1 Introduction

1.1 Overview

The USB interface in the Dongle is used to connect to a PC. It is responsible for creating a wireless audio link with Headset. The main functions include:

- **Send**: To transmit audio stream from PC to Headset.
- **Receive**: To receive control signal and voice audio from Headset to PC.
- **OTA**: To be used as VCOM device to transfer firmware file from PC to Headset that runs the \texttt{OTA\_Headset} firmware at the same time.

To give the audience a systematic view of Dongle in the \textbf{LPC54114 BLE Audio System}, this document describes the hardware design and software architecture (top level design).

1.2 Reference documents

Table 1. References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LPC BLE Audio System]</td>
<td>Introduction to \textbf{LPC54114 BLE Audio System}</td>
</tr>
<tr>
<td>[LPC Headset]</td>
<td>LPC54114 USB Headset with NxH3670</td>
</tr>
<tr>
<td>[LPC OTA]</td>
<td>OTA operation steps of \textbf{LPC54114 BLE Audio System}</td>
</tr>
</tbody>
</table>

2 System overview

2.1 Block diagram

The demo board is designed to support the Dongle and the Headset configurations.
As shown in Figure 1, SPI CONFIG, I\textsubscript{2}C CONFIG, and I\textsubscript{2}S CONFIG refer to the master selections of the communication interface. For example, I\textsubscript{2}C CONFIG can select LPC54114 as I\textsubscript{2}S master in Dongle design while WM8904 as I\textsubscript{2}S master in Headset design.

The block diagram of LPC54114_Dongle is as shown in Figure 2.
As shown in Figure 2, the system contains:

- a Host Controller (LPC54114) that can run Dongle and OTA_Dongle demos.
- an NxA3670 that communicates with LPC54114 through SPI interface and transfers audio stream through the I²S interface.

### 2.2 Software architecture of USB Dongle

The software structure is as shown in Figure 3 and Figure 4.
Figure 3. Application framework architecture

Figure 4. Dongle application architecture
The architecture contains NVM service, USB service, Audio service, NxH service, and User Interface (UI) service. The main functions include:

- NVM service: To read Partition Table.
- NxH control: To boot, start and transfer data with LPC54114 using SPI interface.
- User Interface: Buttons used to control the volume, play and pause.
- Audio service: To transmit audio data to I2S interface.

**NOTE**
Without software intervention, the audio data can move from the ring buffer directly to the I2S interface using the DMA channel.

- USB controller: USB is configured as an audio interface (UAC).

Figure 5 shows the audio transfer process.

---

**Figure 5. Audio path**

- → : Playback (forward channel): The audio path from PC to the Headset.
  - The USB host controller stack creates an interrupt when the transfer is completed. The software copies the transfer data from the USB stack in a ring buffer, the input-buffer. This buffer queues audio data until a fraction of its buffer capacity is filled, such as , 50%.
  - The software enables a DMA channel connected to the I2S interface of the Host Controller. Then, the audio data can move from the ring buffer directly to the I2S interface without software intervention.
  - The BLE controller is connected with host controller via I2S. In turn, the I2S data is transferred over-the-air to the Headset’s BLE controller.
  - The BLE controller of Headset is connected with CODEC through I2S. In turn, the received I2S data is transferred to CODEC without software intervention.
- ← : Record (backward channel): The audio path from the Headset to the PC.
  - The audio is entered through LINE IN or DMIC interface to CODEC. CODEC is connected with the BLE controller via I2S. In turn, the voice data is transferred to the BLE controller without software intervention (CODEC is I2S master device).
— DMA channel transfers the received audio data from I2S-FIFORD (FIFO read data register) to a ring buffer.
— The software copies the transfer data from a ring buffer to the USB stack.

This document introduces the audio transfer process in the Dongle. For more information of Headset, refer to LPC54114 USB Headset with NxH3670.

