Using Nu-Link2 to Access MCU on Linux

Example Code Introduction for NuMicro® Family

Information

Application	On Linux platform, this example code uses the HIDAPI library to connect Nu-Link2 and access NuMicro series microcontrollers (MCUs).
Linux Version	Ubuntu 20.04.3 LTS
Hardware	Nu-Link2 (Nu-Link2-Pro or Nu-Link2-Me)

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

This example code demonstrates the use of HIDAPI to connect Nu-Link2 on Linux, connect to the target microcontroller (MCU) through Nu-Link2, perform chip erase, and read/write operations on Flash and configuration area.

The HIDAPI library used in the example supports two back-ends (hidraw and libusb) on Linux platform. For details, please refer to HIDAPI official website. When compiling this example code, you can choose to use hidraw or libusb back-end in the compilation options.

The example code supports 8-bit (8051) and 32-bit (Arm Cortex-m) target microcontroller access. You can choose to use 8-bit or 32-bit access in the compilation options.

1.1 Principle

In Figure 1-1, the example code uses the ICE command set which is a private command based on the USB HID protocol. Through this command set, the target microcontroller can be connected and controlled.

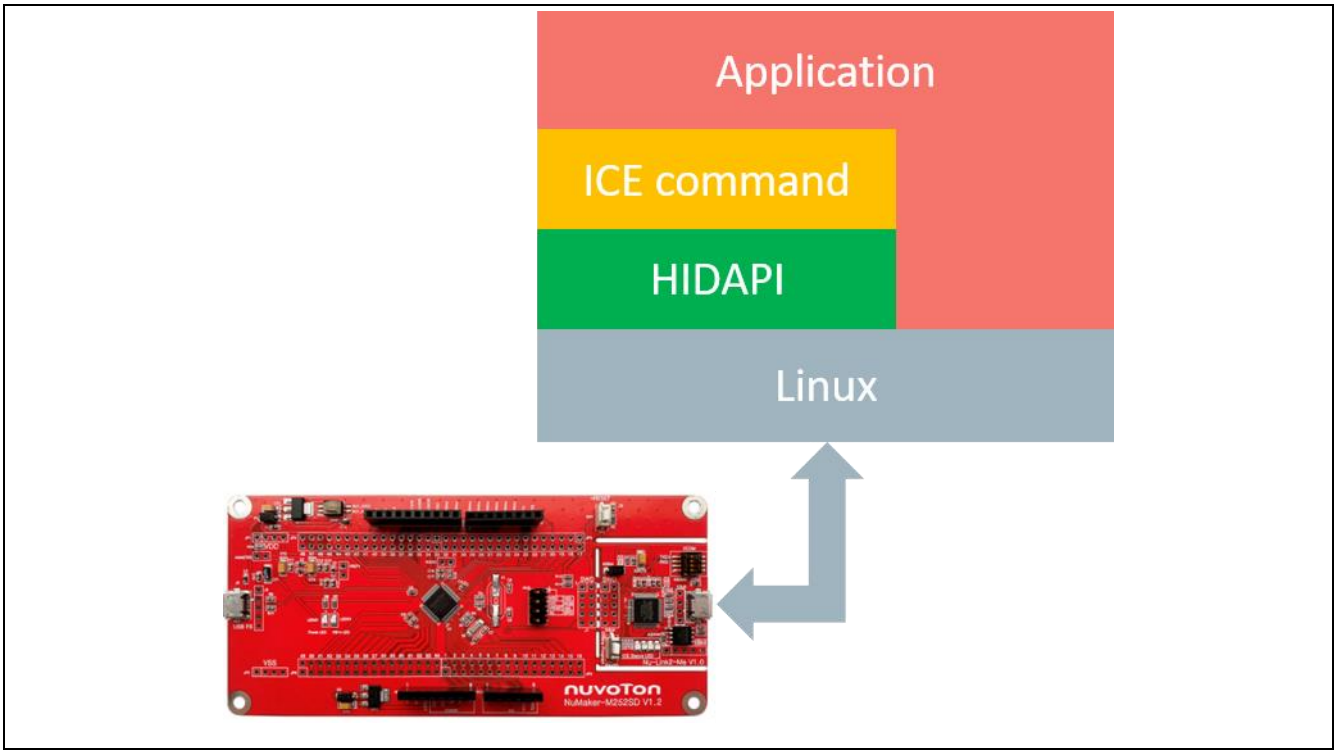

Figure 1-2 Software Architecture

1.2 Demo Result

The demo result is shown below.

