AN12647

K32L2B USB Dongle with NxH3670

Rev. 0 — 01/2020

Application Note

Contents

1 Introduction

1.1 Overview

The Dongle has a USB interface that connects to a PC. It is responsible for creating a wireless audio link with Headset. The functions include:

- **Send**: Transmit audio stream from PC to Headset.
- **Receive**: Receive control signal and voice audio from Headset to PC.
- **Over The Air (OTA)**: Transfer firmware files, as a VCOM device, from PC to Headset that run the `OTA_Headset` firmware at the same time.

To give the audience a systematic view of Dongle in the **K32L2B Bluetooth Low Energy (Bluetooth LE) Audio System**, this document describes the hardware design and software architecture (top-level design).

1.2 Reference documents

Table 1. References

Reference	Definition
K32L2B Bluetooth LE Audio System	Introduction to K32L2B Bluetooth LE audio system
K32L2B Headset	K32L2B Headset with NXH3670
K32L2B OTA	K32L2B Bluetooth LE Audio System OTA operation steps
K32L2B Emulating the I²S Bus	Emulating the I²S bus master with the FlexIO module

2 System overview

2.1 Block diagram

The demo board is designed to support the Dongle and the Headset configurations.

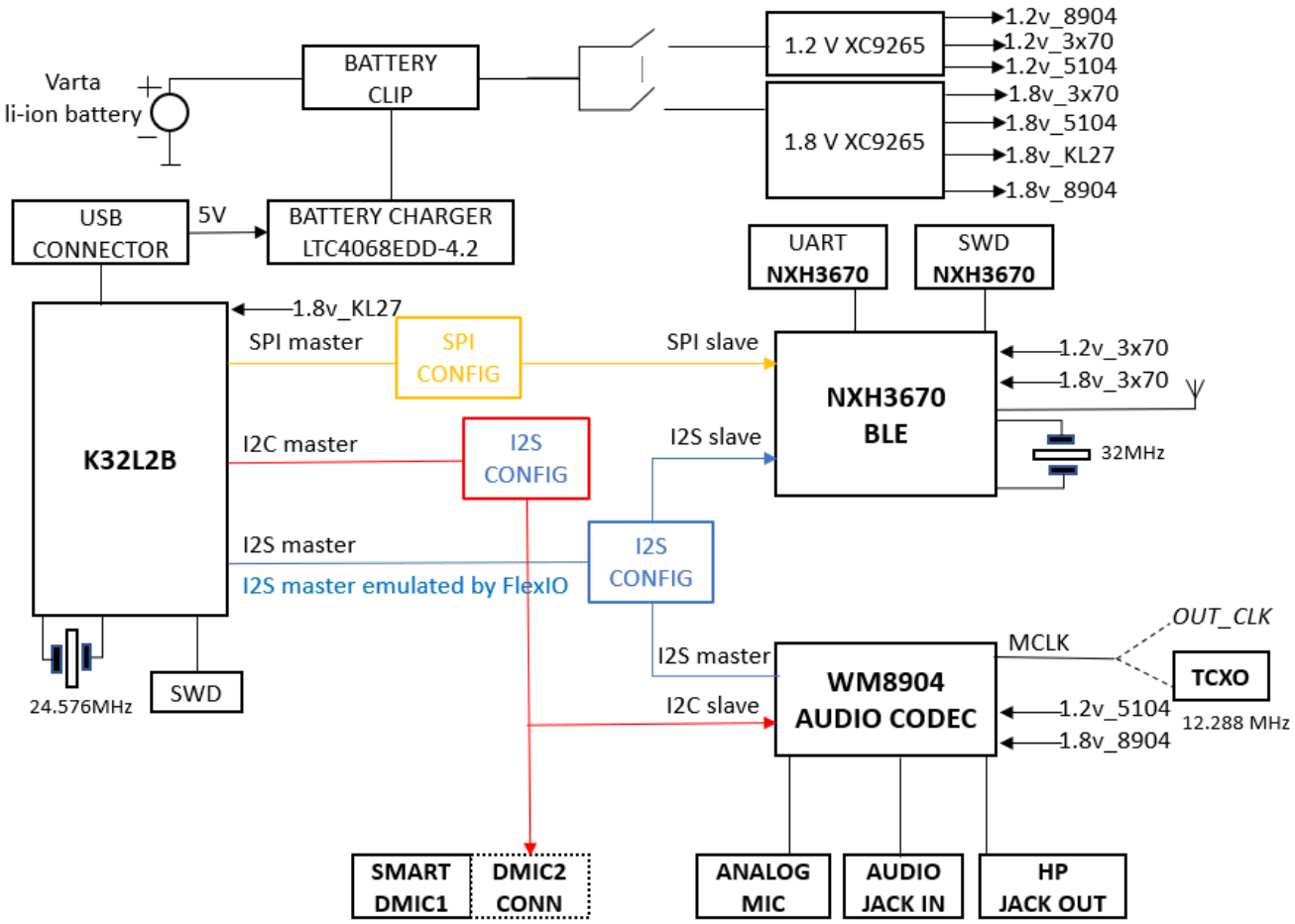

Figure 1. K32L2B + NxH3670 SDK board block diagram

As shown in Figure 1, **SPI CONFIG**, **I2C CONFIG**, and **I2S CONFIG** indicate the master selection of the communication interface. For example, for **I2S CONFIG**, choose **K32L2B** as **I2S master** in the Dongle design section while **WM8904** as **I2S master** in the Headset design.

NOTE

Considering that the K32L2B contains no I^2S peripheral, users can configure FlexIO peripheral to generate all required I^2S bus signals.

Figure 2 shows the block diagram of K32L2B_Dongle.

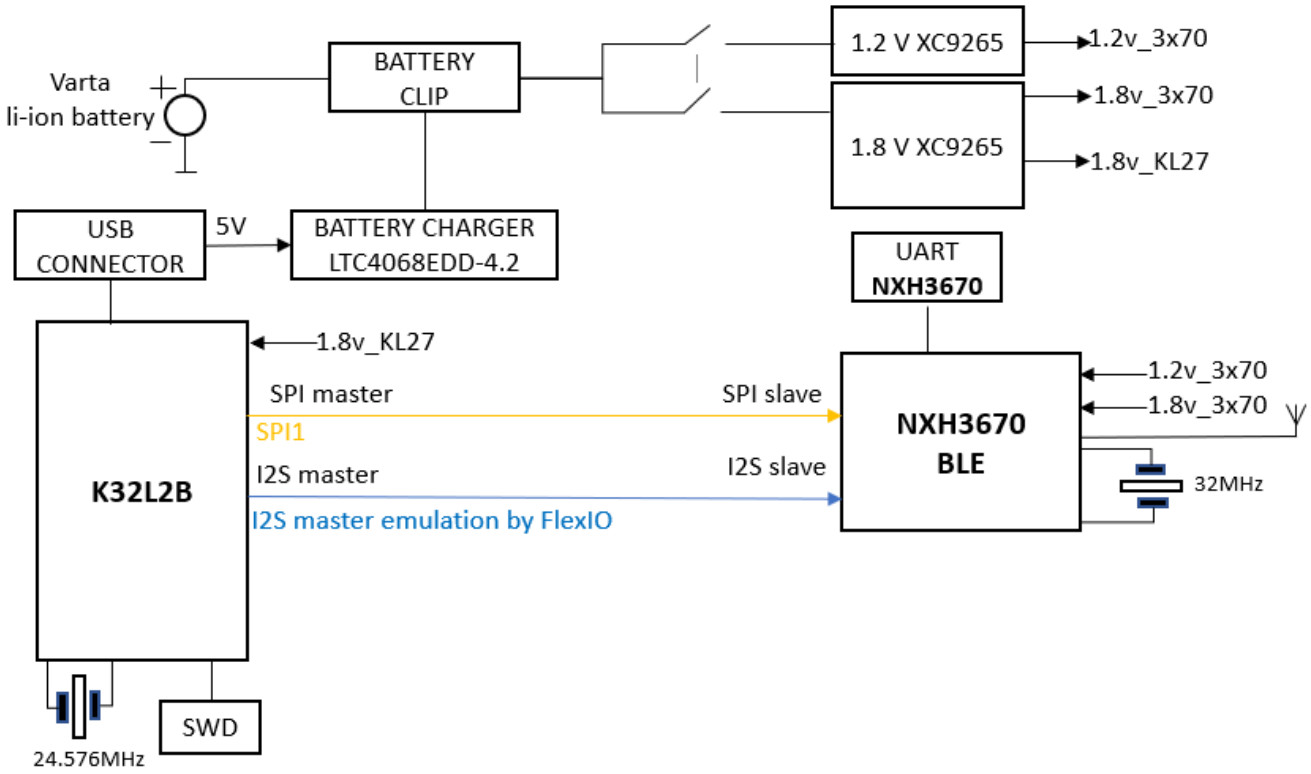

Figure 2. Block diagram of Dongle configurations

As shown in Figure 2, the system consists of:

- A Host controller (K32L2B) to run Dongle and OTA_Dongle demos.
- An NXH3670 to communicate with K32L2B through the SPI interface and transfer audio stream via the I^2S bus signals emulated by FlexIO peripheral.

