MSD with SPI mode SD Card and HID Mouse

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code accesses SD card by the SPI interface for USB mass storage device medium and support USB HID mouse function.
BSP Version	NUC200 Series BSP CMSIS V3.00.003
Hardware	NuTiny-EVB-NUC240-LQFP100 V1.00

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 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 Function Description

1.1 Introduction

An USB Composite Device is a peripheral device that supports more than one device class at the same time. The sample code consists of Mass storage class and HID class in one USB device. The SD card is accessed by the SPI interface for USB storage medium. The PB.15 is used to select content of HID report for report test.

1.2 Principle

1.2.1 SPI mode SD Card

SD card support two operating modes, SD card mode and SPI mode. The SD card SPI mode is compatible with SPI hosts and the signals are defined in Table 1.

Pin	SD Mode			SPI Mode		
	Name	Type	Description	Name	Type	Description
1	CD/DAT3	I/O/PP	Card Detect/ Data Line [Bit 3]	CS	I	Chip Select (neg true)
2	CMD	PP	Command/Response	DI	PP	Data In
3	VSS	S	Supply voltage ground	VSS	S	Supply voltage ground
4	VDD	S	Supply voltage	VDD	S	Supply voltage
5	CLK	I	Clock	SCLK	I	Clock
6	VSS	S	Supply voltage ground	VSS	S	Supply voltage ground
7	DAT0	I/O/PP	Data Line [Bit 0]	DO	O/PP	Data Out
8	DAT1	I/O/PP	Data Line [Bit 1]	RSV		
9	DAT2	I/O/PP	Data Line [Bit 2]	RSV		

Table 1 Pin definition

This sample code provides a set of source code to support the SPI mode SD card commands and initialization flow.

1.2.2 USB Deice

This example code includes 2 interfaces that follow MSC (Mass storage class) and HID (Human Interface Devices) specification. Based on USB spec, endpoint 0 for control transfer is used for USB enumeration. There are 2 endpoints to upload and download Bulk report and 1 endpoint to upload HID report. In this example, control pipe (endpoint 0) is made of two hardware endpoints 0 and 1 in device. The interfaces and endpoints configuration are shown in Table 2.

Function	USB Composite Device	NUC200 Hardware setting

Control Transfer	Control Pipe
	endpoint 0: Control IN/OUT
		endpoint 0: Control IN
		endpoint 1: Control OUT

Mass Storage	Interface 0
	endpoint 2: Bulk IN
	endpoint 3: Bulk OUT
		endpoint 2: Bulk IN
		endpoint 3: Bulk OUT