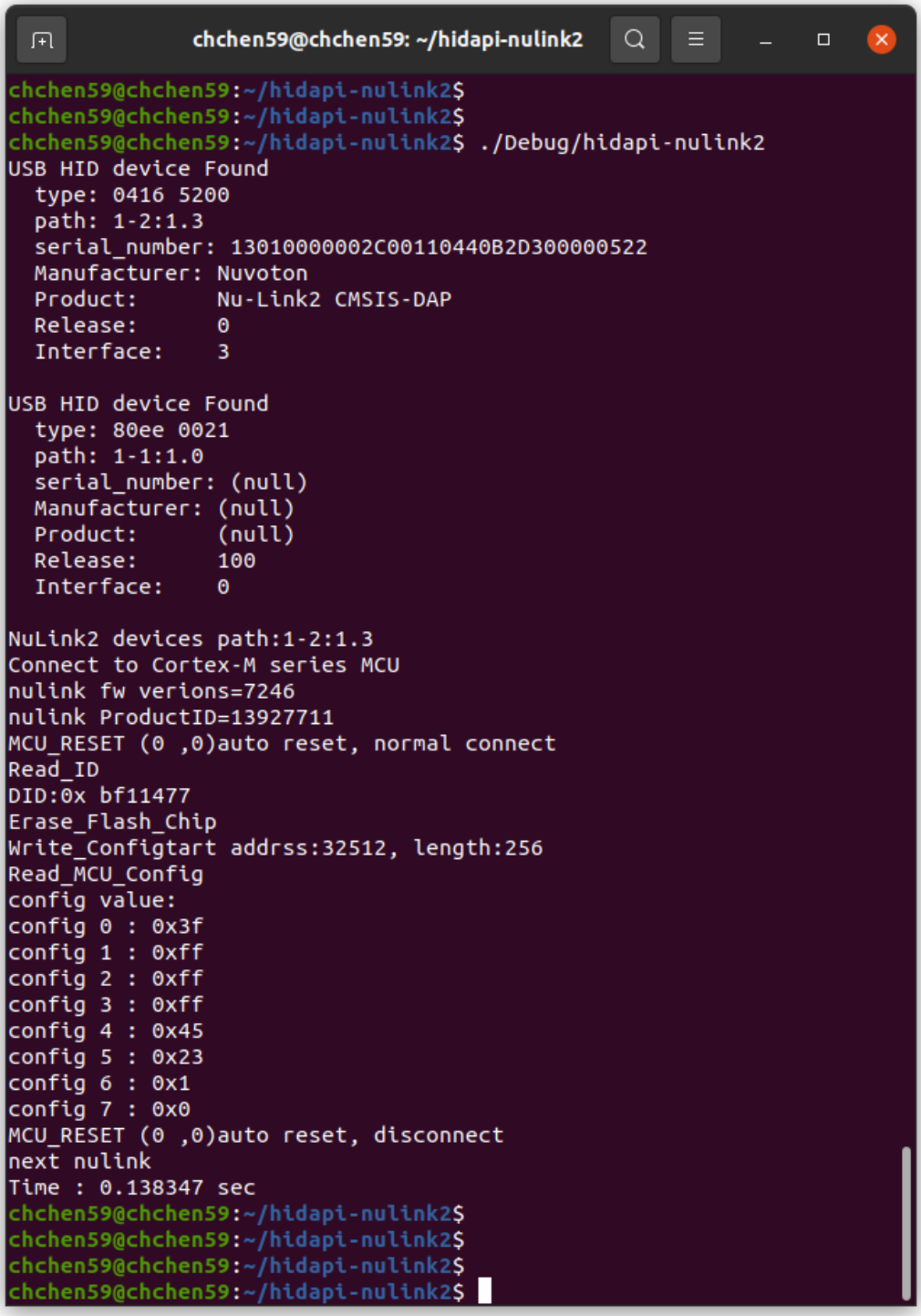

Figure 1-3 Demo Result

2 Code Description

Set the Nu-Link2 USB VID/PID number and the series of target microcontroller:

```
#define NU_LINK_USB_VID 0x0416
#define NU_LINK_USB_PID 0x5200

typedef enum
{
    eNUC_CHIP_TYPE_GENERAL_V6M= 0,
    eNUC_CHIP_TYPE_NUC1XX     = 1,
    eNUC_CHIP_TYPE_M05X       = 3,
    eNUC_CHIP_TYPE_N572       = 4,
    eNUC_CHIP_TYPE_NANO100        = 0x101,
    eNUC_CHIP_TYPE_MINI51     = 0x102,
    eNUC_CHIP_TYPE_MT5XX      = 0x103,
    eNUC_CHIP_TYPE_N570       = 0x104,
    eNUC_CHIP_TYPE_NM1120     = 0x105,
    eNUC_CHIP_TYPE_I96000     = 0x10C,
    eNUC_CHIP_TYPE_AU91XX     = 0x114,
    eNUC_CHIP_TYPE_M0564      = 0x121,
    eNUC_CHIP_TYPE_M031       = 0x10A,
    eNUC_CHIP_TYPE_M0A21      = 0x10E,
    eNUC_CHIP_TYPE_M030G      = 0x10F,
    eNUC_CHIP_TYPE_M031G      = 0x12A,
    eNUC_CHIP_TYPE_M251       = 0x201,
    eNUC_CHIP_TYPE_M2351      = 0x321,
    eNUC_CHIP_TYPE_M2354      = 0x322,