3 Components of USB Dongle

3.1 LPC54114

3.1.1 Host controller

The main features of LPC54114 USB Headset with NXH3670 are as below.

The LPC5411x is an Arm® Cortex®-M4 based microcontroller for embedded applications. These devices include:

- an optional Arm Cortex-M0+ coprocessor
- up to 192 KB of on-chip SRAM
- up to 256 KB on-chip flash
- Full-speed USB device interface
- a DMIC subsystem with dual-channel PDM microphone interface and I2S
- one 24-bit Multi-Rate Timer (MRT)
- eight flexible serial communication peripherals (each of which can be a USART, SPIs, or I2C interface)

3.1.2 Clocks

1. One reference crystal used on the board:
   - 32 MHz crystal connected with the NxH3670.
2. I2S related clocks are as below:
   - The bit clock of the I2S interface is derived from the master clock. The clock division is 16. bclk = mclk/16 = 24.576Mhz/16 = 1.536 Mhz.
   - The Word Select (WS)/Left-Right Clock (LRCK) is 48 KHz.

After I2S peripheral is correctly configured, the clock information is as shown in Figure 6.

Figure 6. Clock information in audio transfer process

3.1.3 Pins connection information

Table 2 lists the connection information of LPC54114 and other components.
Table 2. Connection information

<table>
<thead>
<tr>
<th>Function</th>
<th>Jumper (LPC54114 Dongle)</th>
<th>Name</th>
<th>Jumper (NxH3670)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S (Connect with MCU)</td>
<td>J1_10 (PIN P1.7)</td>
<td>I2S1_SDI</td>
<td>J12_1 (I2S_CONFIG)</td>
<td>BLE_SDO</td>
</tr>
<tr>
<td></td>
<td>J1_20 (PIN P0.5)</td>
<td>I2S0_SDO</td>
<td>J12_3 (I2S_CONFIG)</td>
<td>BLE_SDI</td>
</tr>
<tr>
<td></td>
<td>J1_18 (PIN P0.6)</td>
<td>I2S0_WS</td>
<td>J12_5 (I2S_CONFIG)</td>
<td>BLE_WS</td>
</tr>
<tr>
<td></td>
<td>J1_12 (PIN P1.8)</td>
<td>I2S1_WS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>J1_16 (PIN P0.7)</td>
<td>I2S0_SCK</td>
<td>J12_7 (I2S_CONFIG)</td>
<td>BLE_SCK</td>
</tr>
<tr>
<td></td>
<td>J2_ 9 (PIN P1.12)</td>
<td>I2S1_SCK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NXH Handshake</td>
<td>J2_18 (PIN P1.4)</td>
<td>BLE_SPIS_INTN</td>
<td>J16_9 (BLE_SPI)</td>
<td>SWM4 (- INTN)</td>
</tr>
<tr>
<td></td>
<td>J2_20 (PIN P1.3)</td>
<td>BLE_SPIS_SRQ</td>
<td>J16_11 (BLE_SPI)</td>
<td>SRQ</td>
</tr>
<tr>
<td>SPI</td>
<td>J4_3 (PIN P0.13)</td>
<td>BLE_SPIS_MISO</td>
<td>J16_1 (BLE_SPI)</td>
<td>SW0</td>
</tr>
<tr>
<td></td>
<td>J4_2 (PIN P0.12)</td>
<td>BLE_SPIS_MOSI</td>
<td>J16_3 (BLE_SPI)</td>
<td>SW1</td>
</tr>
<tr>
<td></td>
<td>J4_4 (PIN P0.11)</td>
<td>BLE_SPIS_SCLK</td>
<td>J16_5 (BLE_SPI)</td>
<td>SW2</td>
</tr>
<tr>
<td></td>
<td>J4_7 (PIN P0.4)</td>
<td>BLE_SPIS_SSN</td>
<td>J16_7 (BLE_SPI)</td>
<td>SW3</td>
</tr>
<tr>
<td>NXH Reset</td>
<td>J4_8 (PIN P0.22)</td>
<td>BLE_RESETN</td>
<td>J20_ (BLE_SWD)</td>
<td>POR_RESETN</td>
</tr>
</tbody>
</table>

