Use M031 SPI Driver SI4463

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code uses M031 SPI to drive SI4463 to achieve wireless sending and receiving data.
BSP Version	M031 BSP CMSIS V3.03.000
Hardware	NuMaker-M031SD V1.1

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 uses M031 to drive SI4463 through SPI interface to achieve wireless (433M) sending and receiving data.

1.1 Principle

Silicon Laboratories' Si446x devices are high-performance, low-current transceivers covering the sub-GHz frequency bands from 119 to 1050 MHz. All parts offer outstanding sensitivity of –126 dBm while achieving extremely low active and standby current consumption. The Si446x communicates with the host MCU over a standard 4-wire serial peripheral interface (SPI): SCLK, SDI, SDO, and nSEL. The SPI interface is designed to operate at a maximum of 10 Mbps.

The host MCU writes data over the SDI pin and can read data from the device on the SDO output pin. Figure 1-1 demonstrates an SPI write command. The nSEL pin should go low to initiate the SPI command. The first byte of SDI data will be one of the firmware commands followed by n bytes of parameter data which will be variable depending on the specific command. The rising edges of SCLK should be aligned with the center of the SDI data.

The SPI Write Command and Time sequence is shown below.

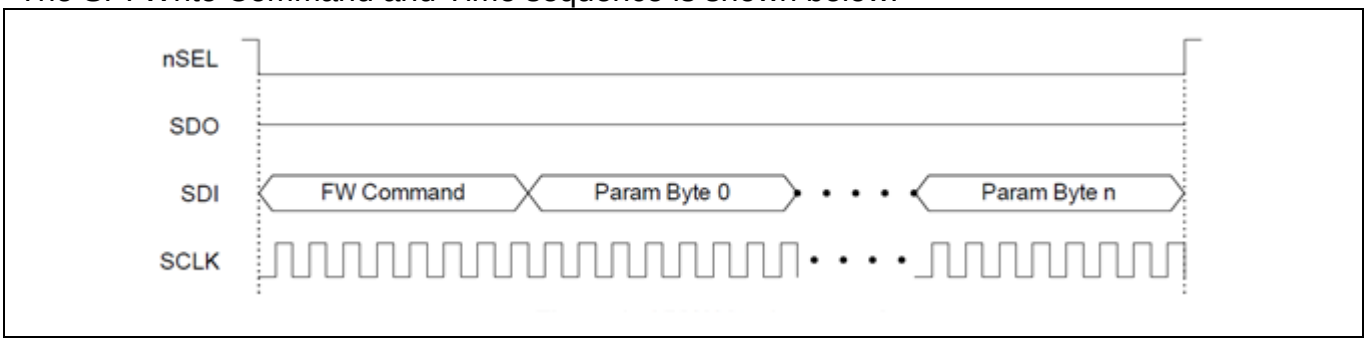

Figure 1-1 SPI Write Command

The Si446x contains an internal MCU which controls all the internal functions of the radio. For SPI read commands a typical MCU flow of checking clear-to-send (CTS) is used to make sure the internal MCU has executed the command and prepared the data to be output over the SDO pin. Figure 1-2 demonstrates the general flow of an SP read command. Once the CTS value reads FFh then the read data is ready to be clocked out to the host MCU. The typical time for a valid FFh CTS reading is 20 us. Figure 1-3 demonstrates the remaining read cycle after CTS is set to FFh. The internal MCU will clock out the SDO data on the negative edge so the host MCU should process the SDO data on the rising edge of SCLK.

The SPI Read Command and Flow is shown below.