2.2 USB Dongle software architecture

Figure 3 and Figure 4 show the software structure.

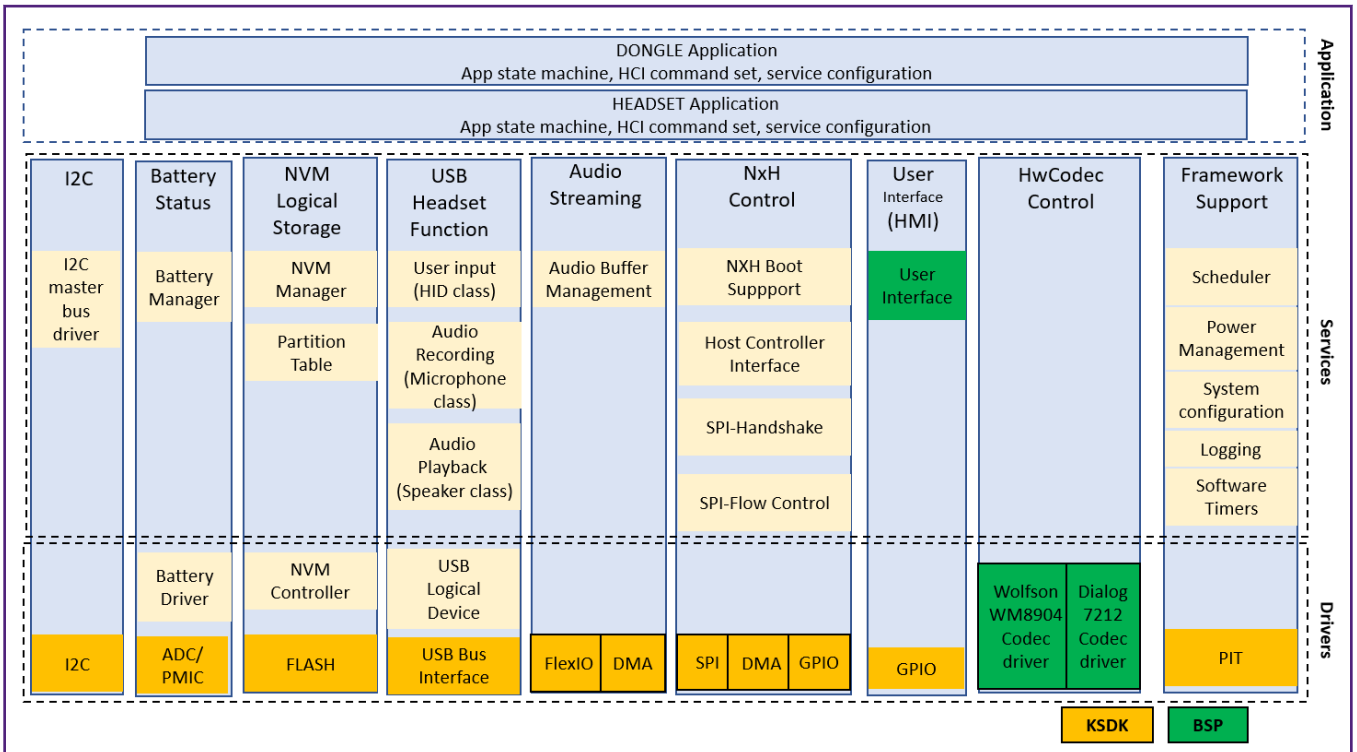

Figure 3. Application framework architecture

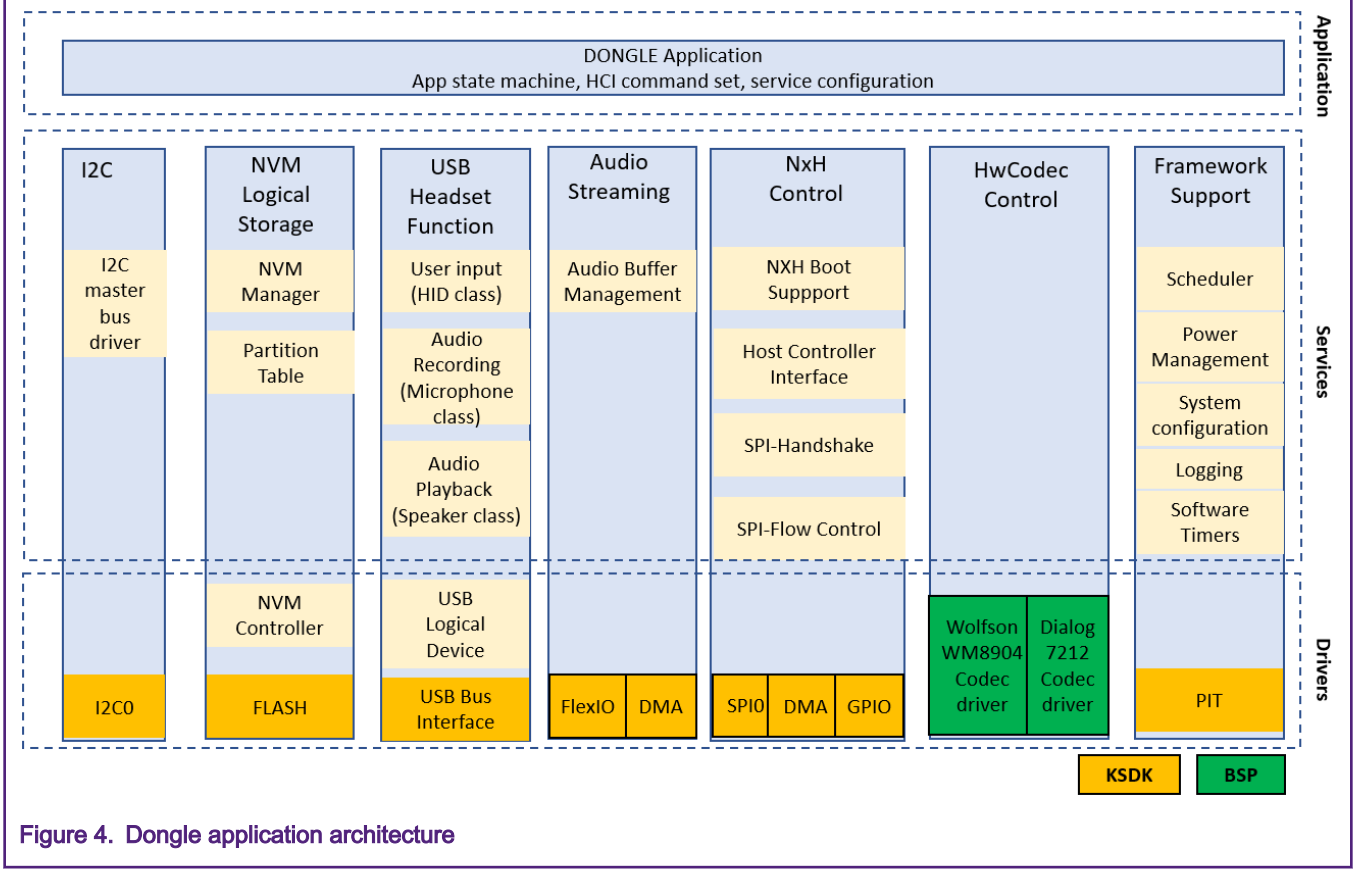

Figure 4. Dongle application architecture

The architecture contains **NVM service**, **USB service**, **Audio service**, **NXH service**, and **User Interfaces (UI) service**. This document lists the following functions:

1. **NVM service**: To read **Partition Table**.
2. **NxH Control**: To boot, start and transfer data with K32L2B using the SPI interface.
3. **UI**: Buttons used to control the volume, play and pause.
4. **Audio service**: To transmit audio data to **SHIFTBUF0** of FlexIO.

NOTE

Audio data is moved from the ring buffer directly to the **SHIFTBUF0** of FlexIO using DMA channel without software intervention.

5. **USB controller**: USB is configured as User Audio Interface (UAC).

Figure 5 shows the audio transfer process.

