AN12649

OTA Process Introduction

Rev. 0 — 01/2020

Application Note

Contents

1 Introduction

The K32L2B is highly-integrated, market leading ultra low-power 32-bit microcontroller based on the enhanced Cortex-M0+ (CM0+) core platform. The family derivatives contain the following features:

- Core platform clock up to 48 MHz, bus clock up to 24 MHz.
- Memory option is up to 256 KB flash and 32 KB RAM.
- Two SPI modules.
- Two inter-integrated circuit (I2C) modules.
- One FlexIO module.

Over-The-Air (OTA) is a procedure to update the firmware without the use of physical wires, so it can be a solution for the current **Bluetooth LE Audio System**. When a product is ready and released in the field, OTA can be used to upload new firmware that brings new features.

- From the point view of a customer, OTA is convenient because the Headset does not need to be connected to a PC.
- From the point view of a manufacturer, OTA reduces the BOM cost as no USB hardware needs to be present.

This document provides **OTA Operation Steps** to support users who would like to use the NXH3670_SDK_Gaming_G3.0 tool to update firmware in memory of K32L2B easily.

For more specific OTA introductions, including HAPI and Concept, refer to **HAPI OTA** of KL27.

2 Concepts

2.1 OTA update process

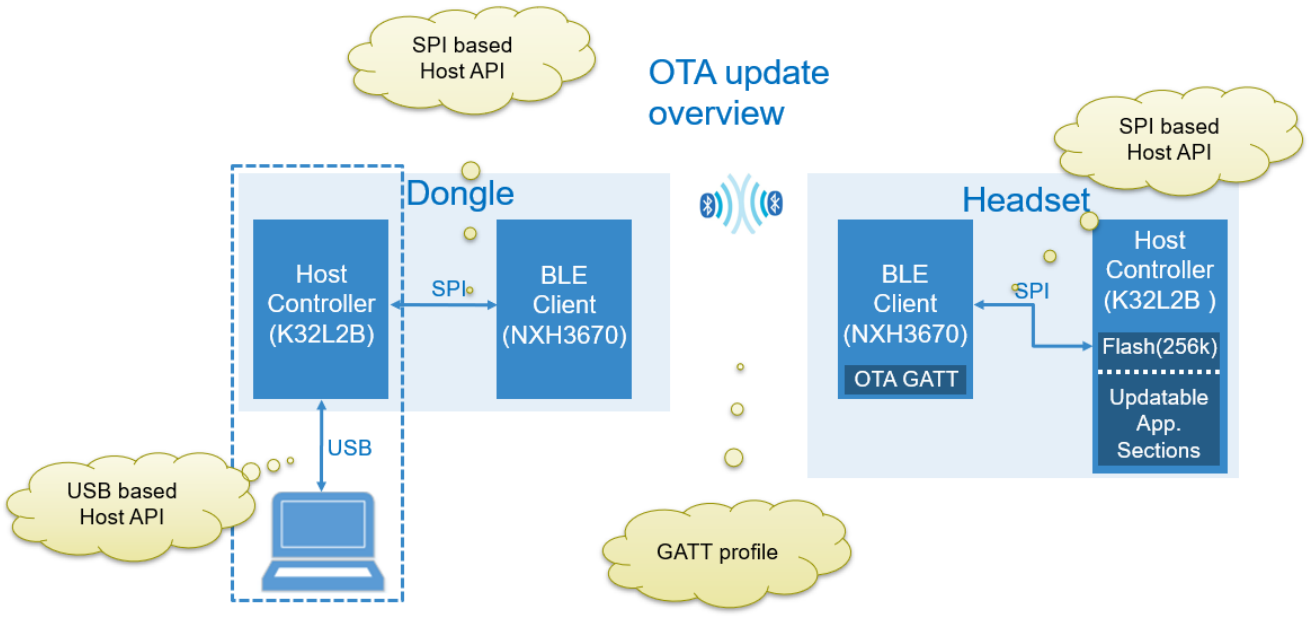

Figure 1. OTA setup

The setup requires a Dongle, Headset, PC and a USB cable connecting the Dongle and PC.