Figure 7 shows the current demo, using LPCXpresso54114 board and NxH3670 board with extra jumper cables.
3.1.4 Schematic

- Audio transfer
  - I²S

![Figure 8. I²S interface connection](image)

In the Dongle, the host controller (LPC54114) transfers data directly to NxH3670 via I²S, so to configure CODEC, the I²S peripheral is required to be initialized instead of I²C peripheral.

- NxH3670
  - NxH handshake

![Figure 9. Handshake pins (SRQ and INTN) connection](image)

- SPI
3.1.5 Pins configurations

- **SPI**
  - Interface: SPI3.
  - Pins: CS (P0.4), SCK (P0.11), MISO (P0.12), MOSI (P0.13)
  - Polarity: Active-high SPI clock (idles low).
  - Phase: First edge on SPSCK occurs at the middle of the first cycle of a data transfer.
  - Baud rate: Configured to be 8000000u for SPI.

- **I^2S pin**
  - I^2S_TXD (P0.5)
  - I^2S_RXD (P1.13)
  - I^2S_BCLK (P0.7): SCLK
  - I^2S_FS (P0.6): WS/LRCK

- **NxH3670 Pin**
  - INIT (P1.4): Configured as digital input.
  - SRQ (P1.3): Configured as digital output.
— POR (P0.22): Configured as digital output.

3.2 NxH3670

Figure 12 shows the process of connection between Dongle and Headset.

1. Downloading and starting NxH3670 images
   • In the **Boot** step, the host controller loads images from flash/eeprom to NxH3670 through the SPI interface.
   • In the **starting** step, the host controller handshakes with NxH3670. Then the software registers an event table with the HCI layer used to handle events sent from NxH3670.

2. Pairing
   • The NxH3670 on Dongle and Headset board pair with each other. For example, Dongle retrieves PD from Headset.

3. Connecting
   When the NxH3670 on Dongle and Headset board successfully connect with each other, they start to transfer data to each other. For example, Dongle sends an audio stream to Headset.
3.2.1 Boot

3.2.1.1 NxH3670UK bootloader

The most important task of the bootloader is to prepare the NxH3670UK to start a user application. The typical bootloader lifecycle is:

1. Configure the device.
2. Load the memories. By default, the NxH3670UK starts up in the host-assisted mode with the SPI slave interface.
3. Enter the active mode.

3.2.1.2 Partition table

Since the NxH3670 BLE radio cannot store data persistent, the Flash memory of the Host Controller is used for storage.
The reference application can split the memory into logical partitions.

The data, either firmware binary data or application configuration data within such a partition, can be read or written.

As shown in Figure 13, the partition_id 0 contains four images, kl_app, nxh_app, rfmac and cf. For example, the offset of nxh_app is 0x20810, which indicates to download the images to 0x21400 (0xbf0+0x20810).
Users can design their own partition table on their needs, but the following should be mentioned.

- To keep Partition1: app_data to be the first partition in the memory, in the layout_release_adk.yml file, put in the order of app_data, app, ....
- Make sure that the value (base_address of Partition + size of partition0) is smaller than base_address of the Partition1.

3.2.1.3 NVM

Non-Volatile Memory (NVM) is a memory technology that maintains stored data during power-off. The flash array is an NVM using NOR-type flash memory technology.

The NVM of LPC54114 can be used to save firmware of NxH3670. Taking Dongle as an example, users need to pre-store phGamingTx.ihex.eep, phStereoInterleavedAsrcTx.eep, and rfmac.eep in the NVM, which will take up about 120 k.