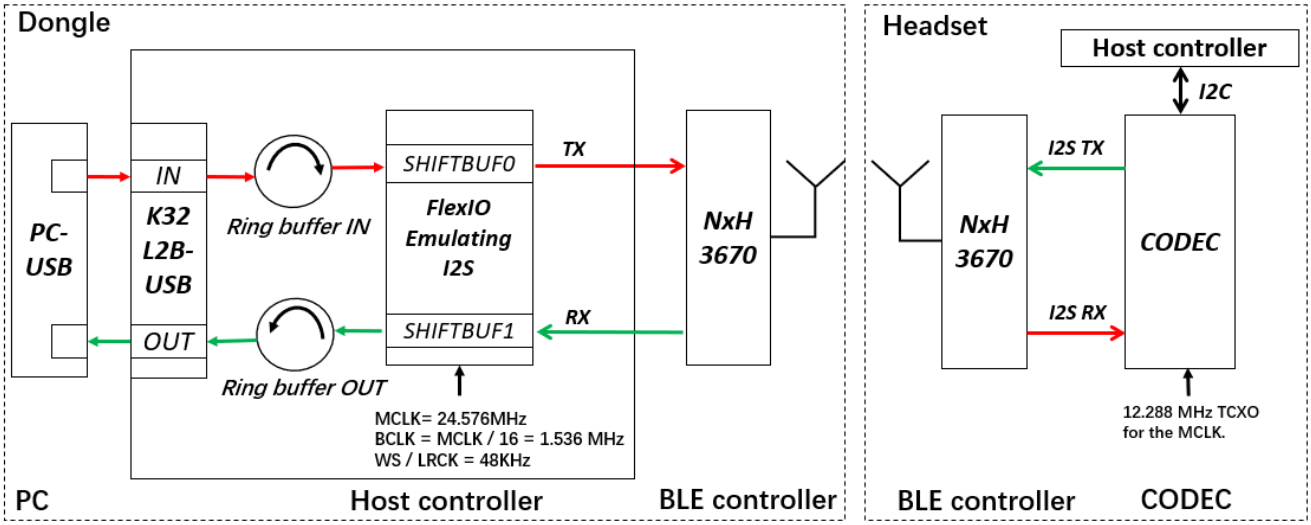

Figure 5. Audio path

- → : Playback (forward channel): The audio path is from the PC to the Headset.
  - The USB stack of the host controller will create an interrupt once a transfer is completed.
    - The software will copy the transfer data from the USB stack into a circular buffer, the input-buffer. This buffer will queue audio data until a fraction of its buffer capacity is filled, for example, 50%.
    - The software will enable a DMA channel to transfer audio data from the circular buffer to the **SHIFTBUF0** of FlexIO without software intervention.
    - The Bluetooth LE controller of Dongle is connected with Host Controller via I^2S bus signals emulated by FlexIO peripheral. In turn, the I^2S data OTA will be transferred to the Bluetooth LE controller of Headset.
    - The Bluetooth LE controller is connected with CODEC via I^2S. In turn, the received I^2S data will be transferred to CODEC without software intervention.
- ← : Record (backward channel): The audio path is from the Headset to the PC.
  - The audio is entered through **LINE IN** or DMIC to CODEC connected with the Bluetooth controller of Headset via I^2S. In turn, the received audio data will be transferred to the Bluetooth LE controller of Dongle without software intervention (currently, CODEC is the master of I^2S).
    - DMA channel will transfer received audio data from **SHIFTBUF1** of FlexIO peripheral to a circular buffer.

— The software will copy the transfer data from a circular buffer to the USB stack.

This document introduces only the audio transfer process of the Dongle section. For more information of the Headset section, refer to *K32L2B USB Headset with NXH3670*.

3 Components of USB Dongle

3.1 K32L2B

3.1.1 Host controller (K32L2B)

The device is highly-integrated, market leading ultra low-power 32-bit microcontroller based on the enhanced Cortex-M0+ (CM0+) core platform. K32L2B USB Headset with NXH3670 contains the following features:

- Core platform clock is up to 48 MHz and bus clock is up to 24 MHz.
- Memory option is up to 256 KB flash and 32 KB RAM.
- Wide operating voltage ranges from 1.71 to 3.6 V with fully functional flash program/erase/read operations.
- Two SPI modules that support 16-bit data length.
- Two inter-integrated circuit (I^2C) modules.
- One FlexIO module.

3.1.2 Clock

1. One reference clocks used on the board.
   - 32 MHz crystal connected with the NxH3670.
2. The FlexIO module acts as the I^2S bus master producing all required signals.
   - BCLK is 1.536 MHz.
   - Word Select (WS)/Left-Right Clock (LRCK) is 48 KHz.

   Figure 6 shows the clock information after FlexIO peripheral is configured correctly.

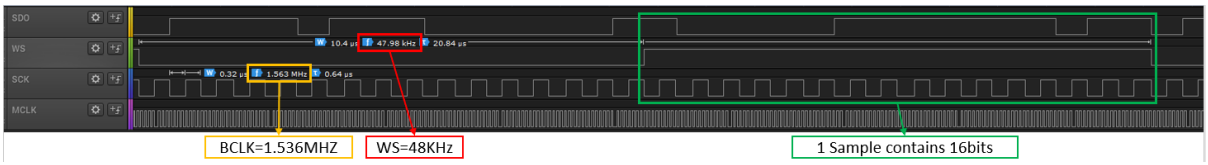

Figure 6. Clock information in audio transfer process

3.1.3 Pin connections

Table 2 lists the pin connection of K32L2B and other components.

Table 2. Pin connections

Function	Jumper	Name	Jumper	Name
	K32L2B Dongle	KL2X_I2S_MASTER	NXH3670	BLE_I2S_SLAVE
FlexIO emulating I^2S (connected with MCU)	J2-8 (PIN PTD6)	KL2X_SDI	J12_1 (I2S_CONFIG)	BLE_SDO

Table continues on the next page...

Table 2. Pin connections (continued)

Function	Jumper	Name	Jumper	Name
	K32L2B Dongle	KL2X_I2S_MASTER	NXH3670	BLE_I2S_SLAVE
	J1-6 (PIN PTD3)	KL2X_SDO	J12_3 (I2S_CONFIG)	BLE_SDI
	J2-12 (PIN PTD5)	KL2X_WS	J12_5 (I2S_CONFIG)	BLE_WS
	J2-6 (PIN PTD4)	KL2X_SCK	J12_7 (I2S_CONFIG)	BLE_SCK
NXH handshake	J1_2 (PIN PTA1)	BLE_SPIS_INTN	J16_9 (BLE_SPI)	SWM4 (-INTN)
	J1_8 (PIN PTA12)	BLE_SPIS_SRQ	J16_11 (BLE_SPI)	SRQ
SPI (SPI0)	J1-11 (PIN PTC7)	BLE_SPIS_MISO	J16_1 (BLE_SPI)	SW0
	J1-9 (PIN PTC6)	BLE_SPIS_MOSI	J16_3 (BLE_SPI)	SW1
	J1-15 (PIN PTC5)	BLE_SPIS_SCLK	J16_5 (BLE_SPI)	SW2
	J1-7 (PIN PTC4)	BLE_SPIS_SSN	J16_7 (BLE_SPI)	SW3
NXH reset	J1_4 (PIN PTA2)	BLE_RESETN	J20_5 (BLE_SWD)	POR_RESETN

3.1.4 Schematic

1. Audio transfer
   - I²S

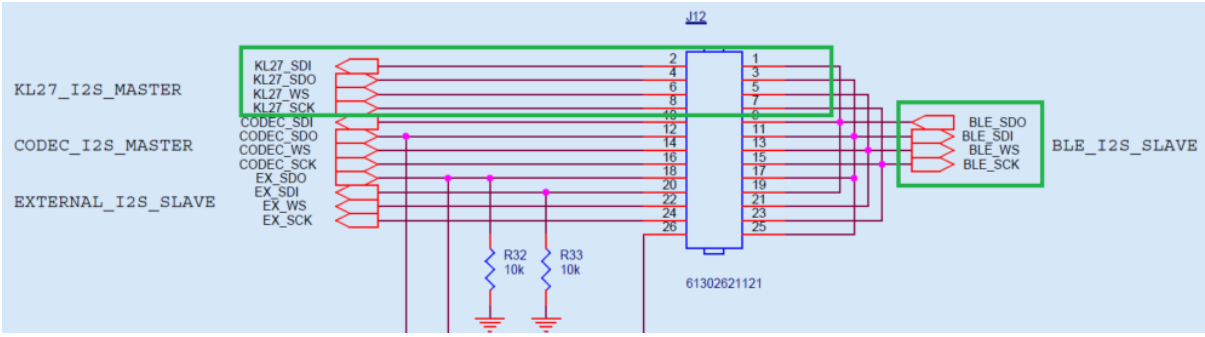

Figure 7. I²S interface connection

In the Dongle section, the host controller (K32L2B) transfers data directly to the NXH3670 via I²S bus signals emulated by FlexIO peripheral.