HID Mouse	Interface 1 endpoint 4: Interrupt IN	endpoint 4: Interrupt IN

Table 2 Configuration of composite interfaces and endpoints

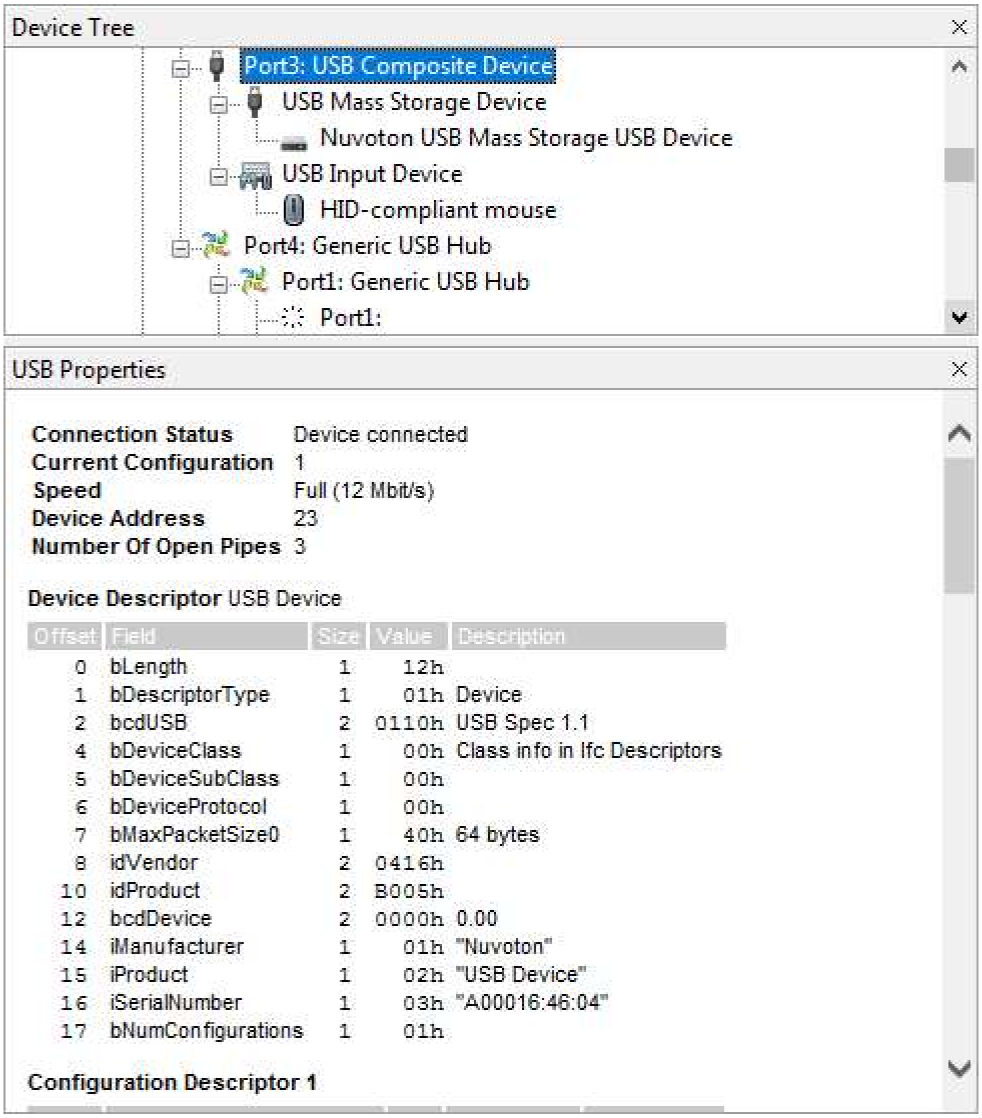
HID report descriptor is used to define HID report format and can be customized based on application demand. This example includes 1 HID report descriptors to support HID report formats for mouse listed in Table 3.

Byte	Bits	Description
0	0~2	Button 1~3
0	3~7	Padding
1	0~7	X-axis
2	0~7	Y-axis

Table 3 HID Mouse report format

1.3 Demo Result

After plug-in USB device, the USB device tree information could be got by USBLyzer in Figure 1. In this demo, the USB composite device includes 2 interfaces to support Mass Storage Device and HID mouse function.



Device Tree

- Port3: USB Composite Device
  - USB Mass Storage Device
    - Nuvoton USB Mass Storage USB Device
  - USB Input Device
    - HID-compliant mouse
- Port4: Generic USB Hub
  - Port1: Generic USB Hub
    - Port1:

USB Properties

Connection Status	Device connected
Current Configuration	1
Speed	Full (12 Mbit/s)
Device Address	23
Number Of Open Pipes	3

Device Descriptor USB Device

Offset	Field	Size	Value	Description
0	bLength	1	12h	
1	bDescriptorType	1	01h	Device
2	bcdUSB	2	0110h	USB Spec 1.1
4	bDeviceClass	1	00h	Class info in Ifc Descriptors
5	bDeviceSubClass	1	00h	
6	bDeviceProtocol	1	00h	
7	bMaxPacketSize0	1	40h	64 bytes
8	idVendor	2	0416h	
10	idProduct	2	B005h	
12	bcdDevice	2	0000h	0.00
14	iManufacturer	1	01h	"Nuvoton"
15	iProduct	1	02h	"USB Device"
16	iSerialNumber	1	03h	"A00016:46:04"
17	bNumConfigurations	1	01h	

Configuration Descriptor 1

Figure 1 Device Tree

2 Code Description

2.1 SPI Mode SD Card

Configure SPI interface and initialize SD card :

```
    uint8_t udcFlashInit(void)
    {
        RoughDelay(100000);

        if(DrvSDCARD_Open() == E_SUCCESS)
            printf("SDCard Open success\n");
        else
            printf("SDCard Open failed\n");

        if (!Flash_Identify())
            return 0;

        return 1;
    }
```