    eNUC_CHIP_TYPE_GENERAL_V7M= 0x400,
    eNUC_CHIP_TYPE_NUC4XX     = 0x401,
    eNUC_CHIP_TYPE_M451       = 0x402,
    eNUC_CHIP_TYPE_M471       = 0x40A,
    eNUC_CHIP_TYPE_M480       = 0x411,
    eNUC_CHIP_TYPE_I94000     = 0x414,
    eNUC_CHIP_TYPE_NUC505     = 0x500,
    eNUC_CHIP_TYPE_GENERAL_ARM7   = 0x700,
    eNUC_CHIP_TYPE_GENERAL_1T = 0x800, // For 8051 1T N76/ML51/MS51 Series
    eNUC_CHIP_TYPE_GENERAL_855= 0x801, // 85J, 37J
    eNUC_CHIP_TYPE_GENERAL_82J= 0x802,
    eNUC_CHIP_TYPE_GENERAL_83J= 0x803,
    eNUC_CHIP_TYPE_GENERAL_35J= 0x804,
    eNUC_CHIP_TYPE_GENERAL_87J= 0x805, // 87J, 23J
```

```
    eNUC_CHIP_TYPE_GENERAL_21J= 0x806,
    eNUC_CHIP_TYPE_GENERAL_658= 0x807,
    eNUC_CHIP_TYPE_GENERAL_659= 0x808,
    eNUC_CHIP_TYPE_GENERAL_ARM9    = 0x900
}E_NUC_CHIP_TYPE;

//Target chip type
#if (TARGET == TARGET_CORTEX_M)
E_NUC_CHIP_TYPE s_eNUCChipType = eNUC_CHIP_TYPE_M251;
#else
E_NUC_CHIP_TYPE s_eNUCChipType = eNUC_CHIP_TYPE_GENERAL_1T;
#endif
```