2. NXH3670
   - NXH handshake

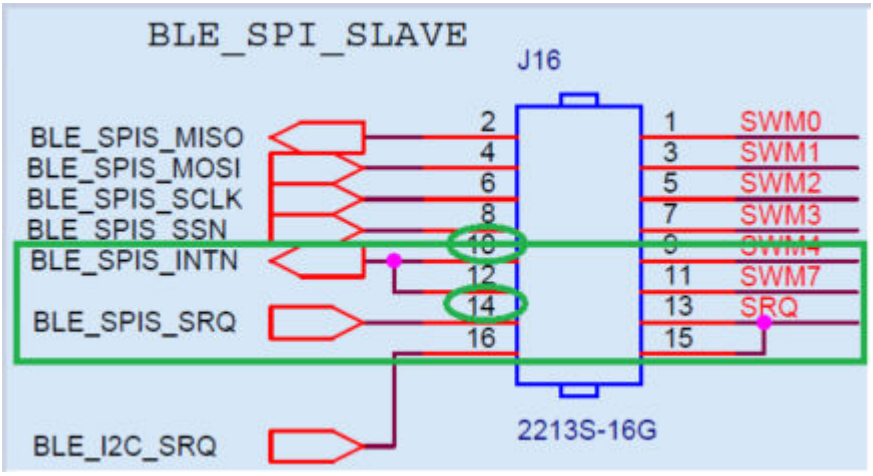

Figure 8. Handshake pins (SRQ and INTN) connection

- SPI

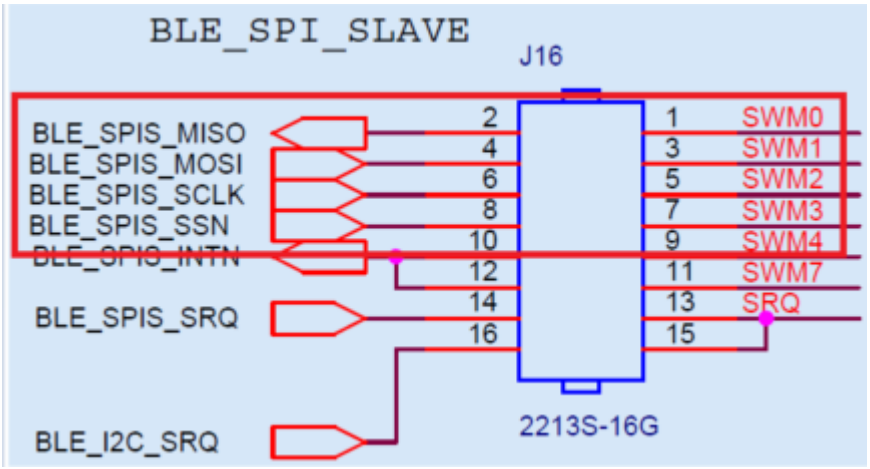

Figure 9. SPI interface connection

- Power On Reset (POR)

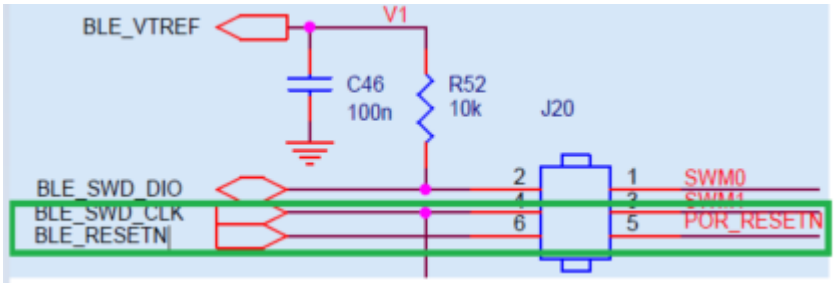

Figure 10. POR pin connection

3.1.5 Pin configurations

- SPI

  - Interface: SPI0.
  - Pins: CS (PTC4), SCK (PTC5), MISO (PTC7), MOSI (PTC6)
  - 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: The value of Baud Rate for SPI is configured to 8000000 u.
- FlexIO pin used to emulate I^2S
  - TXD: PTD3
  - RXD: PTD6
  - BCLK: PTD4
  - FS: PTD5
- NxH3670 pin
  - INIT (PTA1), configured as digital input.
  - SRQ (PTA12), configured as digital output.
  - POR (PTA2), conigured as digital output.

3.2 NXH3670

3.2.1 Bluetooth Low Energy

The NxH3670 is the Bleutooth Low Energy (Bluetooth LE) device. It is a single chip, ultra-low power 2.4 GHz transceiver with embedded MCU, targeted at wireless audio streaming for Headsets, wireless Headsets, and headphones.

The features include:

1. KEY FEATURES
   - Support for high-quality, low-latency (<20 ms) wireless audio streaming.
   - Integral wireless audio streaming solution
     - Integrated Arm® Cortex®-M0 processor
     - Integrated CoolFlux DSP and HW accelerators for audio processing
   - Ultra-low-power operation:
     - Stereo audio streaming with mono reverse channel at 7.5 mW
   - Packaged as bumped die <7.25 mm^2
   - Typical supply voltage: 1.2 V
2. PROPRIETARY AUDIO STREAMING PROTOCOL

   The NxH3670 device simultaneously runs the standard Bluetooth LE protocol and NXP's proprietary audio protocol. This audio streaming protocol is configured to support audio streaming between a Dongle device and a headphone with the following audio configurations:

   - Forward audio path
     - Stereo
     - 48 KHz sampling rate, 16 bit resolution
     - Audio BW > 20 KHz
     - SBC HQ codec

  - Latency < 20 ms
- Simultaneous return audio path
  - Mono
  - 16 KHz sampling rate, 16 bit resolution
  - Audio BW > 6 KHz
  - G.722 codec

Power consumption at the headphone side during audio streaming is an industry-record of only 7.2mW, enabling extended play time and reduced battery size. During audio streaming, a simultaneous, bi-directional data connection between audio source and audio sink is available as well with up to 8kbps of throughput.

Figure 11 shows the connection process between Dongle and Headset of NXH3670.

1. Download and start NxH3670 images.
   - In the **Boot** step, the host controller Load images from flash/eeprom to NXH3670 through the SPI interface.
   - In the **Starting** step, the host controller need handshake with NXH3670. Then the software will register an event table with the HCI layer used to handle event sent from NXH3670.
2. Pair.

   The NXH3670 on Dongle and Headset board will pair with each other. For example, Dongle will retrieve PD from Headset.
3. Connect.

   The NXH3670 on Dongle and Headset board will connect with each other, and then transfer data between each other if connected successfully. For example, Dongle can send audio stream to Headset.