3.2.1.4 EEP

3.2.1.4.1 Definition

For safety reasons, the NVM-image contains CRCs and signatures at different levels (i.e. block level and overall). This allows the detection of corrupt images and the abortion of the loading and the execution of potentially harmful instructions.

![Figure 14. Format of EEP file](image)

All fields listed in Figure 14 are stored in the little-endian format. A valid image must start with a 32-bit signature, 0xBEBAFECA. After this signature, one or more images can be present. Each image starts with a header.
In Figure 15, the value of Image Length is 16287 and of Type ID is 0. It indicates that the host controller sends 65148 (16287 × 4) bytes to NxH3670 through SPI.

In Figure 16, the value of Image Length is 284 and of Type ID is 0. It indicates that the host controller sends 1136 (284 × 4) bytes to NxH3670 through SPI.

3.2.1.4.2 Downloading .EEF file to LPC54114

This document provides two methods to download .EEF files to LPC54114.

1. Transform .EEP files to .HEX buffer.
   - Winhex
     - __attribute__((section(".ARM.__at_address")))
   
   With this method, users can store NxH3670 relevant firmware as a buffer, as OTA process only re-writes the application of the host controller instead of NxH3670.

2. Transform .BIN files to .EEP files.
   - SDK packet contains a tool called to_eep.cmd in release, as shown in Figure 17.
Figure 17. to_eep.cmd in packet

- **Input:** `to_eep.cmd -i spi_dma_b2b_transfer_master.bin -o spi_dma_b2b_transfer_master.eep`

Figure 18. Command string in to_eep.cmd

- As shown in Figure 19 and Figure 20, the `bin` file was packet with SIGNATURE and HEADER.

Figure 19. BIN file

Figure 20. EEPROM file

With this method, users can store the application of the host controller instead of NxH3670 firmware. For example, user can transfer the `.BIN` file of application to `.EEP.BIN` file with CRCs and signatures, which is useful in OTA process.

### 3.2.1.5 NxH3670 host interface: SPI

#### 3.2.1.5.1 SPI bus

For NxH3670, the boot loader configures the SPI slave interface and assumes the host to be SPI master. The configurations of SPI slave operation mode include:

- SPI slave 4-wire mode connection: MOSI, MISO, SCK, SSEL
- SPI slave max speed communication: 8 MHz
- SPI slave mode: `mode0` (CPHA=0, CPOL=0)
- Operating modes: clock and phase selection

SPI interfaces allow configuration of clock phase and polarity. These are sometimes referred to as numbered SPI modes, as described in Table 3 and shown in Figure 21. CPOL and CPHA are configured by bits in the CFG register (LPC54114).
### Table 3. SPI mode summary

<table>
<thead>
<tr>
<th>CPOL</th>
<th>CPHA</th>
<th>SPI mode</th>
<th>Description</th>
<th>SCK rest state</th>
<th>SCK data change edge</th>
<th>SCK data sample edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>The SPI captures serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is changed on the following edge.</td>
<td>Low</td>
<td>Falling</td>
<td>Rising</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>The SPI changes serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is captured on the following edge.</td>
<td>Low</td>
<td>Rising</td>
<td>Falling</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>Same as mode 0 with SCK inverted.</td>
<td>High</td>
<td>Rising</td>
<td>Falling</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>Same as mode 1 with SCK inverted.</td>
<td>High</td>
<td>Falling</td>
<td>Rising</td>
</tr>
</tbody>
</table>

![Diagram](image)

**Figure 21.** Basic SPI operating mode: mode0

The SPI captures serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is changed on the following edge.