A typical scenario based on **K32L2B + NxH3670** is described as below:

1. Dongle and Headset are initially programmed over the USB interface.
2. Dongle and Headset are paired.
   - Pairing Data (PD) of Headset is independent on the PD of Dongle.

     Once booted up, the Headset retrieves PD from its own Memory written in advance and will not store extra PD.
   - PD of Dongle is dependent on the PD of Headset.

     Once booted up, the Dongle retrieves PD and pair with Headset, and then stores the Headset device information in Dongle's PD section (eg. Partition3: Device Info & Bonding Data).
3. Dongle is re-flashed with the `OTA_Dongle` application, which is the start of the OTA process.
   - `OTA_Dongle` can be used as VCOM to transfer data between PC and K32L2B through the USB.
   - Before re-flashing operation, make sure that PD is not erased.(Dongle is responsible for getting Headset's device information and `OTA_Dongle` is not). Two NXH devices are paired firstly and then connected, so they can't be connected if the user has erased PD date stored in Dongle's Flash.
   - In the Debug mode, the code takes up much space. The Headset board does not have eough spaces to store the binary data of Headset and `OTA_Headset` at the same time. So users can re-flash the Headset board with the `OTA_Headset` application to test the OTA function. User may need choose **Release mode** to make sure Flash has enough spaces to store firmware.
4. OTA process is triggered by PC application.
5. OTA finishes, firmware of Headset is updated.
6. Re-flash the Dongle with Gaming application.

2.2 Second Storage Bootloader (SSB)

The SSB is automatically bootstrapped by the (ROM) first stage bootloader.

You can store multiple firmware in flash according to your requirements and inform SSB which firmware to boot.

For example, considering Headset board has no USB port in the **K32L2B Bluetooth LE Audio System**, developers store at least three firmware in advance, including:

- **SSB**: To decide which firmware to boot.
- **OTA firmware**: To receive new firmware.
- **Application firmware**: An actual application, including specific Headset functions.

Taking the current demos as an example, SSB functions include:

1. Set the VTOR to the application vector table address.
2. Set stack pointers to the application stack pointer.
3. Jump to the application (PC now point to application).

NOTE

The current design uses NXH3670 to transfer data over the air and program K32L2B via the SPI interface assisted by the SSB code.

2.3 Partition table

The **Dongle**, **OTA_Dongle**, **Headset** and **OTA_Headset** applications and their locations are required to be mapped in Flash. This mapping is present in the Partition Table stored at a fixed offset in the Flash memory of the Host controller. For more information, refer to HAPI OTA.

Figure 2 shows the partitions and offsets.