Enumerate USB device and identify the Nu-Link2 device node:

```
    //enum nulink 2, to compare vendor_id == 0x0416, and product_id==0x5200 and record
devices path
    char dest_path[MAX_NULINK2_COUNT][64];
    struct hid_device_info *devs, *cur_dev;
    devs = hid_enumerate(0x0, 0x0);
    cur_dev = devs;
    int device_count = 0;
    while (cur_dev) {
        printf("Device Found\n  type: %04hx %04hx\n  path: %s\n  serial_number: %ls",
            cur_dev->vendor_id, cur_dev->product_id, cur_dev->path, cur_dev-
>serial_number);
        printf("\n");
        printf("  Manufacturer: %ls\n", cur_dev->manufacturer_string);
        printf("  Product:      %ls\n", cur_dev->product_string);
        printf("  Release:      %hx\n", cur_dev->release_number);
        printf("  Interface:    %d\n", cur_dev->interface_number);
        printf("\n");



        if ((cur_dev->vendor_id == NU_LINK_USB_VID)&&(cur_dev->product_id ==
NU_LINK_USB_PID))
        {

            strcpy(dest_path[device_count], cur_dev->path);  //copy path name to buffer.
            device_count++;
        }
        cur_dev = cur_dev->next;
    }
```

```
    hid_free_enumeration(devs);
```

Target microcontroller connections:

```
int connect_target_chip(void)
{
#if (TARGET == TARGET_CORTEX_M)
    //command flow
    //Reset command serial
    if (ResetCmdSerial() != 0)
        return -1;
    //set config nulink
    if (Set_Config_NULINK() != 0)
        return -2;
    //mode entry for 32bit mcu
    if (MCU_RESET_CONNECT_32() != 0)
        return -3;

    //Read target chip DID
    if (Read_ID_32() != 0) //READ device id
        return -4;

#else
    //Reset command serial
    if (ResetCmdSerial() != 0)
        return -1;
    //set config nulink
    if (Set_Config_NULINK() != 0)
        return -2;
    //mode entry for 8bit mcu
    if (Mode_Entry() != 0)
        return -3;
    //entry icp mode
    if (ICP_Mode_Entry() != 0)
        return -4;
    //Read target chip UID, DID
    if (Read_ID() != 0) //READ device id
        return -5;

#endif
    return 0;
}
```

Whole chip erase:

```
int Erase_Flash_Chip(void)
{
    printf("Erase_Flash_Chip\n\r");
    int res;
    int false_flag = 0;
    unsigned int local_retry = 0;
    memset(USBBUF_OUT, 0x0, NULINK_PACKAGE_LENGTH + 1); //clear out buffer and report id
    memset(USBBUF_IN, 0X0, NULINK_PACKAGE_LENGTH);//clear in buffer
    USBBUF_OUT[1] = s_u8Index;
    set_out_length(2 * 4);
    outpw(&USBBUF_OUT[4], 0xA4); //command
    outpw(&USBBUF_OUT[4 + 4], 0X00);//Reset_Options:

    USBBUF_OUT[0] = 0;//usb report id, always is 0
    res = hid_write(s_pHandle, USBBUF_OUT, 513);
    while (1)
    {
        res = hid_read(s_pHandle, USBBUF_IN, 512);
        if (USBBUF_IN[0] == s_u8Index&&(res == 512))
            break;
        if (local_retry > USB_TRY_CNT)
        {
            false_flag = 1;
            return false_flag;
        }
    }

    unsigned int return_command;
    return_command = inpw(&USBBUF_IN[3]);
    dbgprint("nulink return_command=0x%x\n\r", return_command);
    index_inc();
    return false_flag;
}
```

Flash access (must execute whole chip erase first):

```
int flash_access(void)
{
    //clear read memory buffer, and set write buffer
```

```
    //for testing, flash content is a fixed pattern
    //you can put your flash content to s_au8MemFlashArray
    for (int i = 0; i < MEMORY_SIZE_TEST; i++)
    {
        s_au8MemFlashArrayRead[i] = 0;
        s_au8MemFlashArray[i] = i & 0xff;
    }
    //program flash
    for (unsigned int i = 0; i < MEMORY_SIZE_TEST; i = i + 32)
    {
        Write_Flash(i, PAGE_PROGRAM_SIZE, &s_au8MemFlashArray[i]);

    }
    //read flash context
    for (unsigned int i = 0; i < MEMORY_SIZE_TEST; i = i + 256)
    {
        Read_Flash(i, READ_SIZE, &s_au8MemFlashArrayRead[i]);
    }
    //compare memory
    for (int i = 0; i < MEMORY_SIZE_TEST; i++)
    {
        if (s_au8MemFlashArrayRead[i] != s_au8MemFlashArray[i])
        {
            printf("error =%d\n\r", i);
            break;
        }
    }

    return 0;
}
```

