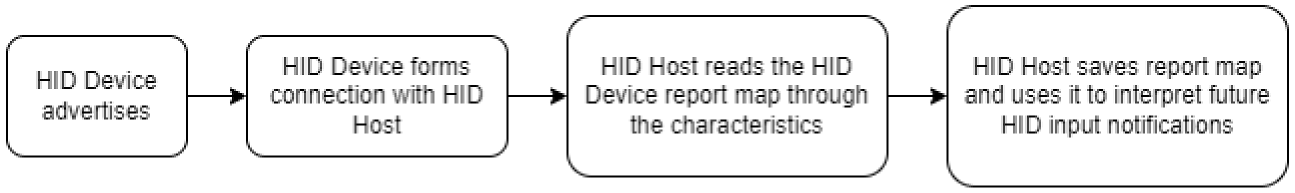

HID Device advertises
HID Device forms connection with HID Host
HID Host reads the HID Device report map through the characteristics
HID Host saves report map and uses it to interpret future HID input notifications

Figure 4-10. Report Map Exchange Block Diagram

As shown in Figure 4-10, the report map is provided to the HID Host shortly after the connection is established. After the report map is provided to the HID Host, the Host will save it and use it to interpret any future notifications sent by the HID Device.

Figure 4-11 shows how the HOGP project, acting as an HID Device, transmits HID inputs to the HID Host.

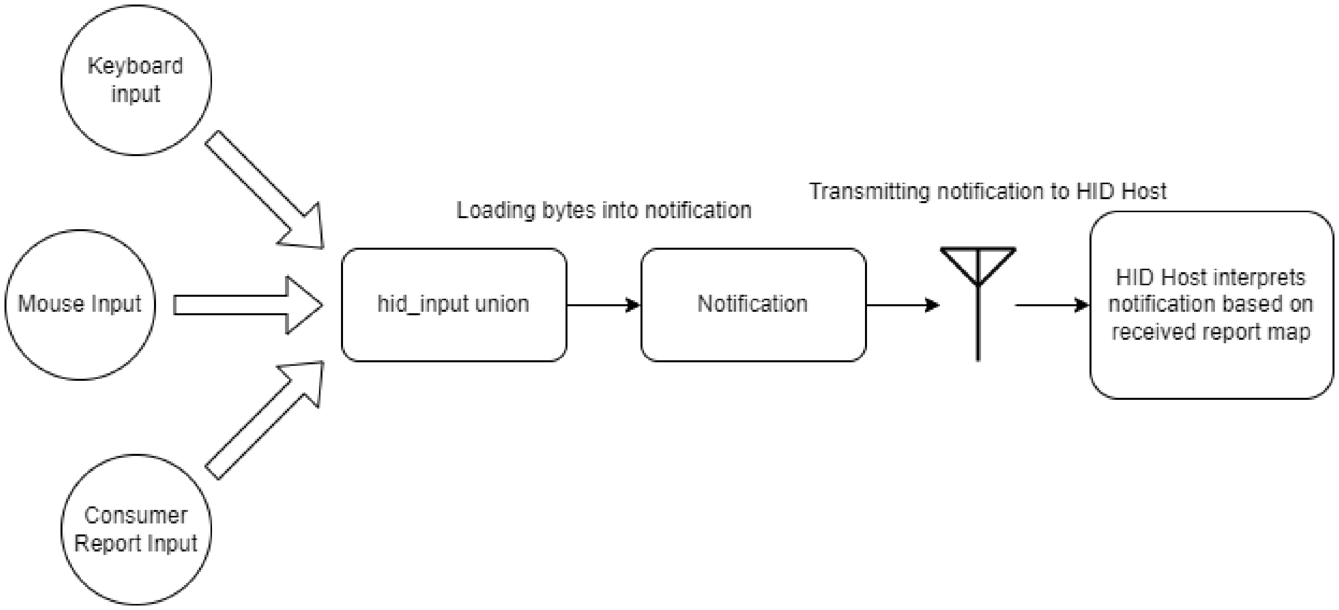

Keyboard input
Mouse Input
Consumer Report Input
Loading bytes into notification
Transmitting notification to HID Host
hid_input union
Notification
HID Host interprets notification based on received report map

Figure 4-11. HID Input Transmission Block Diagram

5 Demo Usage

The following sections discuss how to get started with the hid_emu_kbd project demo. The hardware necessary is discussed as well as the demo features and instructions.

5.1 Hardware/Software Used

The hardware/software used during the development of this project and the hardware/software expected to be used is displayed in Table 5-1.

Table 5-1. System Requirements

PC Operating System	Windows 10 – 64 Bit
Development Board	LAUNCHXL-CC26X2R1
SDK Version	SimpleLink CC13x2 26x2 SDK 5.10
IDE Version	CCS 10.3

The HOGP implementation should function with any properly implemented HID Host, but it was only extensively tested in the environment mentioned above. The latest released version of the HOGP BLE project can be found in the Texas Instruments GitHub. This example will be updated periodically as needed.

5.2 Mouse Demo Usage

To initiate the demo, follow the steps outlined below:

1. Compile and flash the hid_emu_kbd project onto a CC26X2R LaunchPad.
   a. For information on how to compile and flash projects onto this device, see the SimpleLink Academy Bluetooth Low Energy Fundamentals course.
2. Once the device has been flashed with the project. Connect the computer (HID Host) to the CC26X2R LaunchPad. By default, the device should be named “HID Keyboard” and should appear as such in the discovered device list.
3. Once the computer has connected and bonded with the CC26X2R Launchpad, the mouse movement demo may be started by pressing both buttons (the left and right buttons) simultaneously.
4. If the previous steps are done successfully, the mouse demo should begin and the cursor should begin to move in a square formation.
5. To deactivate the demo, press both buttons once more or reset the LaunchPad.
6. The mouse demo does not interfere with the keyboard and consumer report demo.

To initiate the demo, the hid_emu_kbd project must first be flashed onto the LaunchPad and connected to the computer (HID Host). Once the device is connected, the mouse demo mode can be initiated by pressing both buttons on the LaunchPad simultaneously. Once both buttons are pressed, the cursor will begin to move in a square formation. This continues until the mouse demo mode is deactivated or the LaunchPad is reset. To deactivate the mouse demo, both buttons must be pressed again.

5.3 Keyboard and Consumer Report Demo Usage

To initiate the demo, follow the steps outlined below:

1. Compile and flash the hid_emu_kbd project onto a CC26X2R LaunchPad.
   - For information on how to compile and flash projects onto this device, see the SimpleLink Academy Bluetooth Low Energy Fundamentals course.
2. Once the device has been flashed with the project. Connect the computer (HID Host) to the CC26X2R LaunchPad. By default, the device should be named “HID Keyboard” and should appear as such in the discovered device list.
3. Once the computer has connected and bonded with the CC26X2R Launchpad, the keyboard and consumer report demo may be triggered.
4. Press the left button to send a left arrow keystroke and a decrease volume consumer report.
5. Press the right button to send a right arrow keystroke and an increase volume consumer report.
6. The keyboard and consumer report demo does not interfere with the mouse demo.

6 Summary

The hid_emu_kbd project properly implements the HOGP specification published by the Bluetooth Special Interest Group. It offers a demonstration of all of the project’s features in an easy to use format. The hid_emu_kbd project can be modified to act like any HID device that is supported by the HOGP, such as gaming controllers, joysticks, steering wheels, among others. The 7.5 ms connection provides low latency and rapid data transmission with reliable data throughput. The CC26X2 device is shown to successfully implement the hid_emu_kbd project and can serve as the Bluetooth device within a fully-fledged HID product.