Use NUC200 to Control PN512 Card Reader

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	The example code is to control card reader PN512 by NUC200 through I2C interface
BSP Version	NUC200 Series BSP CMSIS v3.00.004
Hardware	NuTiny-EVB-NUC200-LQFP100 V1.0

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

This example code uses the SPI 0 of NUC200 to initialize the card reader PN512.Initialize and read the card information through the frame work of NXP NFC module and prints the card information through the serial port0 to PC.
The example can detect and display card type and UID information.
The card reader supports ISO/IEC 14443B and ISO/IEC 14443B

2 Code Description

First configure SPI0 to communicate with PN512：

```
    phStatus_t phbalReg_Stub_OpenPort(
                                      phbalReg_Stub_DataParams_t * pDataParams
                                      )
    {
        /* Select HCLK as the clock source of SPI0 */
        CLK_SetModuleClock(SPI0_MODULE, CLK_CLKSEL1_SPI0_S_HCLK, MODULE_NoMsk);
        /* Enable SPI0 peripheral clock */
        CLK_EnableModuleClock(SPI0_MODULE);
        /* Setup SPI0 multi-function pins */
        GPIO_PIN_DATA(PORT2, PIN_SSEL) = SSEL_DEASR;  // CS on
        GPIOSetDir( PORT2, PIN_SSEL, SET_OUT );
        SYS->GPC_MFP = SYS_GPC_MFP_PC1_SPI0_CLK | SYS_GPC_MFP_PC2_SPI0_MISO0 |
SYS_GPC_MFP_PC3_SPI0_MOSI0;
        SYS->ALT_MFP = SYS_ALT_MFP_PC1_SPI0_CLK | SYS_ALT_MFP_PC2_SPI0_MISO0 |
SYS_ALT_MFP_PC3_SPI0_MOSI0;
        /* Configure as a master, clock idle low, 32-bit transaction, drive output on
falling clock edge and latch input on rising edge. */
        /* Set IP clock divider. SPI clock rate = 1MHz */
        SPI_Open(SPI0, SPI_MASTER, SPI_MODE_0, 8, 1000000);//10Mhz
        SPI_EnableFIFO(SPI0,4,4);


        return PH_ADD_COMPCODE(PH_ERR_SUCCESS, PH_COMP_BAL);
    }
```

Initialize the cyclic component that reads different cards：

```
phStatus_t DiscLoopInit( phacDiscLoop_Sw_DataParams_t   *pDataParams )    /* DataParams
representing the discovery loop */
{
    phStatus_t status;

    /* Set for poll and listen mode */
    status = phacDiscLoop_SetConfig( pDataParams,
        PHAC_DISCLOOP_CONFIG_MODE,
        DISCOVERY_MODE );
    CHECK_SUCCESS(status);

    /* Set for detection of TypeA, TypeB and Type F tags */
    status = phacDiscLoop_SetConfig(
```

```
        pDataParams,
        PHAC_DISCLOOP_CONFIG_DETECT_TAGS,
        POLL_TYPE
        );

    status = phacDiscLoop_SetConfig(pDataParams, PHAC_DISCLOOP_CONFIG_PAUSE_PERIOD_MS,
1000);
    CHECK_SUCCESS(status);

    /* Set number of polling loops to 5 */
    status = phacDiscLoop_SetConfig(pDataParams, PHAC_DISCLOOP_CONFIG_NUM_POLL_LOOPS, 5);
    CHECK_SUCCESS(status);

    /* Configure felica discovery */
    /* Set the system code to 0xffff */
    status = phacDiscLoop_SetConfig(pDataParams, PHAC_DISCLOOP_CONFIG_TYPEF_SYSTEM_CODE,
0xffff);
    CHECK_SUCCESS(status);
    /* Set the maximum number of Type F tags to be detected to 3 */
    status = phacDiscLoop_SetConfig(pDataParams, PHAC_DISCLOOP_CONFIG_TYPEF_DEVICE_LIMIT,
3);
    CHECK_SUCCESS(status);
    /* Set the polling limit for Type F tags to 5 */
    status = phacDiscLoop_SetConfig(pDataParams, PHAC_DISCLOOP_CONFIG_TYPEF_POLL_LIMIT,
5);
    CHECK_SUCCESS(status);
    /* Assign ATR response */
    discLoop.sTypeFTargetInfo.sTypeF_P2P.pAtrRes = bAtr_Res;
    /* Set ATR response length */
    status = phacDiscLoop_SetConfig(pDataParams,
PHAC_DISCLOOP_CONFIG_TYPEF_P2P_ATR_RES_LEN, bAtr_ResLength);
    }
```

Cycle detection to identify card type and read card information：

```
phStatus_t DiscLoopDemo( phacDiscLoop_Sw_DataParams_t  *pDataParams )
{
    phStatus_t    status=0;
    uint16_t      wTagsDetected=0;
    uint32_t      loop = 0;
    uint16_t      wNumberOfTags=0;
    uint8_t       bGtLen=0;    /* Gt length */
```