Read data to a file on the SD card by call the API - SpiRead.

```
void SpiRead(uint32_t addr, uint32_t size, uint8_t* buffer)
{
    /* This is low level read function of USB Mass Storage */
    uint32_t response;
    if(addr>=LogicSector)
    {
        DBG_PRINTF("Read illegal Sector:0x%x\n",addr);
        return;
    }
    if(SDtype&SDBlock)
    {
        while(size >= PHYSICAL_BLOCK_SIZE)
        {
            while(MMC_Command_Exec(READ_SINGLE_BLOCK,addr,buffer,&response)==FALSE);
            addr   ++;
            buffer += PHYSICAL_BLOCK_SIZE;
            size   -= PHYSICAL_BLOCK_SIZE;
        }
    }
    else
    {
        addr*=PHYSICAL_BLOCK_SIZE;
        while(size >= PHYSICAL_BLOCK_SIZE)
        {
            while(MMC_Command_Exec(READ_SINGLE_BLOCK,addr,buffer,&response)==FALSE);
            addr   += PHYSICAL_BLOCK_SIZE;
            buffer += PHYSICAL_BLOCK_SIZE;
            size   -= PHYSICAL_BLOCK_SIZE;
        }
    }
}
```

Write data to a file on the SD card by call the API - SpiWrite.

```
void SpiWrite(uint32_t addr, uint32_t size, uint8_t* buffer)
{
    uint32_t response;
    if(addr>=LogicSector)
    {
        DBG_PRINTF("Write illegal Sector:0x%x\n",addr);
        return;
    }
    if(SDtype&SDBlock)
    {
        while(size >= PHYSICAL_BLOCK_SIZE)
        {
            while(MMC_Command_Exec(WRITE_BLOCK,addr,buffer,&response)==FALSE);
            addr   ++;
            buffer += PHYSICAL_BLOCK_SIZE;
            size   -= PHYSICAL_BLOCK_SIZE;
        }
    }
    else
    {
        addr*=PHYSICAL_BLOCK_SIZE;
        while(size >= PHYSICAL_BLOCK_SIZE)
        {
            while(MMC_Command_Exec(WRITE_BLOCK,addr,buffer,&response)==FALSE);
            addr   += (PHYSICAL_BLOCK_SIZE);
            buffer += PHYSICAL_BLOCK_SIZE;
            size   -= PHYSICAL_BLOCK_SIZE;
        }
    }
}
```

2.2 Mouse

USB Host issues IN token to request the data from the endpoint 4. While IN token received, EP4_Handler () will be executed to setup the payload and then send data.

```
void EP4_Handler(void)
{
    uint8_t *buf;
    buf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP4));

    buf[0] = 0x00;
    if(PB15 == 0)
        buf[1] = 0x01;
    else
        buf[1] = 0x00;
    buf[2] = 0x00;
    USBD_SET_PAYLOAD_LEN(EP4, 3);
}
```