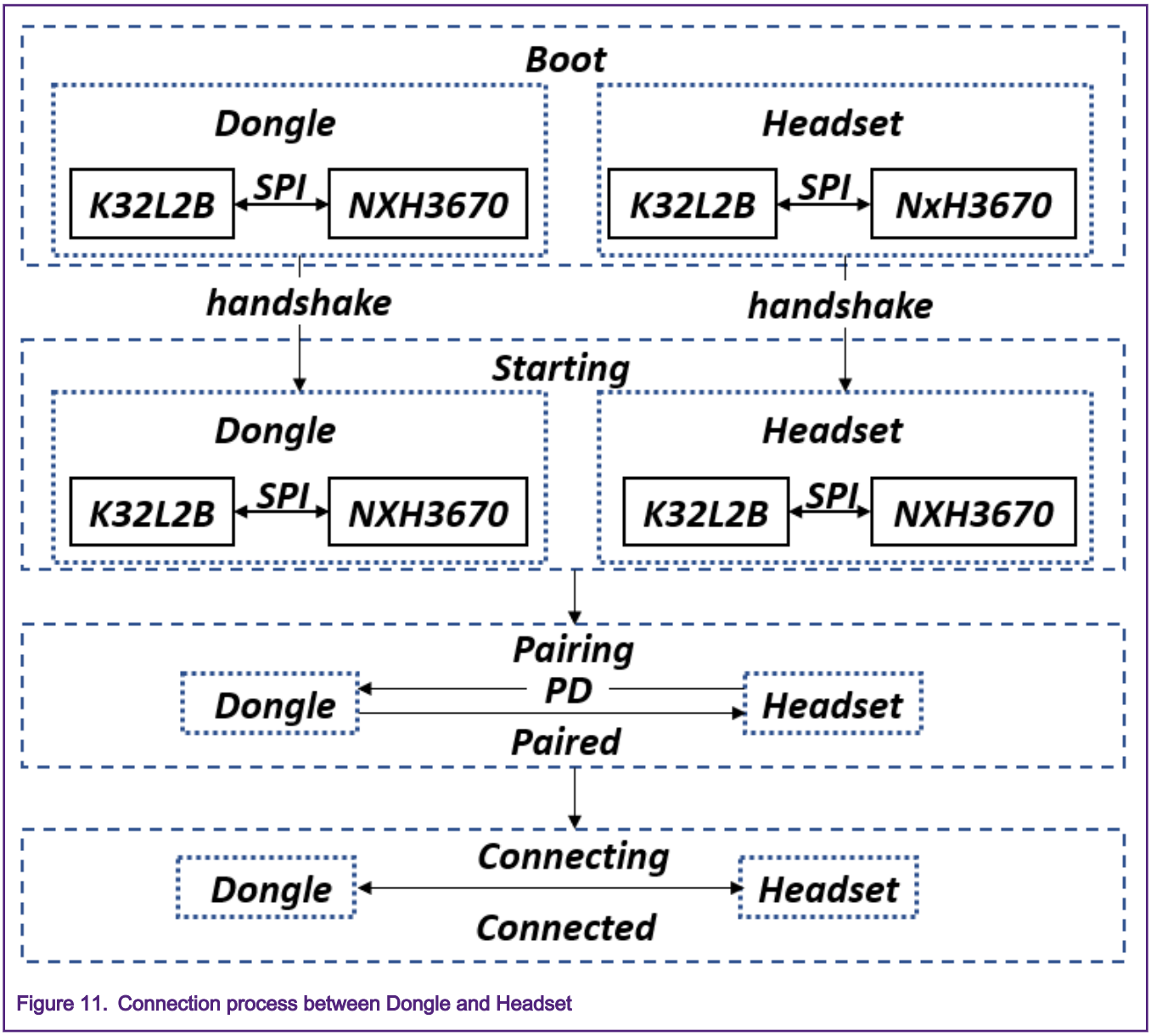

Figure 11. Connection process between Dongle and Headset

3.2.2 Boot

3.2.2.1 NXH3670UK bootloader

The most important task of the bootloader is to prepare the NXH3670UK to start a user application. To achieve this, the typical bootloader lifecycle is:

1. Configure the device.
2. Load the memories. By default, the NXH3670UK starts up in the host-assisted mode: the SPI slave interface.
3. Enter the active mode.

3.2.2.2 Partition table

Since the NxH3670 Bluetooth LE radio has no means to store data persistent, the Flash memory of the Host Controller is used for storage.

The reference application has functionality to split up this memory into logical partitions.

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

layout_release_sdk.yml

```
# number of partitions: 3
# table address: 0xa00
# max used size (including ssb and table): 256 KB
active_partition: 0
layout_version: 0x30
entries:
    # partition_id 0
    - name: "app"
      type: firmware
      base_address: 0xbf0
      size: 0x3ec00 # 251 KB
      offsets:
          - 0x0          # kl_app | 130 KB (95 KB free)
          - 0x20810      # nxh_app | 65 KB
          - 0x30c10      # rfmac | 16 KB
          - 0x34c10      # cf | 39 KB
    # partition_id 1
    - name: "app_data"
      type: appdata
      base_address: 0x3f800
      size: 0x400 # 1 KB
      offsets:
          - 0x0          # app_data | 1 KB
    # partition_id 2
    - name: "pairing_data"
      type: appdata
      base_address: 0x3fc00
      size: 0x400 # 1 KB
      offsets:
          - 0x0          # pairing_data | 1 KB
```

Figure 12. Partition table/layout of release mode of Dongle

As shown in Figure 12, `partition_id 0` contains four images, `kl_app`, `nxh_app`, `rfmac`, and `cf`. For example, the offset of `nxh_app` is `0x20810`, which indicates that this image will be downloaded to `0x21400` (`0xbf0` + `0x20810`).

Users can design their own partition tables and the following two notes are important to keep:

1. If users want **Partition1: app_data** to be the first partition in memory, in the `layout_release_sdk.yml` file, keep the order as: `app_data`, `app`,....

   If users do not follow the rule, the tool will not output a `Partition Table.bin` or be used as correct Partition Table.

2. Users must make sure that **base_address of Partition + size of partition0** is smaller than **base_address of the Partition1**.

3.2.2.3 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 K32L2B can be used to save firmware of NXH3670. Taking Dongle as an example, users need to store `phGamingTx.ihex.eep`, `phStereoInterleavedAsrcTx.eep`, and `rfmac.eep` in advance in the NVM, which will take up about 120 k.

3.2.2.4 EEP

1. **Definition**

   For safety reasons, the NVM-image contains CRCs and signatures at different levels (i.e. block level and overall). The function helps to detect the corrupted images and abort the loading and the execution of potentially harmful instructions.

Table 3. Format of EEP file - Single image (LSB first)

	1 byte	1 byte	1 byte	1 byte
SIGNATURE	0xCA	0xFE	0xBA	0xBE
HEADER	Image Length			Type
	Destination Address			
	CheckSum			
Image Length Size	Program Image			
HEADER	Image Length = 0			Type
	Destination Address			
	CheckSum			

Table 4. Format of EEP file - Multiple image (LSB first)

	1 byte	1 byte	1 byte	1 byte
SIGNATURE	0xCA	0xFE	0xBA	0xBE
HEADER	Image Length			Type
	Destination Address			
	CheckSum			
Image Length Size	Program Image			
HEADER	Image Length = 0			Type
	Destination Address			

Table continues on the next page...

Table 4. Format of EEP file - Multiple image (LSB first) (continued)

	1 byte	1 byte	1 byte	1 byte
	CheckSum			
Image Length Size	Program Image			
HEADER	Image Length = 0			Type
	Destination Address			
	CheckSum			

All fields listed in Table 3 and Table 4 are stored in 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.

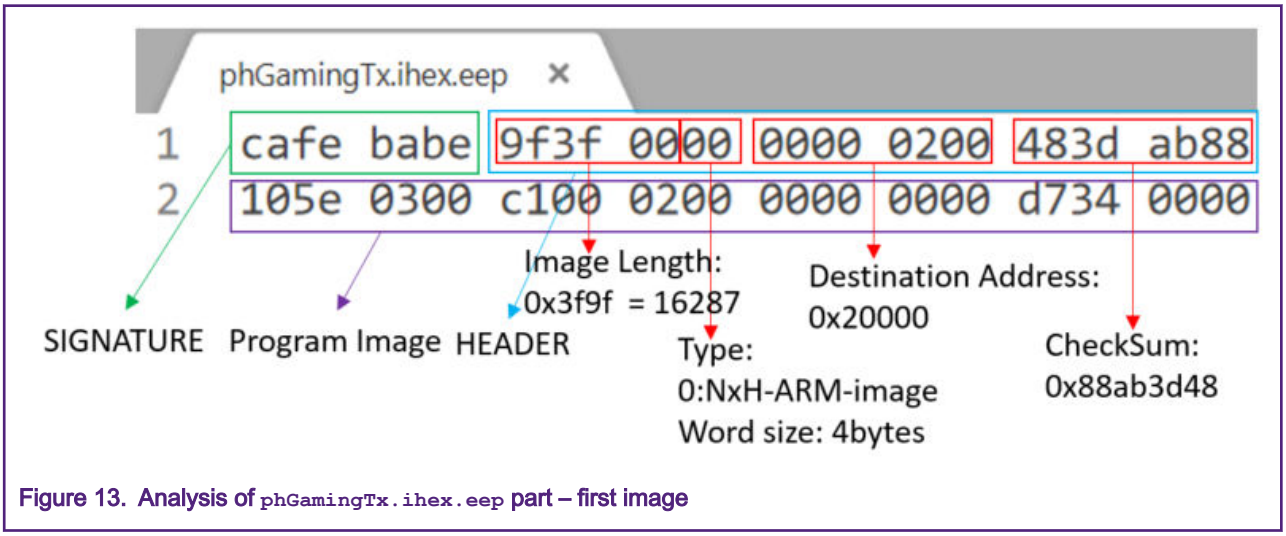

Figure 13. Analysis of `phGamingTx.ihex.eep` part – first image

As shown in Figure 13, the Image Length is 16287 and Type ID is 0. It indicates that host controller will send 65148 (16287*4) bytes to NXH3670 through SPI.