![Diagram](image)

**Figure 22.** mode0 example of logic analyzer

### 3.2.1.5.2 SPI flow control

In the BLE Audio System, the format of SPI transfer is as shown in Figure 23.
Table 4. SPI slave – Supported commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>0b010xxxxx</td>
<td>To write payload to NxH3670 SPI slave.</td>
</tr>
<tr>
<td>Read</td>
<td>0b110xxxxx</td>
<td>To read pending data from NxH3670 SPI slave.</td>
</tr>
<tr>
<td>Read status</td>
<td>0b101xxxxx</td>
<td>To read status byte.</td>
</tr>
<tr>
<td>Read extended status</td>
<td>0b111xxxxx</td>
<td>To read extended status byte.</td>
</tr>
</tbody>
</table>

For example, in the software design, `#define SPI_WRITE_CMD (0x40u)` is used to define the `Write` command. Figure 24 shows the signal of Logic analyzer.

3.2.1.6 Host Controller Interface (HCI)

3.2.1.6.1 HCI command format

The HCI command is embedded in the SPI payload field of the SPI write command (see SPI flow control). The beginning of the HCI command must be aligned to the beginning of the SPI transfer.

All commands and events are formatted as Bluetooth HCI Vendor Specific commands.
Figure 25. HCI command format

A command starts with:

- The command packet byte with a fixed value, \(0x01\).
- A unique 16-bit opcode which identifies the HCI command.
- Parameter Length holds the length of the parameters that follow (in byte). Zero value is allowed.
- The actual parameters (optional). These actual parameters include 8-bit, 16-bit, 24-bit, and so on. It is the command processor that must interpret the byte sequence correctly.

In the commands, HCI opcodes and optional parameters are always LSB first.

The SPI slave - transfer format of logic analyzer is as shown in Figure 26.

Figure 26. SPI slave - Transfer format of logic analyzer

3.2.1.6.2 HCI event format

Results of commands are sent back as HCI formatted events. Whenever the HCI controller must send something back to the host, it queues this event and the host retrieves this queued event.

The HCI event is embedded in the SPI payload field of the SPI read command.

Figure 27. HCI event format

The HCI event format of logic analyzer is as shown in Figure 28.
3.2.1.6.3 HCI command transfer

Figure 29 shows a sequence of how an HCI command is sent to the NxH3670.

Before starting an SPI transfer, assert the SRQ line and wait for the confirmation on the awake/int signal. It is to check whether the NxH3670 is awake and the SPI bus is available.

Although the host knows the NxH3670 is awake, it is required to use the SPI read-status-command to check whether the NxH3670 is ready to accept new SPI data.

The signal changes, including SRQ, Awake/Int, CS, MOSI and MISO, are as shown in Figure 31.
3.2.1.6.4 HCI event transfer

The NxH3670 uses a software queue to store multiple HCI events. If the SPI read buffer is empty, the earliest event is moved to the SPI read buffer and the SPI pending data signal is asserted.

Figure 32. HCI event transfer - Read extended status

The NxH3670 indicates pending data by asserting the awake/int signal. The host can retrieve the extended status to check how many data is pending. The SRQ signal is de-asserted and the awake/int is de-asserted soon after. As the actual pending data has not been read, the awake/int signal is asserted again.

Figure 33. HCI event transfer - Read

The host knows how many bytes are pending and can initiate an SPI read command. The NxH3670 sends the serialized HCI event back to the host.

The NxH3670 transfers only one HCI event at a time. If more HCI events are pending in the software queue, it moves the earliest HCI event to the SPI buffer and starts the sequence as described above.

If the host does not read fast enough, HCI events may get lost due to buffer overflow.
3.2.2 Handshake

The SPI handshake protocol implements three logical signals by using two physical hardware signals.

3.2.2.1 Logical signals

1. Service request signal
   This signal is used by the host to request services by the NxH3670. When detecting the signal, the NxH3670 asserts the awake signal to indicate that it is ready to handle the service request.

2. Awake signal
   This signal is used by the NxH3670 to indicate it is awake. The NxH3670 asserts the awake signal only when the SRQ signal is asserted and the NxH3670 does not wake up every time.

3. Pending data signal
   When the NxH3670 has pending data, it asserts this signal to the host.

3.2.2.2 Physical signals

As shown in Figure 34 and described in Table 5, the three logical signals are mapped onto two physical signals to reduce required pin count.