Configuration area access:

```
int config_access(void)
{
    //set config buffer

#if (TARGET == TARGET_CORTEX_M)
    //config 0, 32bit, Low byte first
    s_au8ConfigArray[0] = 0x3f; //see trm doc for config value setting
    s_au8ConfigArray[1] = 0xff;
    s_au8ConfigArray[2] = 0xff;
    s_au8ConfigArray[3] = 0xff;
```

```
    //config 1, 32bit, Low byte first
    s_au8ConfigArray[4] = 0x45; //see trm doc for config value setting
    s_au8ConfigArray[5] = 0x23;
    s_au8ConfigArray[6] = 0x01;
    s_au8ConfigArray[7] = 0x00;

    Write_Config_32(s_au8ConfigArray);

    //read config from buffer
    Read_MCU_Config_32(s_au8ConfigArrayRead);
    printf("config value:\n\r");
    for (int i = 0; i < 8; i++)
    {
        printf("config %d : 0x%x\n\r", i, s_au8ConfigArrayRead[i]);
        if (s_au8ConfigArrayRead[i] != s_au8ConfigArray[i])
            printf("error =%d\n\r", i);
    }

#else
    s_au8ConfigArray[0] = 0x6f; //see trm doc for config value setting
    s_au8ConfigArray[1] = 0xff;
    s_au8ConfigArray[2] = 0xff;
    s_au8ConfigArray[3] = 0xff;
    s_au8ConfigArray[4] = 0xff;
    Write_Config(s_au8ConfigArray);

    //read config from buffer
    Read_MCU_Config(s_au8ConfigArrayRead);
    printf("config value:\n\r");
    for (int i = 0; i < 5; i++)
    {
        printf("config %d : 0x%x\n\r", i, s_au8ConfigArrayRead[i]);
        if (s_au8ConfigArrayRead[i] != s_au8ConfigArray[i])
            printf("error =%d\n\r", i);
    }

#endif

    return 0;
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- Linux version
  - Ubuntu 20.04.3 LTS
- HIDAPI
  - https://github.com/libusb/hidapi

3.2 Hardware Requirements

- Nu-Link2 (Nu-Link2-Pro or Nu-Link2-Me)
- Evaluation board (e.g. NuMaker-IoT-M252, NuMaker-ML56SD)