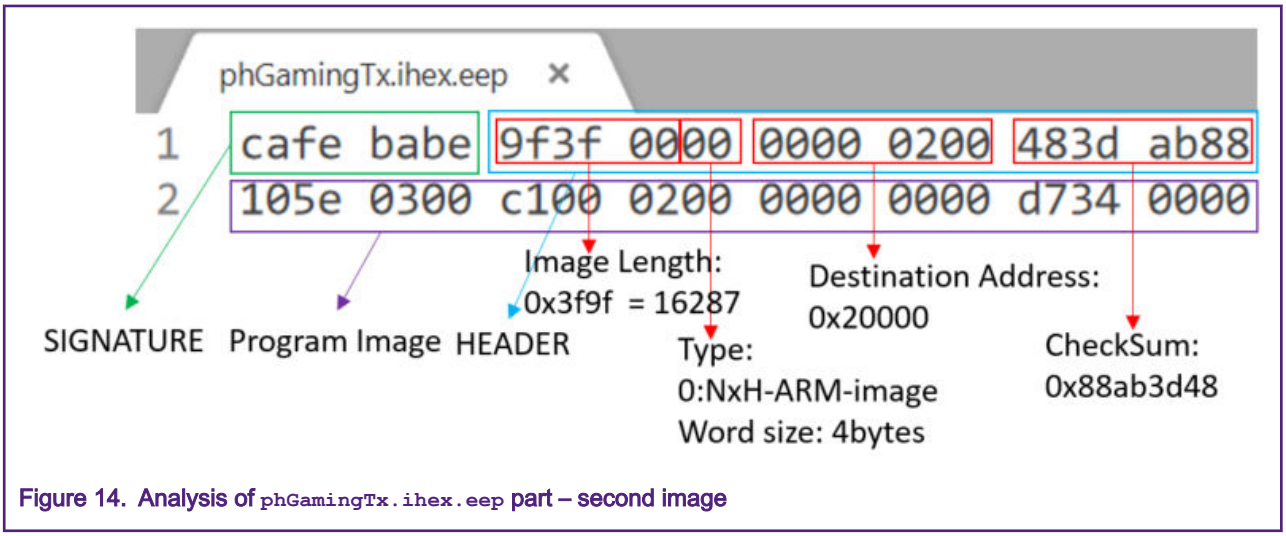

Figure 14. Analysis of `phGamingTx.ihex.eep` part – second image

As shown in Figure 14, the Image Length is 284 and Type ID is 0. It indicates that the host controller will send 1136 (284*4) bytes to NXH3670 through SPI.

2. **Downloading an `.EEF` file to K32L2B3**

This document provides two methods to download an .EEF file to K32L2B3.

a. Transform an .EEF file to the HEX buffer.

- Winhex
- __attribute__ ((section (.ARM.__at_address)))

This method helps to store NXH3670 relevant firmware as a buffer, as the OTA process only re-writes the application of host controller instead of NXH3670.

b. Transform an .BIN file to an .EEP file.

- SDK packet haa a tool called to_eep.cmd in release.

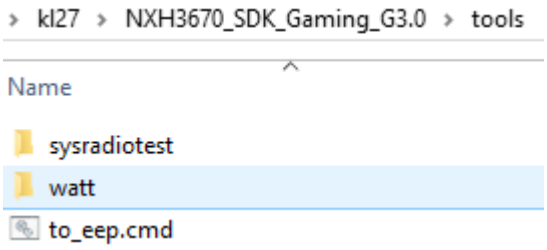

Figure 15. to_eep.cmd in packet

- Input:

```
to_eep.cmd -i spi_dma_b2b_transfer_master.bin -o spi_dma_b2b_transfer_master.eep
```

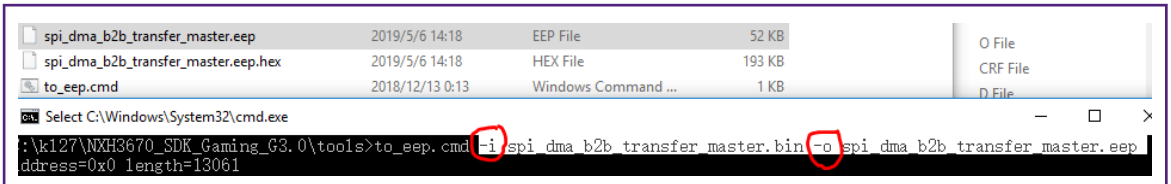

Figure 16. Command string in to_eep.cmd in packet

- As shown in Figure 17, the bin file was packet with **SIGNATURE** and **HEADER**.

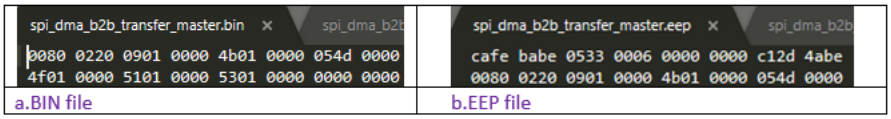

Figure 17. .BIN and .EEP files

This method helps to store the application of host controller instead of NXH3670 firmware. For example, users can transfer the .BIN file of the application to .EEP.BIN file with CRCs and signatures that will be useful in OTA process.

3.2.2.5 NXH3670 host interface: SPI

1. **SPI bus**

For NXH3670, the boot loader configures the SPI slave interface and assumes the host to be SPI master. The SPI slave operation mode is configured as follows:

- 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 typically allow configuration of clock phase and polarity. These are sometimes referred to as numbered SPI modes, as described in Table 5 and shown in Figure 18. CPOL and CPHA are configured by bits in the SPI Control Register 1 (`SPIx_C1`).

Table 5. SPI mode summary

CPOL	CPHA	SPI mode	Description	SCK rest state	SCK data change edge	SCK data sample edge
0	0	0	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.	low	falling	rising
0	1	1	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.	low	rising	falling
1	0	2	Same as mode 0 with SCK inverted.	high	rising	falling
1	1	3	Same as mode 1 with SCK inverted.	high	falling	rising

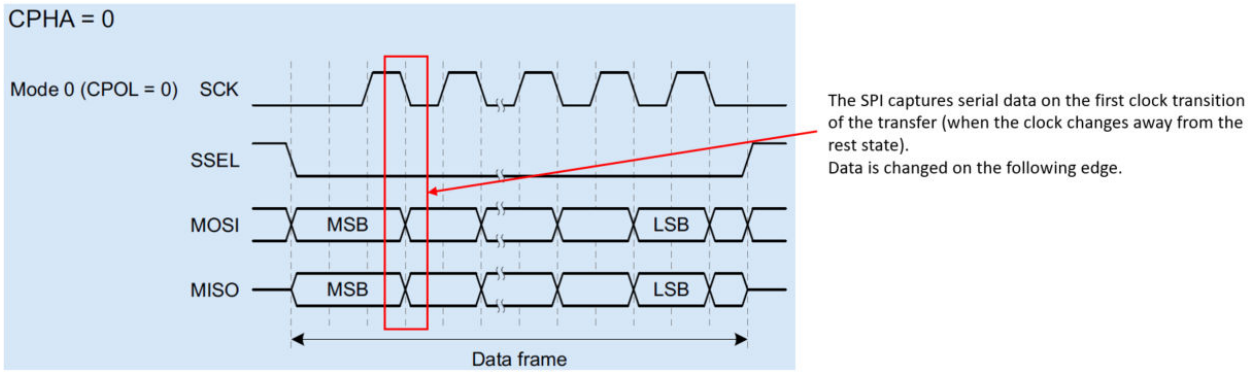

Figure 18. Basic SPI operating mode: mode0

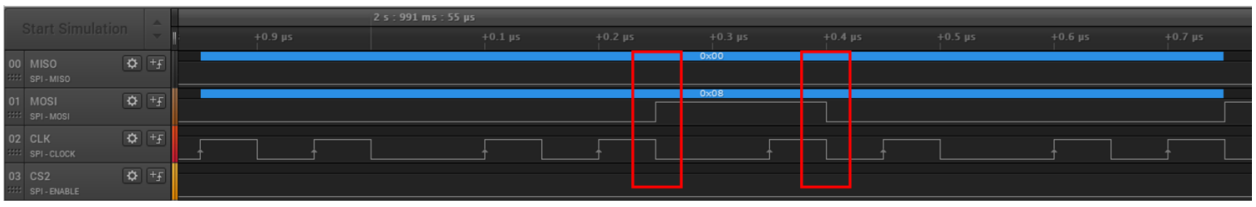

Figure 19. mode0 example of Logic analyzer

2. **SPI flow control**

In the **Bluetooth LE Audio System**, SPI transfer must comply with the format shown in Figure 20.

