

by: NXP Semiconductors

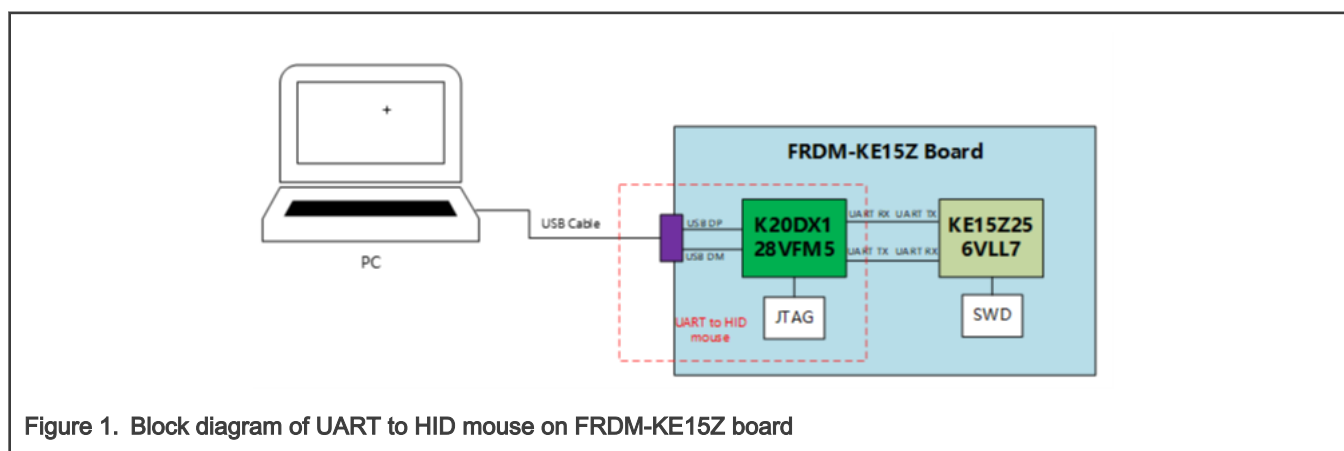
1 Introduction

This application note describes how to implement the function of UART to HID mouse on K20. NXP's Kinetis series FRDM boards provide an onboard debugger with JTAG interface, using K20DX128VFM5 as the onboard debugger. K20DX128VFM5 is a Cortex[®]-M4 core processor, it contains the following features:

- The main frequency is up to 50 MHz.
- The flash size is 160 KB.
- The SRAM size is 16 KB.
- It is equipped with an FS USB device controller.

Kinetis E series MCUs have a Touch Sensing Input (TSI) module. Users may need to use the UART to HID mouse module to display the TSI demo more intuitively and convert touch events into corresponding HID mouse behaviors. This application note takes FRDM-KE15Z board as an example to implement the function of UART to HID mouse on K20. This example implements the function of setting the relative position and absolute position of the mouse through UART commands.

Figure 1 shows the system block diagram.



2 Required environment

2.1 Hardware

- FRDM-KE15Z board
- A USB cable
- PC

Contents

1	Introduction.....	1
2	Required environment.....	1
2.1	Hardware.....	1
2.2	Software.....	2
3	USB HID mouse descriptor.....	2
4	UART to HID mouse protocol.....	5
4.1	Set absolute position of the mouse	5
4.2	Set relative position of the mouse	7
5	Function test.....	8
6	Reference.....	10

2.2 Software

Since NXP does not provide SDK examples for K20DX128VFM5, this application note implements the function of K20 UART to USB HID mouse based on the *usb_device_hid_mouse* project in TWK-K60 SDK. For details, see [AN13079SW](#).