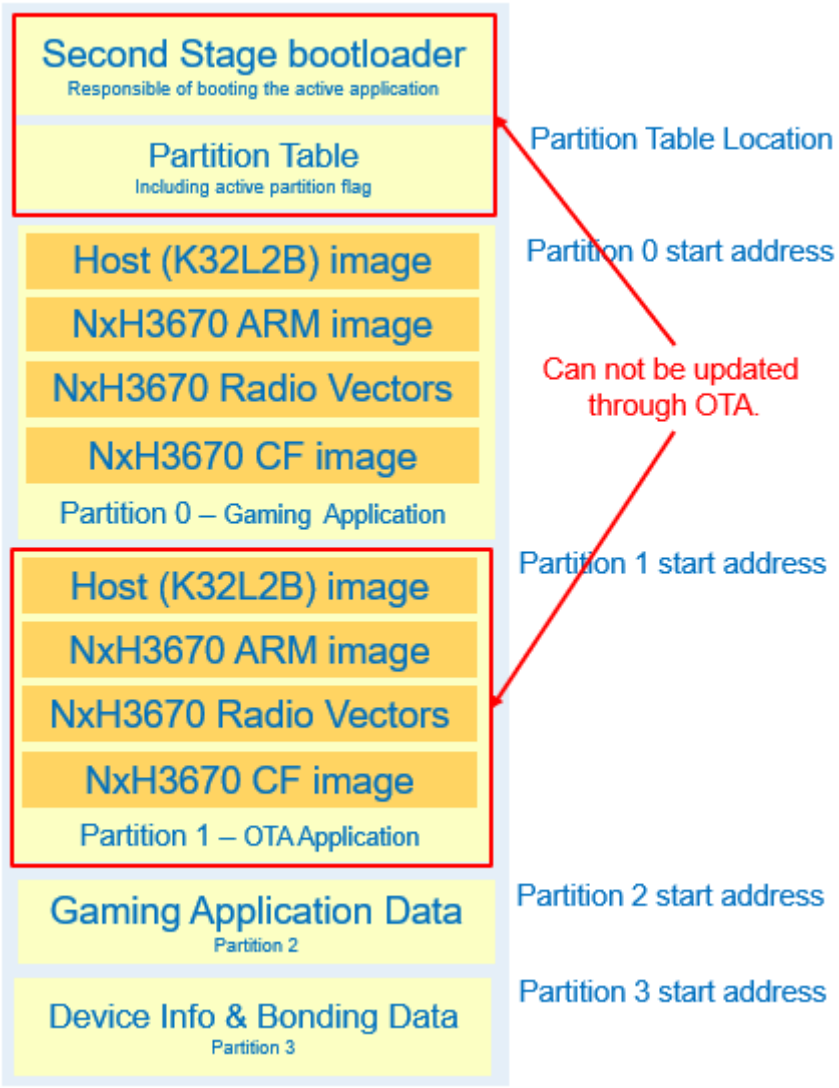

Figure 2. K32L2B Flash Layout example of a K32L2B Headset

- **Partition 0** is the Gaming Application. It contains the firmware for the K32L2B Host Controller firmware, the NxH3670 ARM image, the NxH3670 Audio Radio Vectors, and NxH3670 CoolFlux image.
- **Partition 1** is the OTA application. It contains the Host Controller (K32L2B) firmware and NxH3670 ARM Image.

  For OTA_Dongle, user only need to Flash `ota_app` and `NxH_Binary` (`ARM.phOtaDongle.ihex.eep`).

  — rfmac (`rfmac.eep`) is added already in to **Nxh** image,

  — CF (`phStereoInterleavedAsrcTx.eep`) is not required/used.
- **Partition 2** contains Application data. As general-purpose data storage for the Gaming application, it is currently unused.
- **Partition 3** contains the Device info and Bonding data. Device information contains Bluetooth LE specific attributes that need to be present for the air interface to work. Bonding data makes sure that Dongle and Headset automatically reconnect. Bonding data is only relevant for the Dongle.

3 Using Flashtool

This document lists the operation steps of how to use `.BAT` to update firmware easily and quickly. For more information of Flashtool, refer to the HAPI OTA and tools sections in `NXH3670_SDK_Gaming_G3.0`.