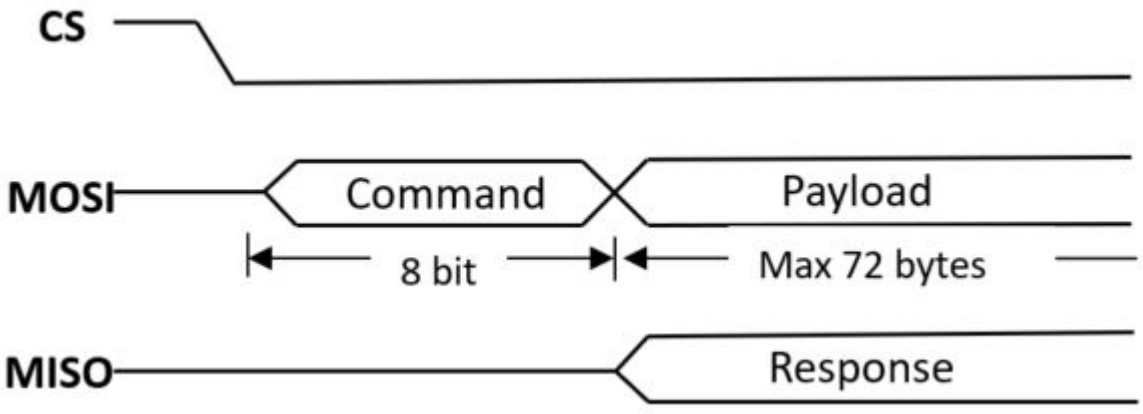

Figure 20. SPI slave – Transfer format

Table 6. SPI slave – Supported command

Command	Opcode	Description
Write command	`0b010xxxxx`	Write payload to NxH3670 SPI slave.
Read command	`0b110xxxxx`	Read pending data from NxH3670 SPI slave.
Read status	`0b101xxxxx`	Read status byte.
Read extended status	`0b111xxxxx`	Read extended status byte.

For example, the **Write command** was defined by `#define SPI_WRITE_CMD (0x40u)` in the software design. To make user understand SPI transfer easily, this document will analyze the signal of Logic analyzer as shown in Figure 21.

Figure 21. Transfer mode

3.2.2.6 HCI command format

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, as shown in Figure 22.

Bit 0	8	16	24	31
HCI Packet type (command = 0x01)		Opcode		ParamLength
Param0		Param2		Param...
...				
Parameter N				

Figure 22. HCI command format

A command starts with:

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

HCI opcodes and optional parameters bytes in the commands are always LSB first.

An SPI slave – Transfer format of Logic analyzer is as shown in Figure 23.

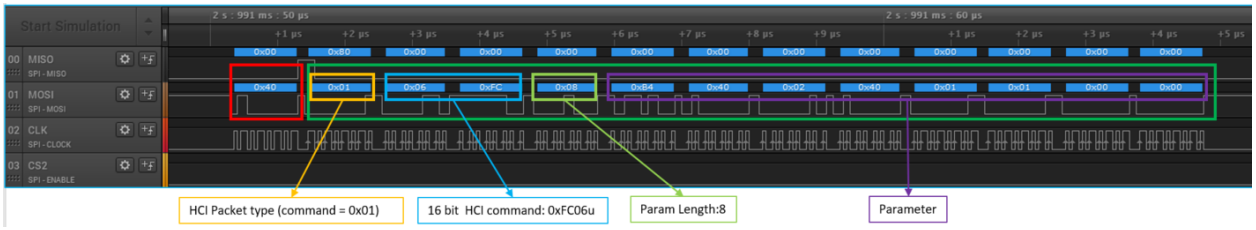

Figure 23. An SPI slave – Transfer format of Logic analyzer

2. **HCI event format**

Results of commands are sent back as HCI formatted events. Whenever the HCI controller sends 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.

Bit 0	8	16	24	31
HCI Packet type (event= 0x04)		Event code	Param Length	Param 0
Param 1		Param 2	Param 3	Param...
...				
Parameter N				

Figure 24. HCI Event format

Figure 25 shows an HCI event format of logic analyzer.

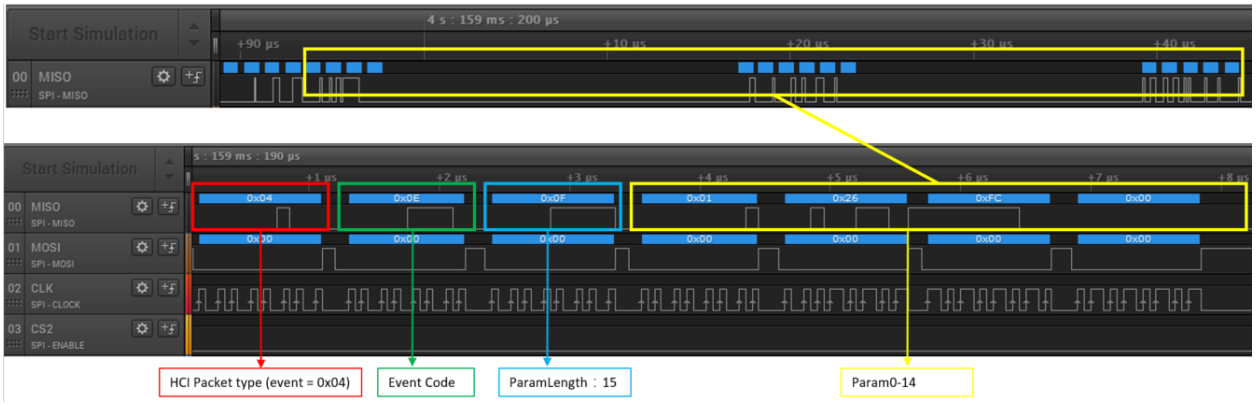

Figure 25. An HCI Event format of Logic analyzer

3. **HCI command transfer**

Figure 26 shows a sequence of how an HCI command to be sent to the NxH3670.

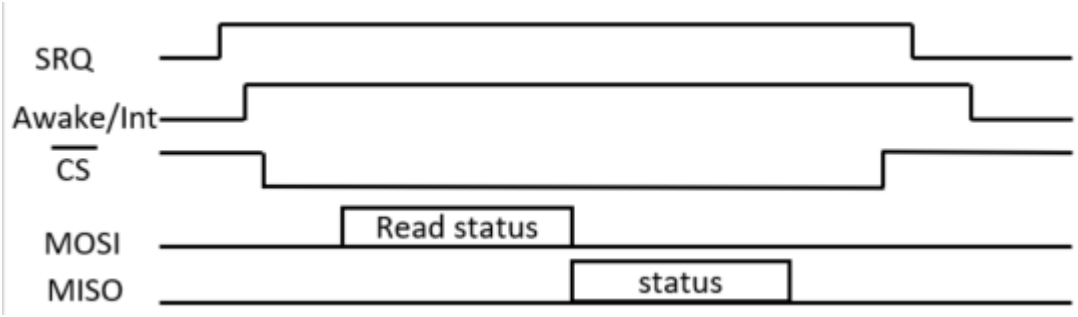

Figure 26. HCI command transfer – Read status

Before an SPI transfer can be started, it must check the NxH3670 is awake and the SPI bus is available. This is done by asserting the SRQ line and waiting for the confirmation on the awake/int signal.

The host knows the NxH3670 is awake but must still check if the NxH3670 is ready to accept new SPI data. This information can be retrieved with the SPI read-status-command.

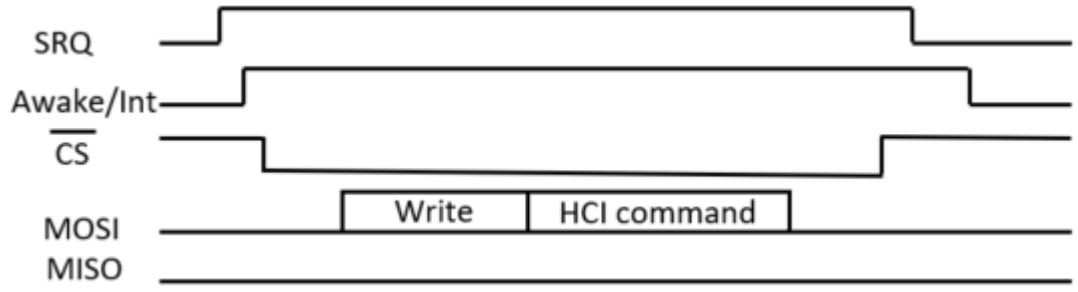

Figure 27. HCI command transfer – Write command

Users can see the change of signals, including CS, MOSI, and MISO, as shown in Figure 28.