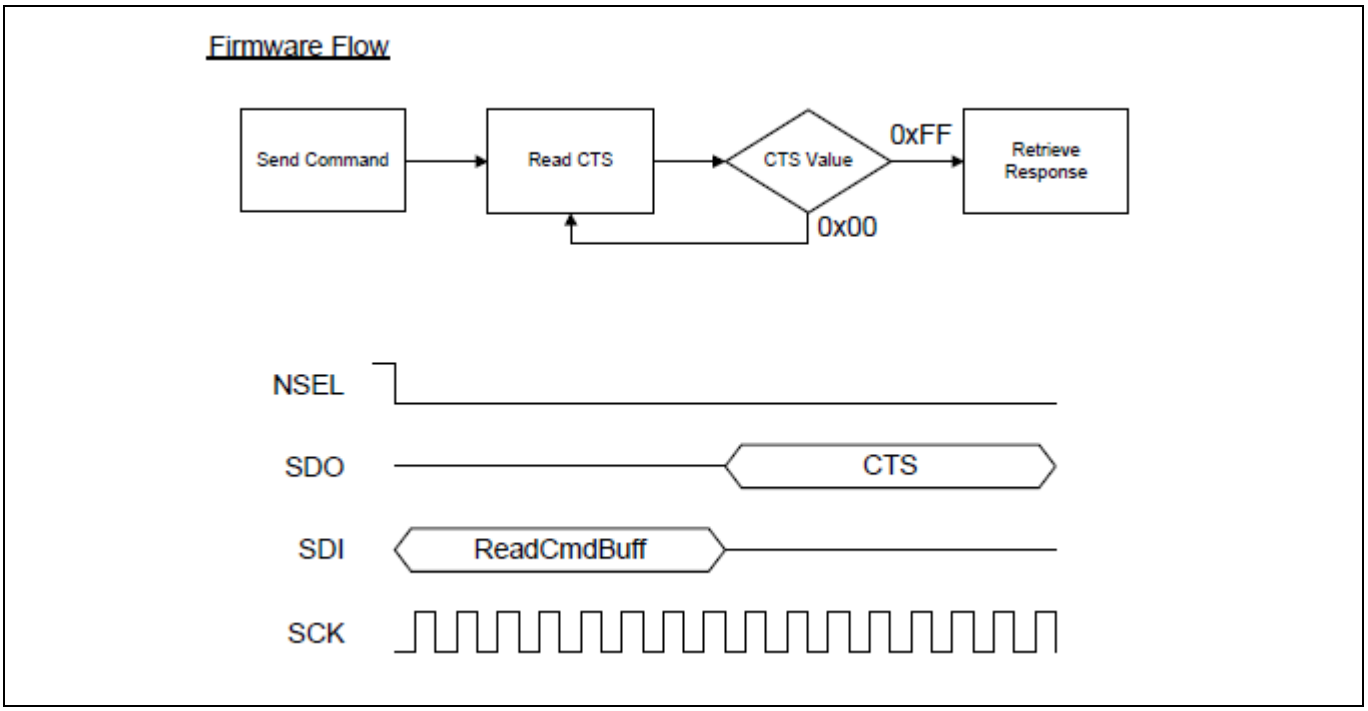

Figure 1-2 SPI Read Command-Check CTS Value

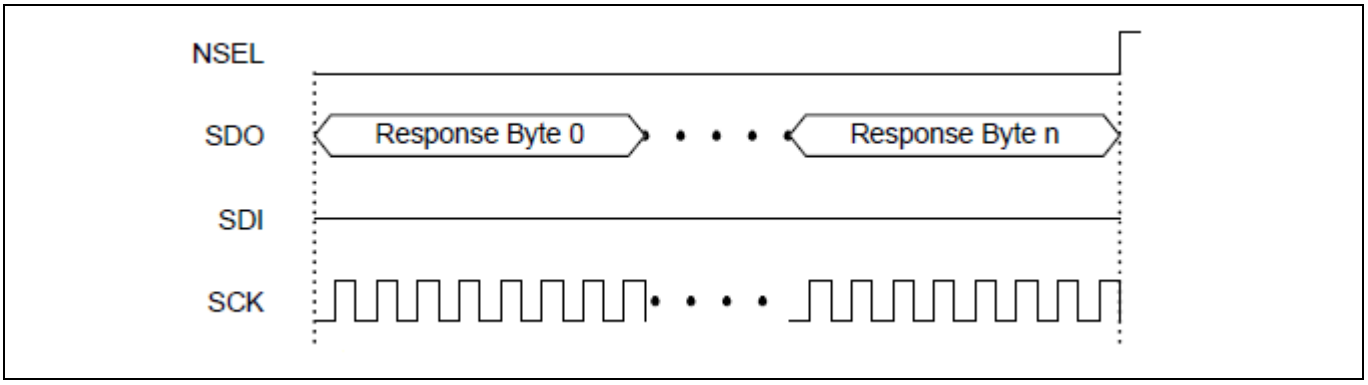

Figure 1-3 SPI Read Command-Clock Out Read Data

Figure 1-1, Figure 1-2 and Figure 1-3 information is sourced from SI4463 Datasheet.
Please refer to following documentation link for more SI4463 and register information:
https://www.silabs.com/documents/public/data-sheets/Si4463-61-60-C.pdf