3.1 Modification introduction

1. `ota_update_headset.bat`

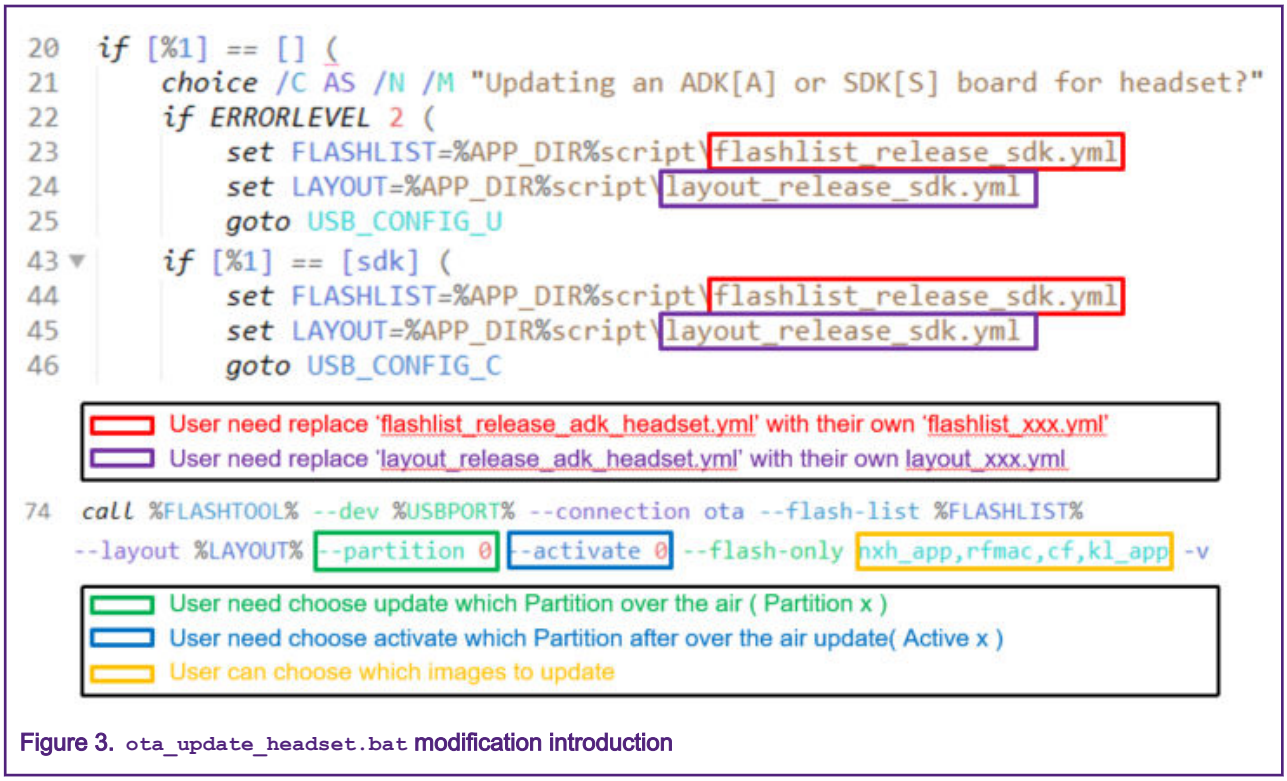

Figure 3. `ota_update_headset.bat` modification introduction

With JLink, perform the following steps to convert `.yml` of Partition table to `.BIN` that will be downloaded to Flash.

a. Open a command line interface

b. Go to your NXH3670_SDK_Gaming_G3.0 folder

c. Run `flash_scripts\flashtool.cmd` -> `dev table.bin` -> `connection export` -> `layout kinetis_democode\apps\kl_dongle\script\layout_debug_sdk.yml`

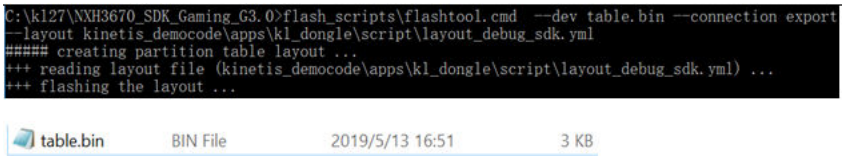

Figure 4. Convert `.yml` to `bin`

However, first 2560 (`0xA00`) bytes of this `table.bin` are all `0x00`. This document introduces two methods to handle it.