![Figure 34. SPI handshake physical signals](image)

<table>
<thead>
<tr>
<th>Table 5. Physical to logical signal mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical signal</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Service request signal</td>
</tr>
<tr>
<td>Wake signal</td>
</tr>
<tr>
<td>Pending data signal</td>
</tr>
</tbody>
</table>

**NOTE**
The pending data signal maps to the INT physical signal.

The following scenarios may occur:

1. The host initiates an SPI transfer.
2. The NxH3670 requests an SPI transfer.
3. The host initiates and the NxH3670 request an SPI transfer simultaneously.

For an easier understanding about the process of handshake, Scenario 2 is taken as an example.

When the NxH3670 has pending data, it generates the sequence to report pending data, as shown in Figure 35.

1. The NxH3670 asserts the AWAKE/INT signal to indicate that it has pending data.
2. To retrieve the pending data, the host initiates an SPI transfer.

![Diagram of SRQ, Awake/Int, and SPI states]

*Figure 35. NxH3670 requested SPI transfer sequence*

The NxH3670 stays awake as long as data is pending. The host must read the pending data as soon as possible to save power.

![Diagram showing boot-up, handshake, and pairing stages]

*Figure 36. NxH3670 requested SPI transfer data*

3.2.3 Start

A USB cable can be used to connect J5 (LPCXpresso54114) with PC to power or download firmware.

4 Porting guide and demo introduction

4.1 Framework

Two SDK-based files are required to be modified, `framework_power` and `framework_timer.c`. Other files independent of SDK are not required be modified.
Figure 37 shows the different configurations between KL27 and LPC54114. Contents in red frame indicate the changed codes to be modified according to actual design requirements without functional change.

### Code Examples

#### Framework Timer

In `framework_timer.c`, LPC54114 uses Multi-Rate Timer (MRT) to provide timing (SW timer) for the system while KL27 uses Low-Power Timer (LPTMR) to implement the same functionality.

```c
static void StartHwTimer(uint32_t ticks)
{
    MRT_StartTimer(MRT0, kMRT_Channel_0, ticks);
}
```

```c
static void StartHwTimer(uint32_t ticks)
{
    LPTMR_SetTimerPeriod(LPTMR_INSTANCE, ticks);
    LPTMR_StartTimer(LPTMR_INSTANCE);
}
```

#### Tips

- Codes can be modified by following the original code instead of completely changed. For example:

<table>
<thead>
<tr>
<th>LPC54114</th>
<th>KL27</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static void StartHwTimer(uint32_t ticks)</code></td>
<td><code>static void StartHwTimer(uint32_t ticks)</code></td>
</tr>
<tr>
<td><code>{MRT_StartTimer(MRT0, kMRT_Channel_0, ticks);}</code></td>
<td><code>{LPTMR_SetTimerPeriod(LPTMR_INSTANCE, ticks);LPTMR_StartTimer(LPTMR_INSTANCE);}</code></td>
</tr>
</tbody>
</table>

### 4.2 NxH

Five MCU’s peripherals and SDK-based files are required to be modified, `nxh_boot.c`, `nxh_ctrl.c`, `transport_ctrl_common.c`, `transport_ctrl.c`, and `transport_spi.c`.

Figure 38 shows the configuration differences between KL27 and LPC54114.
In the NxH service, the followings need to be re-configured:

1. Digital output/input pin (\texttt{nxh\_boot.c}, \texttt{nxh\_ctrl.c}, \texttt{transport\_ctrl\_common.c}, and \texttt{transport\_ctrl.c})

   Configure \texttt{SRQ} and \texttt{POR\_RESET} pins as digital output function and \texttt{INTN} pin as input function, to complete the handshake and transfer processes. Then test the \texttt{boot-up} and \texttt{start} processes.