3 USB HID mouse descriptor

The K60 *SDK usb_device_hid_mouse* example only implements the function of setting the relative position of the mouse. In order to implement the function of setting the absolute position of the mouse, a new HID report descriptor, report ID 2, needs to be added to the original HID report descriptor. The complete HID report descriptor is as shown in [Figure 2](#).

```

199 uint8_t g_UsbDeviceHidMouseReportDescriptor[USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT] = {
200     0x05U, 0x01U, /* Usage Page (Generic Desktop) */
201     0x09U, 0x02U, /* Usage (Mouse) */
202     0xA1U, 0x01U, /* Collection (Application) */
203     0x09U, 0x01U, /* Usage (Pointer) */
204     0xA1U, 0x00U, /* Collection (Physical) */
205     0x85U, 0x01, /* Report ID(1) */
206     0x05U, 0x09U, /* Usage Page (Buttons) */
207     0x19U, 0x01U, /* Usage Minimum (01U) */
208     0x29U, 0x03U, /* Usage Maximum (03U) */
209     0x15U, 0x00U, /* logical Minimum (0U) */
210     0x25U, 0x01U, /* logical Maximum (1U) */
211     0x95U, 0x03U, /* Report Count (3U) */
212     0x75U, 0x01U, /* Report Size (1U) */
213     0x81U, 0x02U, /* Input(Data, Variable, Absolute) 3U button bit fields */
214     0x95U, 0x01U, /* Report count (1U) */
215     0x75U, 0x05U, /* Report Size (5U) */
216     0x81U, 0x01U, /* Input (Constant), 5U constant field */
217     0x05U, 0x01U, /* Usage Page (Generic Desktop) */
218     0x09U, 0x30U, /* Usage (X) */
219     0x09U, 0x31U, /* Usage (Y) */
220     0x09U, 0x38U, /* Usage (Z) */
221     0x15U, 0x81U, /* Logical Minimum (-127) */
222     0x25U, 0x7FU, /* Logical Maximum (127) */
223     0x75U, 0x08U, /* Report Size (8U) */
224     0x95U, 0x03U, /* Report Count (3U) */
225     0x81U, 0x06U, /* Input(Data, Variable, Relative), three position bytes (X & Y & Z)*/
226     0xC0U, /* end collection, Close Pointer collection*/
227     0xC0U, /* end collection, Close Mouse collection */
228
229     0x05, 0x01, /* USAGE_PAGE (Generic Desktop) */
230     0x09, 0x02, /* USAGE (Mouse) */
231     0xA1, 0x01, /* COLLECTION (Application) */
232     0x09, 0x01, /* USAGE (Pointer) */
233     0xA1, 0x00, /* COLLECTION (Physical) */
234     0x85, 0x02, /* REPORT_ID (2) */
235     0x05, 0x09, /* USAGE_PAGE (Button) */
236     0x19, 0x01, /* USAGE_MINIMUM (Button 1) */
237     0x29, 0x03, /* USAGE_MAXIMUM (Button 3) */
238     0x15, 0x00, /* LOGICAL_MINIMUM (0) */
239     0x25, 0x01, /* LOGICAL_MAXIMUM (1) */
240     0x75, 0x01, /* REPORT_SIZE (1) */
241     0x95, 0x03, /* REPORT_COUNT (3) */
242     0x81, 0x02, /* INPUT (Data,Var,Abs) */
243     0x75, 0x01, /* REPORT_SIZE (1) */
244     0x95, 0x05, /* REPORT_COUNT (5) */
245     0x81, 0x03, /* INPUT (Cnst,Var,Abs) */
246     0x05, 0x01, /* USAGE_PAGE (Generic Desktop) */
247     0x09, 0x30, /* USAGE (X) */
248     0x15, 0x00, /* LOGICAL_MINIMUM (0) */
249     0x26, 0xFF, 0x0F, /* LOGICAL_MAXIMUM (4095) */
250     0x35, 0x00, /* PHYSICAL_MINIMUM (0) */
251     0x46, 0xFF, 0x0F, /* PHYSICAL_MAXIMUM (4095) */
252     0x75, 0x10, /* REPORT_SIZE (16) */
253     0x95, 0x01, /* REPORT_COUNT (1) */
254     0x81, 0x02, /* INPUT (Data,Var,Abs) */
255     0x09, 0x31, /* USAGE (Y) */
256     0x15, 0x00, /* LOGICAL_MINIMUM (0) */
257     0x26, 0xFF, 0x0F, /* LOGICAL_MAXIMUM (4095) */
258     0x35, 0x00, /* PHYSICAL_MINIMUM (0) */
259     0x46, 0xFF, 0x0F, /* PHYSICAL_MAXIMUM (4095) */
260     0x75, 0x10, /* REPORT_SIZE (16) */
261     0x95, 0x01, /* REPORT_COUNT (1) */
262     0x81, 0x02, /* INPUT (Data,Var,Abs) */
263     0xC0, /* END_COLLECTION */
264     0xC0 /* END_COLLECTION */
265 };

```

Figure 2. HID mouse report descriptor

Report descriptor 1, report id 1, is used to set the relative position of the mouse. Report descriptor 2, report id 2, is used to set the absolute position of the mouse. Table 1 and Table 2 describe the data structure of the USB HID mouse corresponding to the report descriptor.