2.3 MSC & HID Class Request

In the function of MSC_ClassRequest(), it includes MSC class-specific and HID class-specific requests.

```
void MSC_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if(buf[0] & EP_INPUT)    /* request data transfer direction */
    {
        /* Device to host */
        switch(buf[1])
        {
        case GET_MAX_LUN:
        {
```

```
        if((buf[4] == gsInfo.gu8ConfigDesc[LEN_CONFIG + 2]) &&
             (buf[2] + buf[3] + buf[6] + buf[7] == 1))
        {
            M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = 0;
            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 1);
            /* Status stage */
            USBD_PrepareCtrlOut(0, 0);
        }
        else
            USBD_SET_EP_STALL(EP1); /* Stall when wrong parameter */
        break;
    }
    case GET_REPORT:
    case GET_IDLE:
    case GET_PROTOCOL:
    default:
    {
        /* Setup error, stall the device */
        USBD_SetStall(0);
        DBG_PRINTF("Unknow MSC req(0x%x). stall ctrl pipe\n", buf[1]);
        break;
    }
    }
}
else
{
    /* Host to device */
    switch(buf[1])
    {
    case BULK_ONLY_MASS_STORAGE_RESET:
    {
        /* Check interface number with cfg descriptor and check wValue = 0,
           wLength = 0 */
        if((buf[4] == gsInfo.gu8ConfigDesc[LEN_CONFIG + 2]) &&
            (buf[2] + buf[3] + buf[6] + buf[7] == 0))
        {
            g_u32Length = 0; /* Reset all read/write data transfer */
            USBD_LockEpStall(0);
            /* Clear ready */
            USBD->EP[EP2].CFGP |= USBD_CFGP_CLRRDY_Msk;
            USBD->EP[EP3].CFGP |= USBD_CFGP_CLRRDY_Msk;

            /* Prepare to receive the CBW */
            g_u8EP3Ready = 0;
            g_u8BulkState = BULK_CBW;
            USBD_SET_DATA1(EP3);
            USBD_SET_EP_BUF_ADDR(EP3, g_u32BulkBuf0);
            USBD_SET_PAYLOAD_LEN(EP3, 31);
        }
        else
        {
            /* Stall when wrong parameter */
            USBD_SET_EP_STALL(EP1);
        }
        /* Status stage */
        USBD_SET_DATA1(EP0);
        USBD_SET_PAYLOAD_LEN(EP0, 0);
        break;
    }
    case SET_REPORT:
    {
        if(buf[3] == 3)
        {
            /* Request Type = Feature */
            USBD_SET_DATA1(EP1);
            USBD_SET_PAYLOAD_LEN(EP1, 0);
        }
        break;
    }
    case SET_IDLE:
    {
        /* Status stage */
        USBD_SET_DATA1(EP0);
        USBD_SET_PAYLOAD_LEN(EP0, 0);
        EP4_Handler();
        break;
    }
    case SET_PROTOCOL:
    default:
        {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            DBG_PRINTF("Unknow MSC req (0x%x). stall ctrl pipe\n", buf[1]);
            break;
        }
        }
    }
}
```

2.4 Device Descriptor

2.4.1 HID Report Descriptor

HID_MouseReportDescriptor array includes the HID Report Descriptor for mouse function. It defines the data format in Table 3 and contains 3 buttons, X-axis, and Y-axis.

```
const uint8_t HID_MouseReportDescriptor[] =
{
    0x05, 0x01,     /* Usage Page(Generic Desktop Controls) */
    0x09, 0x02,     /* Usage(Mouse) */
    0xA1, 0x01,     /* Collection(Application) */
    0x09, 0x01,     /* Usage(Pointer) */
    0xA1, 0x00,     /* Collection(Physical) */
    0x05, 0x09,     /* Usage Page(Button) */
    0x19, 0x01,     /* Usage Minimum(0x1) */
    0x29, 0x03,     /* Usage Maximum(0x3) */
    0x15, 0x00,     /* Logical Minimum(0x0) */
    0x25, 0x01,     /* Logical Maximum(0x1) */
    0x95, 0x03,     /* Report Count(0x3) */
    0x75, 0x01,     /* Report Size(0x1) */
    0x81, 0x02,     /* Input(3 button bit) */
    0x95, 0x01,     /* Report Count(0x1) */
    0x75, 0x05,     /* Report Size(0x5) */
    0x81, 0x01,     /* Input(5 bit padding) */
    0x05, 0x01,     /* Usage Page(Generic Desktop Controls) */
    0x09, 0x30,     /* Usage(X) */
    0x09, 0x31,     /* Usage(Y) */
    0x15, 0x81,     /* Logical Minimum(0x81)(-127) */
    0x25, 0x7F,     /* Logical Maximum(0x7F)(127) */
    0x75, 0x08,     /* Report Size(0x8) */
    0x95, 0x02,     /* Report Count(0x2) */
    0x81, 0x06,      /* Input(1 byte) */
    0xC0,            /* End Collection */
    0xC0,            /* End Collection */
};
```