- Make sure that `table.bin` is flashed before flashing the SSB located in `0x00`. Otherwise, the `table.bin` will overwrite the SSB. So for OTA, user have to port the `kl_ssb` application or flash SSB file as well.
- Or, you can delete 2560 (`0xA00`) bytes of this `table.bin` and then download the changed `table.bin` to Partition table address.(In our software, we put it to `0x3f400`).

2. `flashlist_release_sdk.yml`

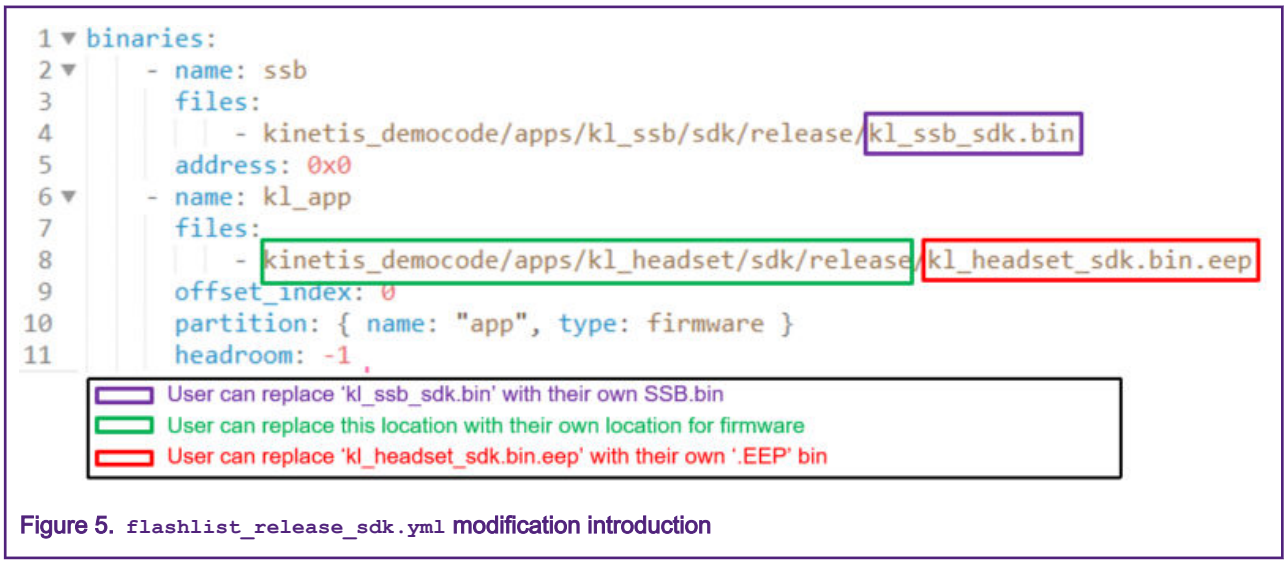

Figure 5. `flashlist_release_sdk.yml` modification introduction

`flashlist_release_sdk.yml` (`kl_headset`) lists the binaries and `offset_index` of Partition to be used to operate OTA. In this example, the user want update 'kl_headset_sdk.bin.eep' to `offset_index_0` of the current Partition.

3. `layout_release_sdk.yml`

```
# partition_id 0
- name: "app"
  type: firmware
  base_address: 0xbf0
  size: 0x2a000 # 168 KB
  offsets:
      - 0x0          # kl_app | 52 KB (21 KB free)
      - 0xd010       # nxh_app | 65 KB (4 KB free)
      - 0x1d410      # rfmac | 18 KB (2 KB free)
      - 0x21c10      # cf | 32 KB (2 KB free)
# partition_id 1
- name: "ota"
  type: firmware
  base_address: 0x2abf0
  size: 0x14c00 # 83 KB
  offsets:
      - 0x0          # kl_ota_app | 24 KB
      - 0x6010       # nxh_ota_app | 58 KB
```

If user want put OTA related code in Partition0, they can replace 'app' with 'ota'

Partition0's base_address

The offsets of Images in Partition0 is based on base_address.