Table 1. Data format of report descriptor 1

Report 1	Descriptor
Byte 0	Report ID – Must be 0x01
Byte 1	<ul style="list-style-type: none"> • bit7: <ul style="list-style-type: none"> — 1: The change of Y coordinate exceeds the range of -256-255. — 0: No overflow • bit6: <ul style="list-style-type: none"> — 1: The change of X coordinate exceeds the range of -256-255. — 0: No overflow • bit5: Sign bit of Y coordinate <ul style="list-style-type: none"> — 1: The mouse moves to the right. • bit4: Sign bit of X coordinate. <ul style="list-style-type: none"> — 1: The mouse moves to the left. • bit3: Always 1 • bit2: <ul style="list-style-type: none"> — 1: Middle button press • bit1 <ul style="list-style-type: none"> — 1: Right click • bit0: <ul style="list-style-type: none"> — 1: Left click
Byte 2	Change value in X coordinate
Byte 3	Change value in Y coordinate
Byte 4	Wheel change

Table 2. Data format of report descriptor 1

Report 2	Descriptor
Byte 0	Report ID – Must be 0x02
Byte 1	<ul style="list-style-type: none"> • bit7: <ul style="list-style-type: none"> — 1: The change of Y coordinate exceeds the range of -256-255. — 0: No overflow • bit6:

Table continues on the next page...

Table 2. Data format of report descriptor 1 (continued)

Report 2	Descriptor
	<ul style="list-style-type: none"> — 1: The change of X coordinate exceeds the range of -256-255. — 0: No overflow • bit5: Sign bit of Y coordinate <ul style="list-style-type: none"> — 1: The mouse moves to the right. • bit4: Sign bit of X coordinate. <ul style="list-style-type: none"> — 1: The mouse moves to the left. • bit3: Always 1 • bit2: <ul style="list-style-type: none"> — 1: Middle button press • bit1 <ul style="list-style-type: none"> — 1: Right click • bit0: <ul style="list-style-type: none"> — 1: Left click
Byte 2	Low 8 bits of X coordinate
Byte 3	High 8 bits of X coordinate
Byte 4	Low 8 bits of Y coordinate
Byte 5	High 8 bits of Y coordinate

4 UART to HID mouse protocol

This section introduces the implementation of the UART to HID mouse commands, including two commands to set the absolute position and relative position of the mouse.

Table 3. UART to HID mouse commands

Command	Description
CMD_SEND_MS_ABS_DATA	Set absolute position of the mouse
CMD_SEND_MS_REL_DATA	Set relative position of the mouse

4.1 Set absolute position of the mouse

Use the `CMD_SEND_MS_ABS_DATA` command to set the absolute position of the mouse. [Table 4](#) describes the corresponding UART data format.

Table 4. Data format of CMD_SEN_MS_ABS_DATA command

HEAD	ADDR	CMD	LEN	DATA field	SUM ¹
0x57, 0xAB	0x00	0x04	7	7-byte data	0x?

1. SUM = HEAD + ADDR + CMD + LEN + DATA

This command has a 7-byte data field. [Table 5](#) describes the specific content of data field.

Table 5. Data field of CMD_SEND_MS_ABS_DATA command

Data field	Description
Byte 0	Must be 0x02
Byte 1	<ul style="list-style-type: none"> • Bit 0: Left button • Bit 1: Right button • Bit 2: Middle button • Bit 3 - Bit 7: 0 • Bit 0 - Bit2: <ul style="list-style-type: none"> — 1: The button is pressed. — 0: The button is released or not pressed.
Byte 2, 3	Two bytes of X-axis coordinate value. The high byte follows the low byte.
Byte 4, 5	Two bytes of Y-axis coordinate value. The high byte follows the low byte.
Byte 6	Wheel variation <ul style="list-style-type: none"> • 0: The wheel does not move. • 0x01-0x7F: The wheel scrolls up. • 0x81-0xFF: The wheel scrolls down.

After receiving the complete UART command from KE15Z, K20 sends a response to KE15Z. [Table 6](#) describes the data format of response.