2.4.2 Configuration Descriptor for 2 Interfaces Composite Device

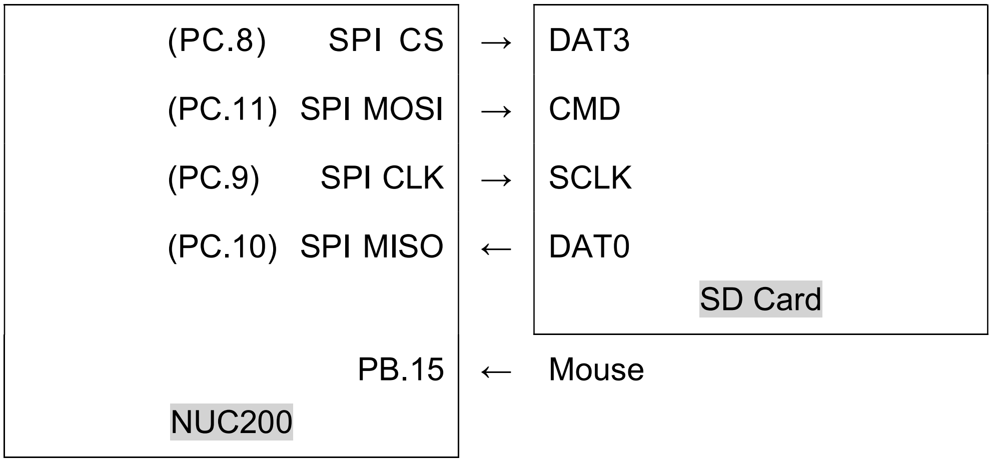
USB host requests configuration descriptor for device enumeration. gu8ConfigDescriptor is the configuration descriptor which contains description of 2 interfaces. Field “bNumInterfaces” be set as 2 to inform that there are 2 interfaces in the composite device. The descriptor is listed sequentially in following.

```
/*!<USB Configure Descriptor */
const uint8_t gu8ConfigDescriptor[] =
{
   /* Configuration Descriptor */
    LEN_CONFIG,                     /* bLength             */
    DESC_CONFIG,                    /* bDescriptorType     */
    LEN_TOTAL & 0x00FF,             /* wTotalLength        */
    (LEN_TOTAL & 0xFF00) >> 8,
    0x02,                           /* bNumInterfaces      */
    0x01,                           /* bConfigurationValue */
    0x00,                           /* iConfiguration      */
    0xC0,                           /* bmAttributes        */
    0x32,                           /* MaxPower            */

    /* Interface Descriptor */
    LEN_INTERFACE,                  /* bLength             */
    DESC_INTERFACE,                 /* bDescriptorType     */
    0x00,                           /* bInterfaceNumber    */
    0x00,                           /* bAlternateSetting   */
    0x02,                           /* bNumEndpoints       */
    0x08,                           /* bInterfaceClass     */
    0x05,                           /* bInterfaceSubClass  */
    0x50,                           /* bInterfaceProtocol  */
    0x00,                           /* iInterface          */

    /* Endpoint Descriptor */
    LEN_ENDPOINT,                   /* bLength             */
    DESC_ENDPOINT,                  /* bDescriptorType     */
    (BULK_IN_EP_NUM | EP_INPUT),    /* bEndpointAddress    */
    EP_BULK,                        /* bmAttributes        */
    EP2_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize      */
    0x00,                           /* bInterval          */

    /* Endpoint Descriptor */
    LEN_ENDPOINT,                   /* bLength            */
    DESC_ENDPOINT,                  /* bDescriptorType    */
    BULK_OUT_EP_NUM,                /* bEndpointAddress   */
    EP_BULK,                        /* bmAttributes       */
    EP3_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize     */
    0x00,                           /* bInterval          */

    /* Interface Descriptor */
    LEN_INTERFACE,                  /* bLength            */
    DESC_INTERFACE,                 /* bDescriptorType    */
    0x01,                           /* bInterfaceNumber   */
    0x00,                           /* bAlternateSetting  */
    0x01,                           /* bNumEndpoints      */
    0x03,                           /* bInterfaceClass    */
    0x01,                           /* bInterfaceSubClass */
    HID_MOUSE,                      /* bInterfaceProtocol */
    0x00,                           /* iInterface         */

    /* HID Descriptor */
    LEN_HID,                        /* bLength            */
    DESC_HID,                       /* bDescriptorType    */
    0x10, 0x01,                     /* bcdHID             */
    0x00,                           /* bCountryCode       */
    0x01,                           /* bNumDescriptors    */
    DESC_HID_RPT,                   /* DbDescriptorType   */
    LEN_REPORT_DESC & 0x00FF,       /* wDescriptorLength  */
    (LEN_REPORT_DESC & 0xFF00) >> 8,

    /* Endpoint Descriptor */
    LEN_ENDPOINT,                   /* bLength            */
    DESC_ENDPOINT,                  /* bDescriptorType    */
    (INT_IN_EP_NUM | EP_INPUT),     /* bEndpointAddress   */
    EP_INT,                         /* bmAttributes       */
    EP4_MAX_PKT_SIZE & 0x00FF,      /* wMaxPacketSize     */
    (EP4_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL     /* bInterval          */
};
```