Partition1's base_address should be equal to or bigger than the sum of Partition0's address and size.

Figure 6. `layout_release_sdk.yml` modification introduction

You can design your own `layout_release_sdk.yml` to meet the use of Flash. In the software design, MCU reads `NXH_Binaries` from specified location and then transfer data to NXH3670 through the SPI interface. Make sure the design of Flash layout is correct.

Perform the following step to convert `.BIN` to `.EEP` that will be used in OTA process.

- Use `…\tools\to_eep.cmd` by inputting `-i XXX.bin -o XXX.bin.eep`

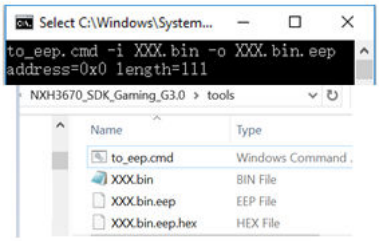

Figure 7. Converting `.BIN` to `.EEP`

3.2 Test process

When users change `ota_update_headset.bat`, `flashlist_release_sdk.yml` and `layout_release_sdk.yml` correctly, OTA can work.

Assuming that **Dongle** and **Headset** have already paired successfully, follow the steps as below.

1. Download `OTA_Dongle` and make sure that the PC can recognize it as a **USB Serial Device**.

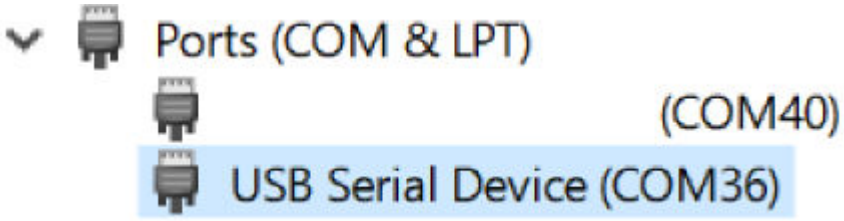

Figure 8. VCOM USB serial device

   As shown in Figure 8, PC recognize it as **COM36**.

2. This document list two cases:

   a. **Case 1**: Users are running `app` instead of `ota_app`.

   - In the **Release mode**, `Active_flag` of `APP_Partition` is **0** (Partition 0 – Gaming Application) currently, which indicates that users need send command to switch `Active_Partition` from **0** to **1** (Partition 1 – OTA Application).

```
##### flashing binaries ...
+++ reading flash list file ...
+++ checking flash list file ...
+++ connecting to the remote device ...
+++ switching remote active partition to OTA partition ...
+++ rebooting remote to OTA partition ...
+++ connecting to the remote device ...
+++ getting remote table version ...
```

Figure 9. Need Switch operation

   - If users use `phOtaHeadset.ihex.eep` instead of `phGamingRx.ihex.eep` in the **app** case, it means they have boot and start NXH3670 as OTA function (users can use OTA-related tool to communicate with Dongle board with firmware `phOtaHeadset.ihex.eep`) and do not need switch remote `Active_Partition` actually. However, the `hci_table` of **app** do not have OTA related code, so it cannot be used to operate OTA .

   b. **Case 2**: Users are running `ota_app` and `NXH_Binary` is `phOtaHeadset.ihex.eep`.

   - In the **Debug mode**, `Active_flag` of `OTA_Partition` is **1** (Partition 1 – OTA Application) currently, whcih indicates that the code is ready for OTA process and do not need switch remote `Active_Partition`.

   The following example assumes that user is using Case 1.

   Open a command line interface and go to the `flash_scripts` folder. Input:

   - `ota_demo_sdk.bat` (Uses may change it and rename it)
   - board (this document uses the SDK board to test, so input **S**)
   - USB port name (**COM36**).