1.2 Demo Result

The Master/Slave is connected to the computer through serial port, and the printed information is as follows:
As Master, the SI4463 sends data packet and receives ACK:

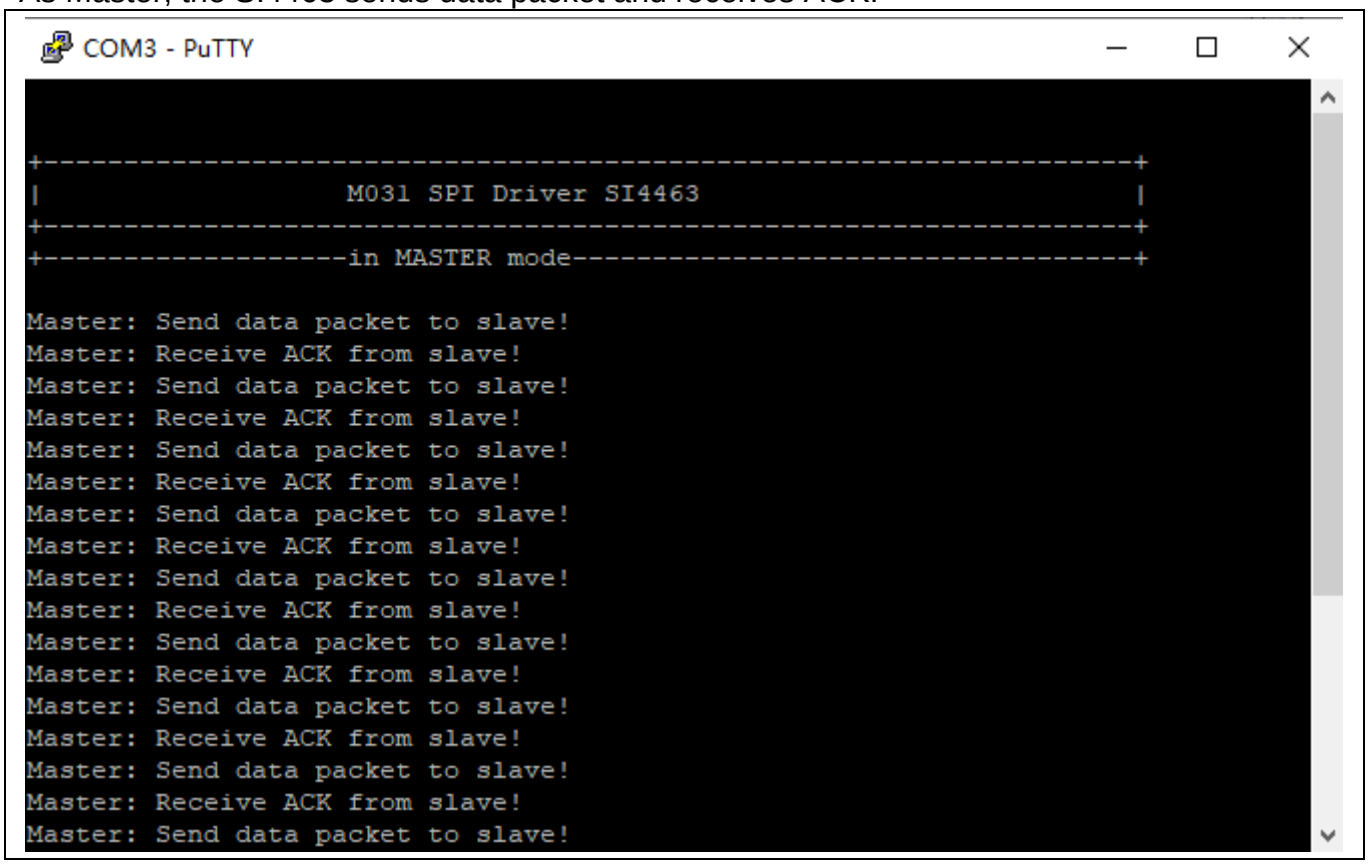

Figure 1-4 Master Communication Result

As Slave, the SI4463 receives data packet and sends ACK:

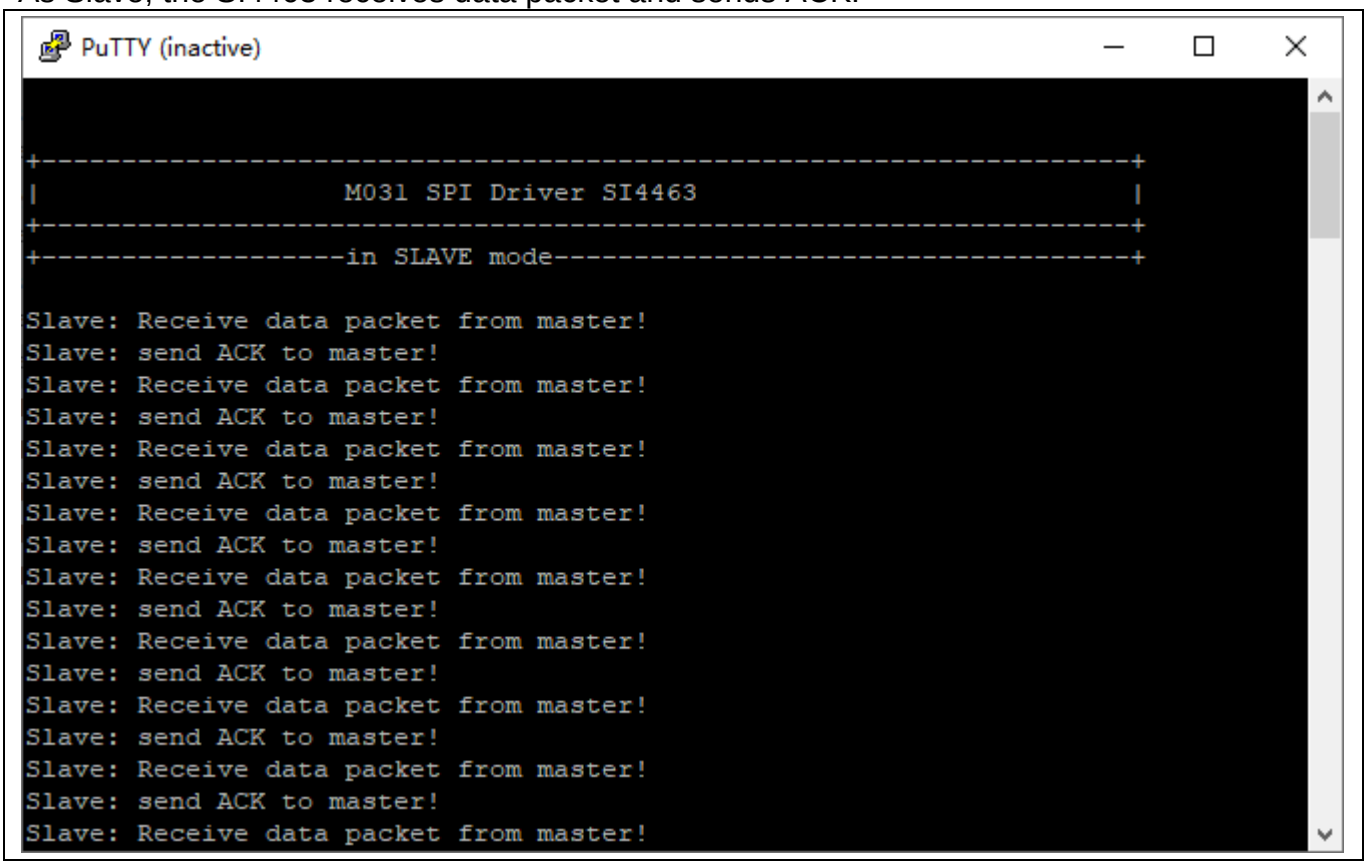

Figure 1-5 Slave Communication Result

2 Code Description

Set up the master or slave:

```
#define EnableMaster   1       //in master mode
//#define EnableMaster   0        //in slave mode
```