3 Software and Hardware Environment

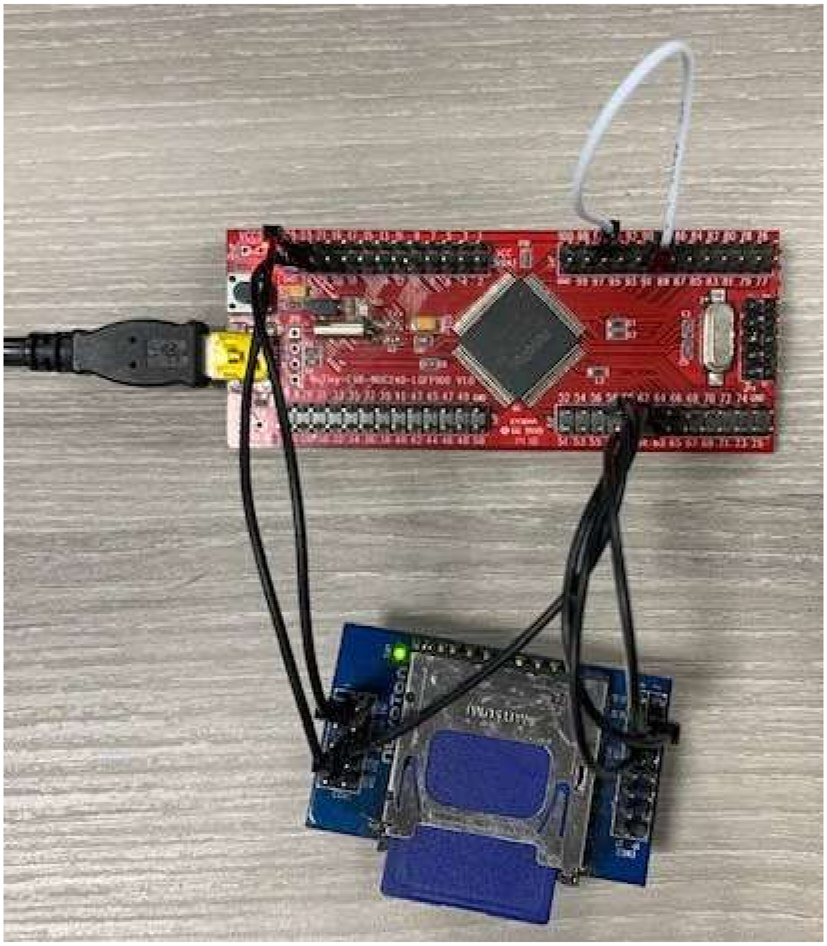
- **Software Environment**
  - BSP version
    - NUC200 Series BSP CMSIS V3.00.003
  - IDE version
    - Keil uVersion 5.26
- **Hardware Environment**
  - Circuit components
    - NuTiny-EVB-NUC240-LQFP100 V1.00
    - SD card socket board
    - USB mini USB cable
  - Diagram

NUC200 uses SPI1 interface to access SD card and use GPB.15 as Mouse input. The connection between NUC200 and SD card as below:



- SPI CS is Chip Select (The SPI master is NUC200) and connect to DAT3 pin of SD card.
- SPI MOSI is Data Out of NUC200 and connect to DAT0 pin of SD card. It need pull-up resistors to avoid floating data.

- SPI SCK is clock. NUC200 provides clock to SD card to sync the data.
- SPI MISO is Data In of NUC200 and connect to CMD pin of SD card. It need pull-up resistors to avoid floating data.



4 Directory Information

EC_NUC200_USBD_MSD_SPI_Mode_SDCard_HID_Mouse_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click USBD_MSD_SPI_Mode_SDCard_HID_Mouse.uvproj.
2. Enter Keil compile mode
    a. Build
    b. Download
    c. Start/Stop debug session
3. Enter debug mode
    a. Run

6 Revision History

Date	Revision	Description
Nov. 22, 2019	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.