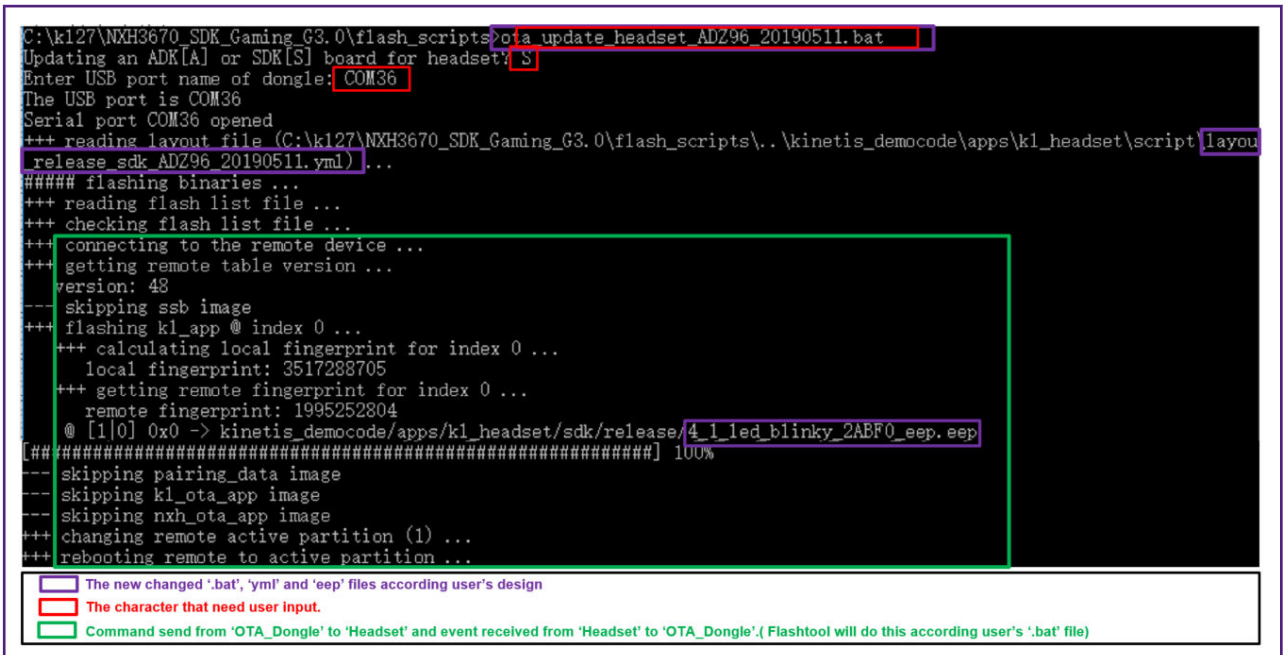

Figure 10. CMD of OTA process

As shown in Figure 10, **[##..##] 100%** indicates the update progress.

3. LOG information

   To view OTA progress better, users can download `OTA_Headset_Debug_mode` code that can provide the LOG information.

Figure 11. `OTA_Headset` WriteToPartition event

4 Software design

In order to describe the software design clearly, some programs are attached for user as reference.

4.1 Code of SSB

```
enum _vector_table_entries { kInitialSP = 0,kInitialPC };

uint32_t *appVectorTable = NULL;
uint32_t applicationAddress = 0;
uint32_t stackPointer = 0;

appVectorTable = (uint32_t *)(entry.startAddress + entry.imageOffsets[0] +
NVMMGR_EEP_INITIAL_HEADER_SIZE);
applicationAddress = appVectorTable[kInitialPC];
stackPointer = appVectorTable[kInitialSP];
```

```
JumpToApplication(applicationAddress, stackPointer);
```

```
void JumpToApplication(uint32_t applicationAddress, uint32_t stackPointer)
{
    /* Static variables are needed as we need to ensure the values we are using are not stored on the
previous stack */
    static uint32_t s_stackPointer = 0;
    s_stackPointer = stackPointer;
    static void (*farewellBootloader)(void) = 0;
    farewellBootloader = (void (*)(void))applicationAddress;

    /* Set the VTOR to the application vector table address */
    SCB->VTOR = applicationAddress;

    /* Set stack pointers to the application stack pointer */
    __set_MSP(s_stackPointer);
    __set_PSP(s_stackPointer);

    /* Jump to the application */
    farewellBootloader();
}
```

```
bootValidApp    PROC
            EXPORT  bootValidApp
    LDR    r1, [r0, #0]    ; Get app stack pointer
    MOV     sp,r1        ;
    LDR    r1, [r0, #4]    ; Get app reset vector
        BX    r1        ; PC now point to App_Firmware
    ENDP
```