   - Reminder 1
     
     The \texttt{NxH3670's INTN} pin changes the level to inform MCU with status change during the process of \texttt{Boot}. This function can be configured on MCU design.
     
     For example, use the \texttt{Pseudo code} to introduce \texttt{ProgramSpiAwakeInt}.

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtrlCtx-&gt;currentIntConfig = intConfig;</td>
<td>CtrlCtx-&gt;currentIntConfig = intConfig;</td>
</tr>
<tr>
<td>PORT_SetPinInterruptConfig(XX, XX, intConfig);</td>
<td>/* Select pins &amp; polarity for GINT0 */ GINT_ConfigPins(XX, XX, XX, intConfig);</td>
</tr>
</tbody>
</table>

   - Reminder 2
     
     — KL27 uses \texttt{PORTC\_PORTD\_IRQHandler} to call \texttt{TRANSPORTCTRL\_IrqHandler}.
     
     — LPC54114 uses \texttt{GINT0\_Driver\_IRQHandler} to call \texttt{TRANSPORTCTRL\_IrqHandler}.  

Figure 38. NxH service porting
2. SPI (transport_spi.c)

As MCU communicates with NxH3670 through the SPI interface, the base address of SPI interface should be decided first. Then, configure MOSI, MISO, SCK, CS pins based on SDK.

3. DMA (transport_spi.c)

In Audio Service, use API SPI_MasterTransferDMA to transfer the audio stream through the DMA channel and API SPI_MasterTransferNonBlocking to communicate with NxH3670.

4.3 USB

Three MCU’s peripherals and SDK-based files are required to be modified, usb_device_dci.c, usb_osa_bm.c, and usb_ctrl.c. Figure 40 shows the middleware configuration difference between KL27 and LPC54114.

Figure 41 shows the service configuration difference between KL27 and LPC54114 (VBUS is required to be configured for USB peripheral).
Figure 41. BSP porting related to USB service

- Reminder 1 (usb_device_config.h and usb_ctrl.c)

  Modify USB configurations to port the demo from KL27 to LPC54114.

  → void USB0_IRQHandler(void)

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_DeviceKhciIsrFunction</td>
<td>USB_DeviceLpcIp3511IsrFunction</td>
</tr>
</tbody>
</table>

KL27: USB_mideleware of KL27
LPC54114: USB_mideleware of LPC
KL27: Clock
LPC54114: Clock & GPIO (VBUS)
--- #define CONTROLLER_ID XXXX

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>kUSB_ControllerKhci0</td>
<td>kUSB_ControllerLpcIp3511Fs0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
</tr>
<tr>
<td>irqNo</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>khciIrq[CONTROLLER_ID - kUSB_ControllerKhci0];</td>
<td>khciIrq[CONTROLLER_ID - kUSB_ControllerLpcIp3511Fs0];</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
</tr>
<tr>
<td>#define USB_DEVICE_CONFIG_KHCI (1U)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define USB_DEVICE_CONFIG_KHCI (1U)</td>
<td>#define USB_DEVICE_CONFIG_KHCI (0U)</td>
</tr>
<tr>
<td></td>
<td>#define USB_DEVICE_CONFIG_EHCI (0U)</td>
</tr>
<tr>
<td></td>
<td>#define USB_DEVICE_CONFIG_LPCIP3511FS (1U)</td>
</tr>
</tbody>
</table>

### 4.4 Audio

Three MCU’s peripherals and SDK-based files are required to be modified, `audio_rx.c`, `audio_tx.c`, and `dma_interface.c`. Other files in the `audio` are not required to be modified.

Figure 42 shows the configuration difference between KL27 and LPC54114.

---

Figure 42. Audio service porting

1. In `audio_tx.c`:
• KL27 uses DMA to transfer audio data from ring-buffer to Synchronous Audio Interface (SAI).
• LPC54114 uses DMA to transfer audio data from ring-buffer to FIFOWR Register of I²S.