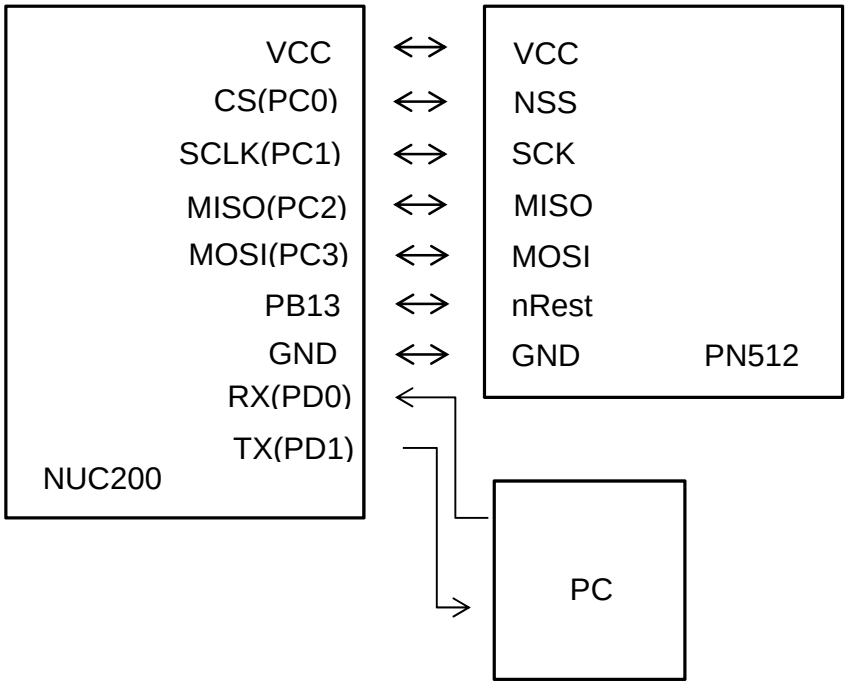
```
    PH_UNUSED_VARIABLE(bGtLen);

    while(1)
    {
        DEBUG_PRINTF1("Ready to detect.\n");

     status = phhalHw_FieldOff(pHal);
        status = phacDiscLoop_Start(pDataParams);

        if ((status & PH_ERR_MASK) == PH_ERR_SUCCESS)
        {
            /* Get the tag types detected info */
            status = phacDiscLoop_GetConfig(pDataParams,
PHAC_DISCLOOP_CONFIG_TAGS_DETECTED, &wTagsDetected);

            if (((status & PH_ERR_MASK) != PH_ERR_SUCCESS) || wTagsDetected ==
PHAC_DISCLOOP_NO_TAGS_FOUND)
            {
                phhalHw_FieldReset(&hal);
                continue;
            }
            DEBUG_PRINTF1 ("\n");

            LedOn();
            if (PHAC_DISCLOOP_CHECK_ANDMASK(wTagsDetected, PHAC_DISCLOOP_TYPEA_DETECTED))
            {
                if (wTagsDetected & PHAC_DISCLOOP_TYPEA_DETECTED_TAG_P2P)
                {
                    /* If Type F tag with P2P capability was discovered, do and P2P
exchange */
                  DEBUG_PRINTF1("\nType A with P2P has been detected");

                    status = phOsal_Timer_Wait(&osal, 1, 1000);
                    CHECK_SUCCESS(status);

                    status = phhalHw_FieldReset(&hal);
                    CHECK_SUCCESS(status);

                    /* Start the next polling cycle */
                    continue;
                }
                /* Get the number of Type A tags detected */
```

```
                status = phacDiscLoop_GetConfig(pDataParams,
PHAC_DISCLOOP_CONFIG_TYPEA_NR_TAGS_FOUND, &wNumberOfTags);
                CHECK_SUCCESS(status);



                /* Loop through all the Type A tags detected and print their UIDs */
                for (loop = 0; loop < wNumberOfTags; loop++)
                {
                  DEBUG_PRINTF1 ("\nCard %d",loop + 1);
                  DEBUG_PRINTF1 ("\nUID: ");
                  PRINT_BUFF(
                         pDataParams->sTypeATargetInfo.aTypeA_I3P3[loop].aUid,
                         pDataParams->sTypeATargetInfo.aTypeA_I3P3[loop].bUidSize
                         );
                  if(loop == 0)
                  {
                           DEBUG_PRINTF1("\nAtqa:");
                           PRINT_BUFF(
                               pDataParams->sTypeATargetInfo.aTypeA_I3P3[loop].aAtqa,
                               2
                               );
                           DEBUG_PRINTF1("\nSak:");
                           PRINT_BUFF(
                               pDataParams->sTypeATargetInfo.aTypeA_I3P3[loop].aSak,
                               1
                               );
                           DEBUG_PRINTF1 ("\n");
}
```

3 Software and Hardware Environment

- **Software Environment**
  - BSP Version
    - NUC200 Series BSP CMSIS v3.00.004
  - IDE Version
    - Keil uVersion 4.6
- **Hardware Environment**
  - Circuit Components
    - NuTiny-EVB-NUC200-LQFP100 V1.0
    - PN512
  - Diagram

NUC200 is communicate with PN512 through SPI0. Detects and reads card information to NUC200 by PN512, and report data to PC for printing through UART0 .



4 Directory Information

- EC_NUC200_NFC_Card_Reader_PN512_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
  - ThirdParty
    - PNEV512B-LPC1227-LPC1769-P2P_Classic_Polling-SPI-I2C — NXP Reader Library Framework

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click NUC200_NFC_Card_Reader_PN512.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
Oct.12, 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.