Initialize SI4463:

```
void vRadio_Init(void)
{
    uint16_t wDelay;

    /* Power Up the radio chip */
    vRadio_PowerUp();

    /*get  part information6 */
    si446x_part_info();

    /*part information check */
    if (Si446xCmd.PART_INFO.PART != 0x4463)
    {
        while (1)
        {
            /* ERROR!!!!  SPI communicate error  or part information error */
#ifdef RADIO_COMM_ERROR_CALLBACK
            RADIO_COMM_ERROR_CALLBACK();
#endif
        }
    }

    /* Load radio configuration */
    while (SI446X_SUCCESS != si446x_configuration_init(pRadioConfiguration-
>Radio_ConfigurationArray))
    {
        for (wDelay = 0x7FFF; wDelay--;) ;

        /* Power Up the radio chip */
        vRadio_PowerUp();
    }

    /* Read ITs, clear pending ones  */
    si446x_get_int_status(0, 0, 0);
}
```

The Si4463 receives and sends state switching process:

```
uint8_t bRadio_Process()
{
    switch (RFstate)
    {
        case RFLR_STATE_RX_INIT:
            radio_TX(0);
            radio_RX(1);
            vRadio_StartRX(pRadioConfiguration->Radio_ChannelNumber, pRadioConfiguration-
>Radio_PacketLength);
```

```
            TickCounter = 0;
            RFstate = RFLR_STATE_RX_RUNNING;
            break;

        case RFLR_STATE_RX_RUNNING:
            //Delay_Us(300);
            //si446x_get_modem_status_fast_clear_read();//RSSI
            bMain_IT_Status = bRadio_Check_Tx_RX();

            if (bMain_IT_Status ==
SI446X_CMD_GET_INT_STATUS_REP_PH_PEND_PACKET_RX_PEND_BIT)
            {
                if (EnableMaster)
                {
                    RFstate = RFLR_STATE_RX_ACK_DONE;
                }
                else
                {
                    RFstate = RFLR_STATE_RX_DONE;
                }
            }

            if (TickCounter > 2000)
            {
                RFstate = RFLR_STATE_RX_TIMEOUT;
            }
            break;

        case RFLR_STATE_TX_INIT:
            radio_TX(1);
            radio_RX(0);

            gSampleCode_SendVariablePacket();