2. In `audio_rx.c`:
• KL27 uses DMA to transfer audio data from Synchronous Audio Interface (SAI) to ring-buffer.
• LPC54114 uses DMA to transfer audio data from FIFORD Register of I²S to ring-buffer.

The I²S interface of LPC54114 is required to be configured to achieve the same function of KL27’s SAI.

3. In `dma_interface`:
• KL27 obtains `SourceAddress` and `DestinationAddress` using SDK API.

  — Reminder 1
  To liberate the task of MCU, KL27 uses a `LinkDMA` to re-configure `DMA_Channel`, which is used to transfer audio data. The related code is `audio_DMATxCallback`.

  The `LinkDMA` help to re-configure total number of transfers to be performed.
• LPC54114 do not provide such Register, so users can save `SourceAddress` and `DestinationAddress` as The global variable.

  — Reminder 2
  Configure LPC54114’s `XFERCFGn` register (16-25 bits) of total number of transfers to be performed, to make sure that DMA moves data continuously if ring-buffer is not empty.

4.5 NVM

One MCU’s peripherals and SDK-based file is required to be modified, `nvm_controller.c`. This file writes or reads Flash and is called by `nvm_mgr.c` and `nvm.c`.

Figure 43 shows the configuration difference between KL27 and LPC54114.

![Figure 43. Drive of NVM porting](image)

• Reminder 1
  The `NVM.c` service reads **Partition table** which contains the Flash location of different firmware. It is a necessary operation in Over The Air (OTA) firmware update process, so user must make sure configure their Flash correctly on LPC54114 to perform OTA or obtain the **Partition table** information.

The following lists items to be modified in the `nvm_controller.c` using pseudo codes.

1. Flash definition

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define SECTOR_SIZE_IN_BYTES (1024)</td>
<td>#define SECTOR_SIZE_IN_BYTES (32768)</td>
</tr>
<tr>
<td>#define FLASH_SIZE (0x40000)</td>
<td>#define FLASH_SIZE (0x40000)</td>
</tr>
<tr>
<td>KL27</td>
<td>LPC54114</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>#define ROUND_DOWN_TO_SECTOR_SIZE_MASK (0xFFFFFC00)</td>
<td>#define ROUND_DOWN_TO_SECTOR_SIZE_MASK (0xFFFF8000)</td>
</tr>
<tr>
<td>#define GET_OFFSET_IN_SECTOR_MASK (0x03FF)</td>
<td>#define GET_OFFSET_IN_SECTOR_MASK (0x7FFF)</td>
</tr>
<tr>
<td>#define NOTHING_CACHED (0xFFFFFFFF)</td>
<td>#define NOTHING_CACHED (0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

2. static void EraseSector (uint32_t addr)

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_Erase</td>
<td>FLASHIAP_PrepareSectorForWrite</td>
</tr>
<tr>
<td>FLASH_VerifyErase</td>
<td>FLASHIAP_EraseSector</td>
</tr>
<tr>
<td></td>
<td>FLASHIAP_BlankCheckSector</td>
</tr>
</tbody>
</table>

NOTE
One sector (32kb) is erased instead of one page (256 bytes).

3. static void ProgramSector (uint8_t *data, uint32_t len, and uint32_t addr)

<table>
<thead>
<tr>
<th>KL27</th>
<th>LPC54114</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_Program</td>
<td>FLASHIAP_PrepareSectorForWrite</td>
</tr>
<tr>
<td>FLASH_VerifyProgram</td>
<td>FLASHIAP_CopyRamToFlash</td>
</tr>
</tbody>
</table>

5 Conclusion

This document describes the hardware design and software architecture (top level design) of LPC54114_Dongle in the BLE Audio System. This document can be a reference for users to build their own demo.
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.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C-5, CodecTEST, CodecWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QoriQ, QoriQ Converge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybride, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKPro, μVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.