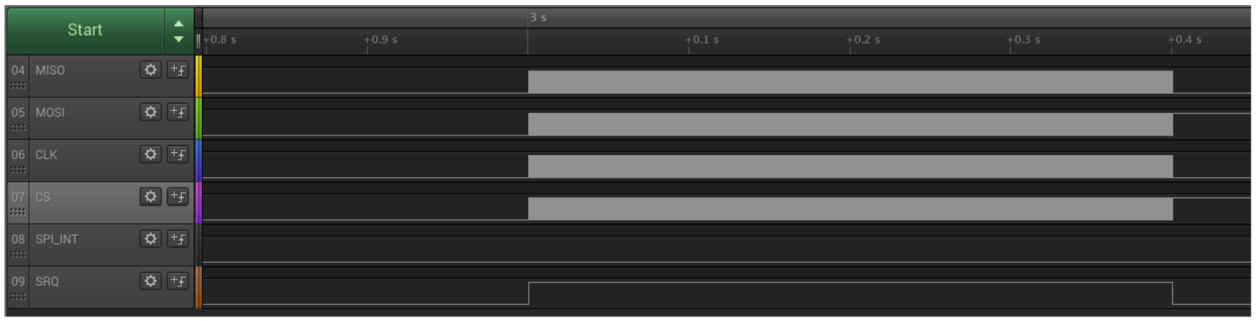

Figure 28. Boot process of NXH3670

4. **HCI event transfer**

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

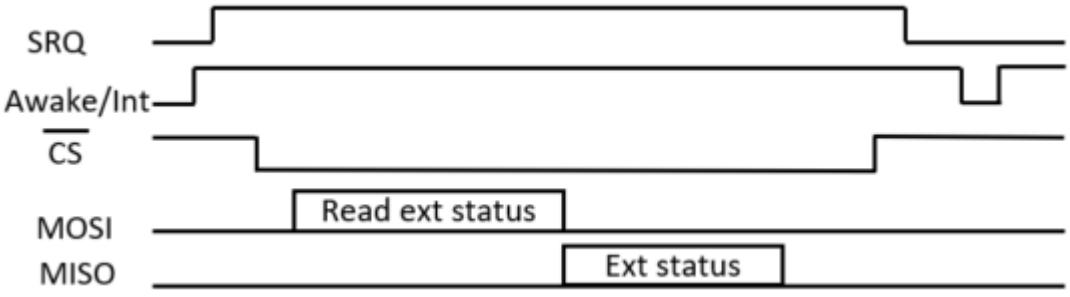

Figure 29. HCI event transfer – Read extended status

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 deasserted soon after. Since the actual pending data has not been read, the awake/int signal is asserted again.

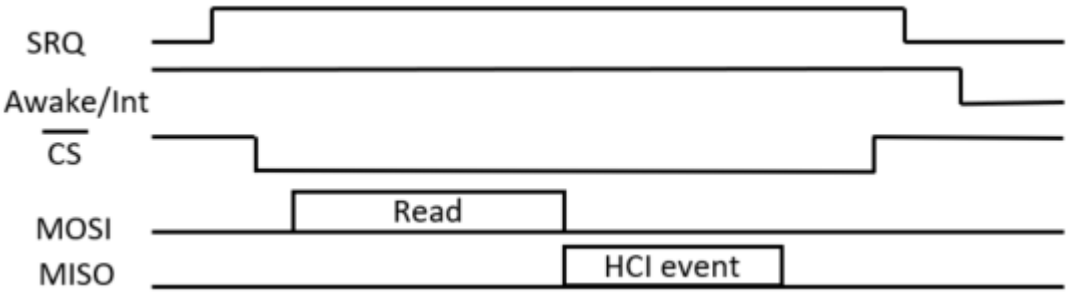

Figure 30. HCI event transfer – Read

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

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

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

3.3 Handshake

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

3.3.1 Logical signals

1. Service request signal

   This signal is used by the host to request service by the NxH3670. When the NxH3670 detects the signal, it indicates it is ready to handle the service request by asserting the awake signal.

2. Awake signal

   This signal is used by the NxH3670 to indicate it is awake. The NxH3670 only asserts this when the SRQ signal is asserted and not every time it wakes up.

3. Pending data signal

   When the NxH3670 has pending data, it asserts this signal toward the host.

3.3.2 Physical signals

These three logical signals are mapped onto two physical signals to reduce required pin count.

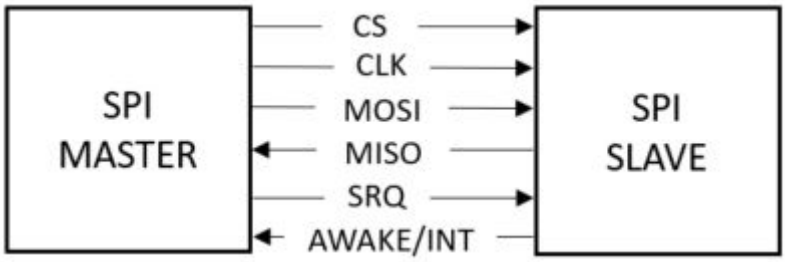

Figure 31. SPI handshake physical signals

Table 7. Physical to logical signal mapping

Logical signal	SRQ physical signal	AWAKE/INT physical signal
service request signal	asserted	don’t care
wake signal	asserted	asserted
pending data signal	deasserted	asserted

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 requests an SPI transfer simultaneously.

To make user understand the process of handshake easily, this document introduces the signal of Logic analyzer about scenario 2.

When the NxH3670 has pending data, it generates the sequence to report pending data.

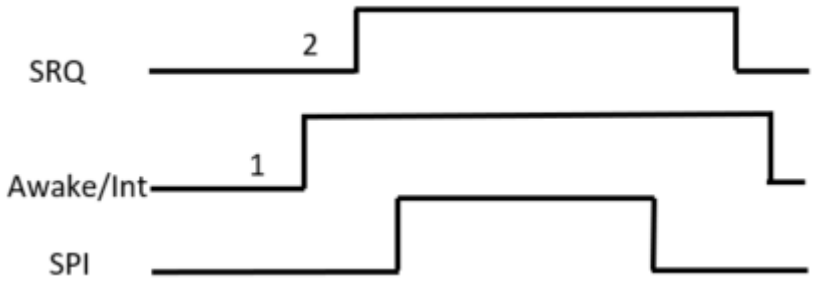

Figure 32. NxH3670 requested SPI transfer

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.

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

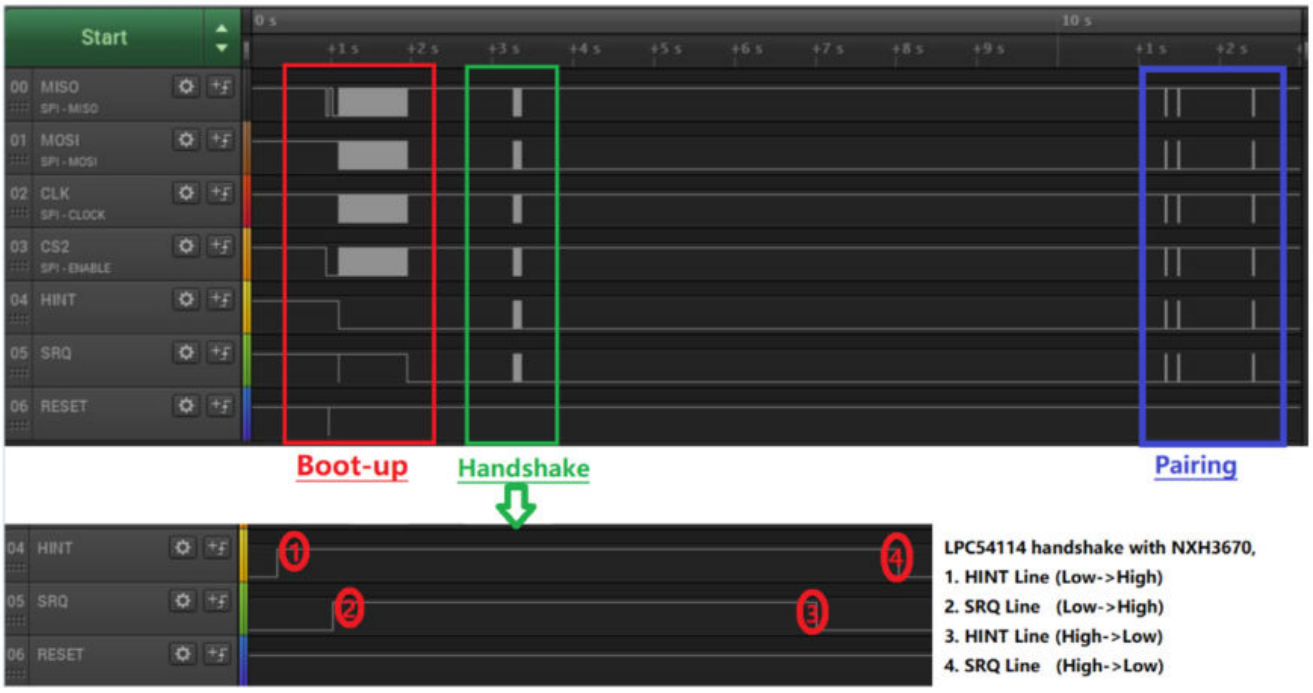

Figure 33. NxH3670 requested SPI transfer

3.4 Start

Users can use a USB cable to connect J13 (FRDM-K32L2B) with PC to power or download firmware.

4 Conclusion

This document describes the hardware design and software architecture (top-level design) of K32L2B_Dongle in the **Bluetooth LE Audio System**. It can be a reference for users to build their own demo.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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: AN12647

arm