            TickCounter = 0;
            RFstate = RFLR_STATE_TX_RUNNING;
            break;

        case RFLR_STATE_TX_ACK_INIT:
            radio_TX(1);
            radio_RX(0);

            printf("Slave: send ACK to master! \n") ;
```

```
            vSampleCode_SendAcknowledge();
            TickCounter = 0;
            RFstate = RFLR_STATE_TX_RUNNING;
            break;

        case RFLR_STATE_TX_RUNNING:
            bMain_IT_Status = bRadio_Check_Tx_RX();

            if (bMain_IT_Status ==
SI446X_CMD_GET_INT_STATUS_REP_PH_PEND_PACKET_SENT_PEND_BIT)
            {
                if (EnableMaster)
                {
                    RFstate = RFLR_STATE_TX_DONE;
                }
                else
                {
                    RFstate = RFLR_STATE_TX_ACK_DONE;
                }
            }

            if (TickCounter > 2000)
            {
                RFstate = RFLR_STATE_TX_TIMEOUT;
            }
            break;

        case RFLR_STATE_RX_DONE:
            if (gSampleCode_StringCompare(customRadioPacket, tx_data, sizeof(tx_data)) ==
true)
            {
                RFstate = RFLR_STATE_TX_ACK_INIT;
                printf("Slave: Receive data packet from master! \n") ;
                HAL_Delay_nMS(200);
            }
            else
            {
                RFstate = RFLR_STATE_RX_INIT;
            }
            break;

        case RFLR_STATE_RX_ACK_DONE:
```

```
            if (gSampleCode_StringCompare(customRadioPacket, ACK_data, 3) == true)
            {
                printf("Master: Receive ACK from slave! \n") ;
            }

            HAL_Delay_nMS(100);
            RFstate = RFLR_STATE_TX_INIT;
            break;

        case RFLR_STATE_TX_DONE:
            printf("Master: Send data packet to slave! \n") ;
            RFstate = RFLR_STATE_RX_INIT;
            break;

        case RFLR_STATE_TX_ACK_DONE:
            HAL_Delay_nMS(20);
            RFstate = RFLR_STATE_RX_INIT;
            break;