4 Directory Information

The directory structure is shown below.

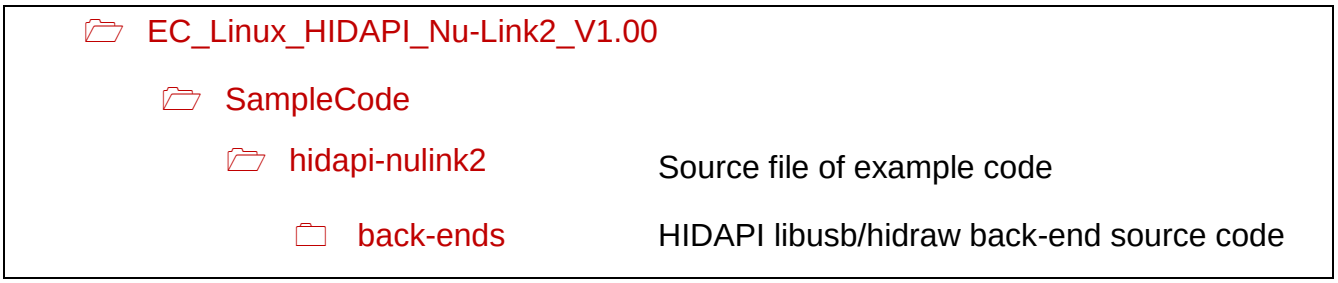

Figure 4-1 Directory Structure

5 Example Code Execution

1. Install related tools and libraries.

   - Compilation tool
     - #sudo apt install autotools-dev autoconf automake libtool
   - USB-related libraries
     - #sudo apt install libudev-dev libusb-1.0-0-dev

2. Since the write function of Nu-Link2 ICE commands will be used, the write permission of the Nu-Link2 HID device node needs to be enabled. The opening of write permission uses udev for dynamic device management, so user needs to add udev rules for Nu-Link2. There is a default udev rule file (nulink2.rules) in the hidapi-nulink2 folder. Make sure that the idVendor and idProduct settings in this file are consistent with the USB VID/PID of Nu-Link2, and finally copy this rule file to /etc/udev/rules.d directory.

   - After connecting to Nu-Link2, use lsusb to query the information of Nu-Link2.
   - Open the rule file to confirm USB VID/PID consistency.

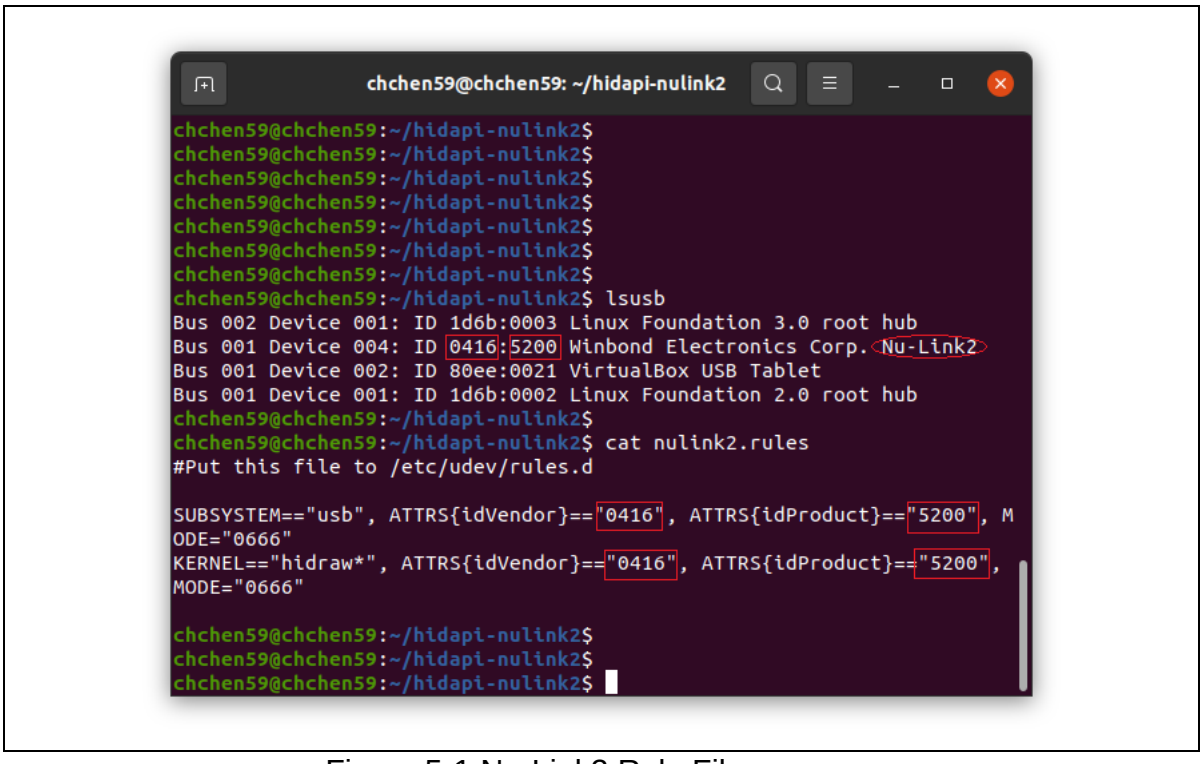

Figure 5-1 Nu-Link2 Rule File

3. Enter the hidapi-nulink2 folder according to the directory information chapter, and execute the compilation script.

   - Compilation option
     - -b libusb/hidraw: Select HIDAPI back-end. Default is libusb.
     - -t 8051/cortex-m: Select the target microcontroller type. Default is cortex-m.
   - Compile
     - #./build.sh –b libusb –t cortex-m

4. After connecting Nu-Link2 to the evaluation board, run this example code in “Debug” directory.

   - #./hidapi-nulink2

6 Revision History

Date	Revision	Description
2021.11.09	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.