Table 6. The response of CMD_SEND_MS_ABS_DATA command

HEAD	ADDR	CMD	LEN	DATA	SUM
0x57, 0xAB	0x00	0x84	1	1-byte data	0x?

The returned 1-byte data field indicates the current command execution status, which can be defined by the user.

The default absolute mouse resolution in the USB HID descriptor is 4096 × 4096. When KE15Z sends the absolute position in the X and Y coordinates, it needs to be calculated according to the actual screen resolution, and then sends the calculated value.

For example, the current screen resolution is: X_MAX(1920) × Y_MAX(1080). To move the mouse to the point (100,100), you need to perform the following calculation:

$X1 = (4096 \times 100) / X_MAX;$ $Y1 = (4096 \times 100) / Y_MAX;$

The calculation results are as follows:

$X1 = (100 \times 4096) / 1920 = 213 = 0xD5$ $Y1 = (100 \times 4096) / 1080 = 379 = 0x17B$
--

The KE15Z sends the UART data packet, 0x57 0xAB 0x00 0x04 0x07 0x02 0x00 0xD5 0x00 0x7B 0x01 0x00 0x60, to K20. After receiving the UART command, K20 parses the UART data and sends it to the PC according to the HID mouse data format to implement the function of setting the absolute position of the mouse. At the same time, K20 responds to KE15Z with data of 0x57 0xAB 0x00 0x84 0x01 0x00 0x87 through UART interface to tell the KE15Z that the UART command is successfully received.

4.2 Set relative position of the mouse

Use the `CMD_SEND_MS_REL_DATA` command to set the relative position of the mouse. Table 7 describes the corresponding UART data format.

Table 7. Data format of `CMD_SEN_MS_REL_DATA` command

HEAD	ADDR	CMD	LEN	DATA	SUM
0x57, 0xAB	0x00	0x05	5	5-byted data	0x?

This command has a 5-byte data field. Table 8 describes the specific content of data field.

Table 8. Data field of `CMD_SEND_MS_REL_DATA` command

Data field	Description
Byte 0	Must be 0x01
Byte 1	<ul style="list-style-type: none"> • Bit 0: Left button • Bit 1: Right button • Bit 2: Middle button • Bit 3 - Bit 7: 0 • Bit 0 - Bit 2: <ul style="list-style-type: none"> — 1: The button is pressed. — 0: The button is released or not pressed.
Byte 2	1 byte of pixels moved in X direction
Byte 3	1 byte of pixels moved in Y direction
Byte 4	Wheel variation <ul style="list-style-type: none"> • 0: The wheel does not move. • 0x01-0x7F: Wheel scrolls up. • 0x81-0xFF: Wheel scrolls down.

After receiving the complete UART command from KE15Z, K20 sends a response to KE15Z. Table 9 describes the data format of response.

Table 9. The response of `CMD_SEND_MS_REL_DATA` command

HEAD	ADDR	CMD	LEN	DATA	SUM
0x57, 0xAB	0x00	0x85	1	1-byte data	0x?

Here is an example of how to set the mouse to move three pixels to the left.

1. KE15Z sends UART data, 0x57 0xAB 0x00 0x05 0x05 0x01 0x00 0xFD 0x00 0x00 0x0A, to K20.
2. After receiving the UART command, K20 parses the UART data and sends it to the PC according to the HID mouse data format to implement the function of setting the relative position of the mouse.
3. At the same time, K20 responds to KE15Z with the data of 0x57 0xAB 0x00 0x85 0x01 0x00 0x88 through UART interface to tell the KE15Z that the UART command has been successfully received.

5 Function test

This section describes how to control the relative position and absolute position of the mouse through the two touch buttons on the FRDM-KE15Z board.

The steps are as follows:

1. Download the *K20_UART_to_HID_Mouse* project through K20's JTAG interface, J11.
2. Open the *tsi_v5_selfmode* example in the KE15Z SDK and add the function of sending `CMD_SEND_MS_ABS_DATA` and `CMD_SEND_MS_REL_DATA` commands via UART (add the code in [Figure 3](#) to the *main.c* file of the *tsi_v5_selfmode* project).


```

25  /******
26  * Variables
27  *****/
28  /* Move the mouse three pixels to the left */
29  uint8_t gMouseRelDataLeft[] = {0x57, 0xAB, 0x00, 0x05, 0x05, 0x01, 0x00, 0xFD, 0x00, 0x00, 0x0A};
30
31  /* Set the absolute position of the mouse to (100,100) */
32  uint8_t gMouseAbsData[] = {0x57, 0xAB, 0x00, 0x04, 0x07, 0x02, 0x00, 0xD5, 0x00, 0x7B, 0x01, 0x00, 0x60};
33  /******
34  * Prototypes
35  *****/
36
37  /******
38  * Code
39  *****/
40  void Send_Mouse_Rel_Data_CMD(void)
41  {
42      LPUART_WriteBlocking(LPUART1, (const uint8_t *)gMouseRelDataLeft, sizeof(gMouseRelDataLeft));
43  }
44
45  void Send_Mouse_Abs_Data_CMD(void)
46  {
47      LPUART_WriteBlocking(LPUART1, (const uint8_t *)gMouseAbsData, sizeof(gMouseAbsData));
48  }
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71  while (1)
72  {
73      /* debug key function */
74      key_event = TSI_KeyDetect(&touched_key_id);
75
76      if (key_event == kKey_Event_Touch)
77      {
78          LED1_ON();
79
80          if (touched_key_id == 0)
81          {
82              /* Move the mouse three pixels to the left */
83              Send_Mouse_Rel_Data_CMD();
84          }
85
86          if (touched_key_id == 1)
87          {
88              /* Set the absolute position of the mouse to (100,100) */
89              Send_Mouse_Abs_Data_CMD();
90          }
91      }
92      else if (key_event == kKey_Event_Release)
93      {
94          /* if (touched_key_id == 0) */
95          {
96              LED1_OFF();
97          }
98      }
99  }
100  else
101  {
102      /* Here to deal with other key events */
103      key_event = kKey_Event_Idle;
104  }
105  }
106  }

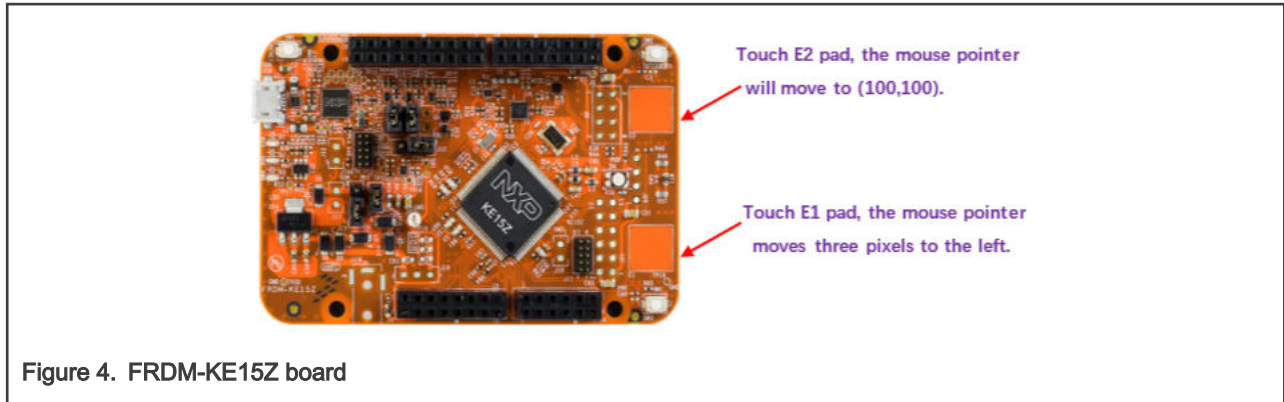
```

Figure 3. Add the function of sending UART HID command in tsi_v5_selfmode example

Then compile the project and download the program to KE15Z through the SWD interface, J17.

3. Reset the board.

- Press the E1 touch button, and the mouse pointer on the PC will move three pixels to the left.
- Press the E2 touch button, and the mouse pointer on the PC will move to (100,100).



Users can also connect their own touchpads to the FRDM-KE15Z board and use the function of UART to HID Mouse to display their demos more intuitively. For more details on the function implement, see [AN13079SW](#).

6 Reference

- [CH9329 Data sheet](#)
- *Kinetis KE1xZ with up to 256 KB Flash* (document [KE1xZP100M72SF0](#))
- *K20 Sub-Family Reference Manual* (document [K20P32M50SF0RM](#))

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, 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, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D 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.

© NXP B.V. 2020.

All rights reserved.

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

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

Date of release: 2 December, 2020

Document identifier: AN13079