        case RFLR_STATE_TX_TIMEOUT:
            if (EnableMaster)
            {
                RFstate = RFLR_STATE_TX_INIT;
            }
            else
            {
                RFstate = RFLR_STATE_RX_INIT;
            }
            break;

        case RFLR_STATE_RX_TIMEOUT:
            if (EnableMaster)
            {
                RFstate = RFLR_STATE_TX_INIT;
            }
            else
            {
                RFstate = RFLR_STATE_RX_INIT;
            }
            break;
```

```
        default:
            break;
    }

    return RFstate;
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.03.000
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M031SD V1.1
  - SI4463 module
- Pin Connect

The hardware uses the M031 to connect the SI4463 module, as Figure 3-1.

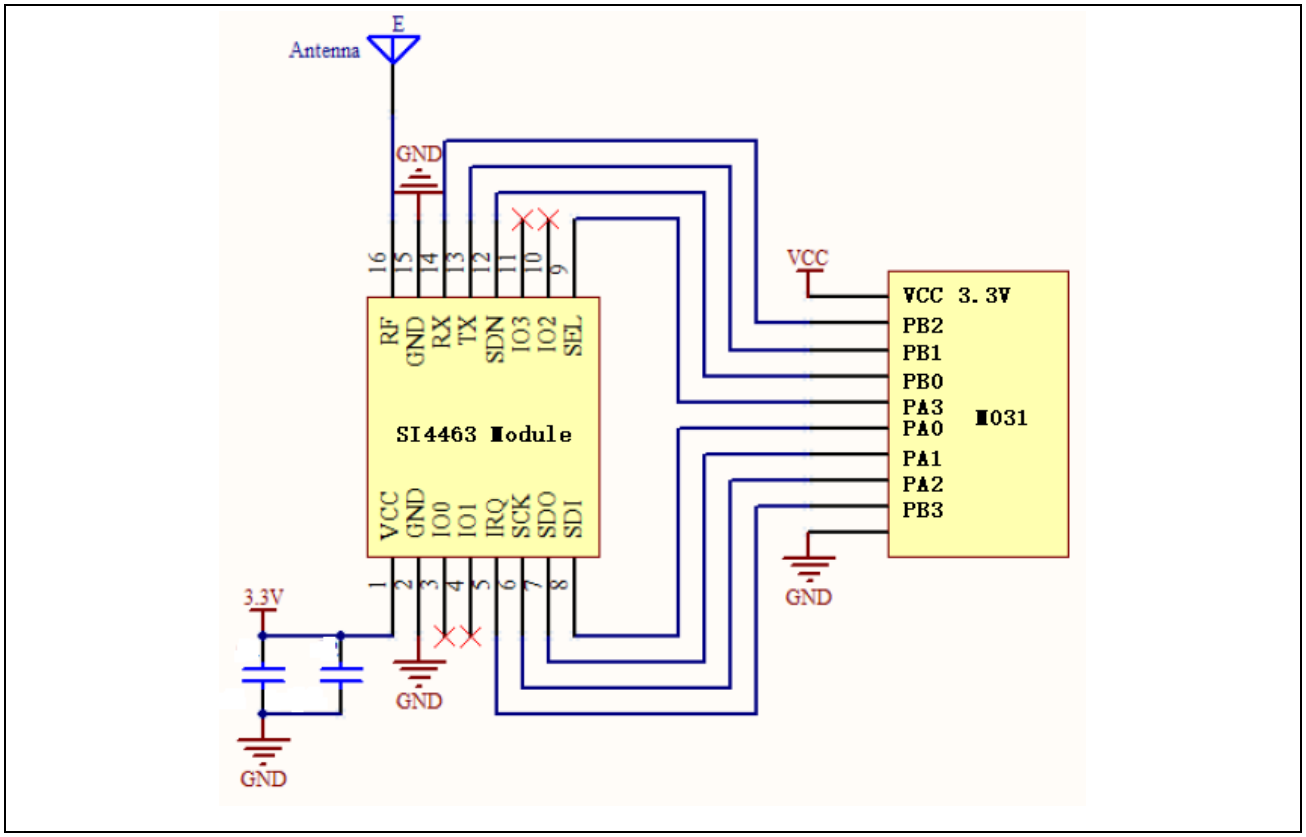

Figure 3-1 Pin Connect

The relevant pin functions of SI4463 module are described, as Table 3-1.

Pin name	I/O	Description
IO0	I/O	General Purpose Digital I/O
IO1	I/O	General Purpose Digital I/O
IRQ	O	General Microcontroller Interrupt Status Output. When the Si4463/61 exhibits any one of the interrupt events, the IRQ pin will be set low = 0.
SCK	I	SPI Clock Input
SDO	O	SPI MISO Output
SDI	I	SPI MOSI Input
SEL	I	SPI Select Input.
IO2	I/O	General Purpose Digital I/O
IO3	I/O	General Purpose Digital I/O
SDN	I	Shutdown Input Pin. SDN should be = 0 in all modes except Shutdown mode. When SDN = 1, the chip will be completely shut down, and the contents of the registers will be lost.
TX	I	Transmit output control. RX must be low.
RX	I	Receive input control. TX must be low.
RF	I/O	RF signal input/output, connected to 50Ω antenna.

Table 3-1 Pin Description

4 Directory Information

The directory structure is shown below.

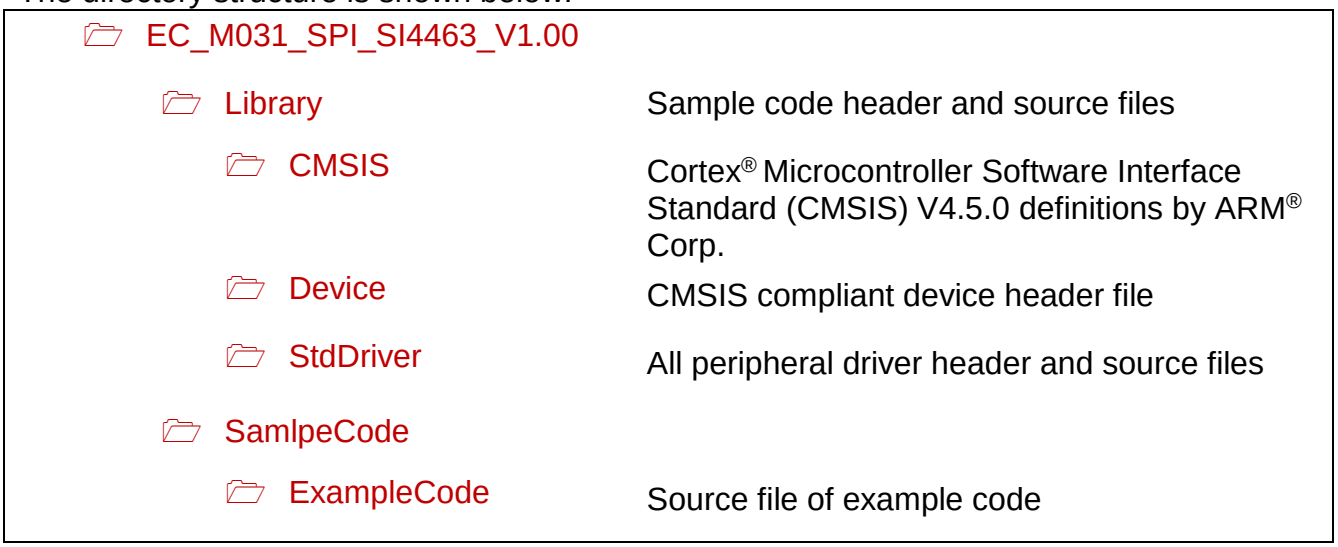

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M031_SPI_SI4463.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.06.25	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.