4.2 Code of OTA receive

To let users understand the OTA receive process easily, this section provides an **event handler** in the `OTA_Headset` code: `HCI_VS_WRITE_TO_PARTITION_SUB_EVENT`, to introduce how to write firmware to Flash.

Assuming Dongle board is running the **OTA_Dongle** demo and Headset board is running **OTA_Heatset** demo, the NXH3670 of Headset can receive event from Dongle and transmit event to Host Controller (K32L2B) through the SPI interface.

1. The NXH3670 of Headset receive `HCI_VS_WRITE_TO_PARTITION_SUB_EVENT` (`0Xe1`) sent from the NXH3670 of Dongle, then will run `HCI_EvtWriteToPartitionHandler`.

   ```
       {
           .evtCode = HCI_VS_EVENT_CODE,
           .subEvtCode = HCI_VS_WRITE_TO_PARTITION_SUB_EVENT,
           .evtHandler = HCI_EvtWriteToPartitionHandler,
           .evtParmsLen = HCI_UNDEFINED_PARAMETER_LENGTH,
        },
   ```

2. Write the data to the requested partition with offset. This document lists some APIs as below:
   - Users need to write outside of the current cached sector, so all data need to be copied in the current sector.

     ```
     ReadFromFlash(s_Context.cacheBuf, SECTOR_SIZE_IN_BYTES, s_Context.cachedSectorAddr)
     ```

- Users can modify Cache with the data sent from NXH3670 of Dongle board.

```
memcpy(&s_Context.cacheBuf[cacheOffset], data, cpyLen);
```

- When the data of one packet is copied to `cacheBuf`, users can program Sector by using Flash write API.

```
ProgramSector(s_Context.cacheBuf, SECTOR_SIZE_IN_BYTES, s_Context.cachedSectorAddr);
```

3. The operation is successful to NXH3670 with a command as below:

```
HCI_CmdDataWritenToPartition(&req);
HCI_SendCmdBlocking(&req)
```

4.3 Code of OTA send

To let user understand the OTA send process easily, this section uses Pseudo code to introduce how **OTA_Dongle** sends firmware to **OTA_Headset**.

```
void UsbVcomDataReceived_Cb(uint32_t length, uint8_t *data)
{
    switch (opcode) {
        case HCI_CMD_VS_CONNECT_OPCODE: {
            …
            HCI_CmdPrepareConnect(&hciReq, &connectParams);
            ….
            break;}

        default: {
            ….
            HCI_CmdPrepareHostGenericCmd(&hciReq, data, length);
            ….
            break;}
    }
```

NOTE

- Case `HCI_CMD_VS_CONNECT_OPCODE`

  This CMD means that the **OTA_Dongle** wants to connect **OTA_Headset**.

- Default CMD

  OTA_Dongle will send any other CMD to **OTA_Headset** by using NXH3670. Actually, the MCU of **OTA_Headset** is responsible for writing these data to Flash.

5 Conclusion

Users can implement program update by using the Flashtool and files in `NXH3670_SDK_Gaming_G3.0`. Changes may be required on design needs. The firmware update speed via OTA is about 1 KB per second.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 01/2020

Document identifier: AN12649

arm