



Application note

2015/2016

Implementation of the wireless communication module

Papa Amadou SECK
GE5A

Table des matières

1. SPI Protocol	2
2. Boot-Up Sequence.....	3
3. Command Structure	3
4. WICED Wi-Fi SDK Broadcom.....	5

Table des illustrations

Figure 1: gSPI Write protocol	2
Figure 2: gSPI read protocol	2
Figure 3: WLAN boot-up sequence	3
Figure 4: Command structure	4

The SN8000 is a certified 2.4 GHz IEEE 802.11b/g/n Wi-Fi module of Murata based on the Broadcom BCM43362 chipset. It is designed to fit indoor and outdoor sensor and control applications. The SN8000 integrates a Wi-Fi IC, RF front end, temperature compensated crystal (TCXO) and an on-board antenna.

The implementation of the wireless communication module SN8000 was realized using the documentation of BCM43362 chipset.

1. SPI Protocol

The SPI protocol supports both 16-bit and 32-bit word operation. Byte endianness is supported in both modes. Figure 1 and Figure 2 show the basic write and write/read commands.

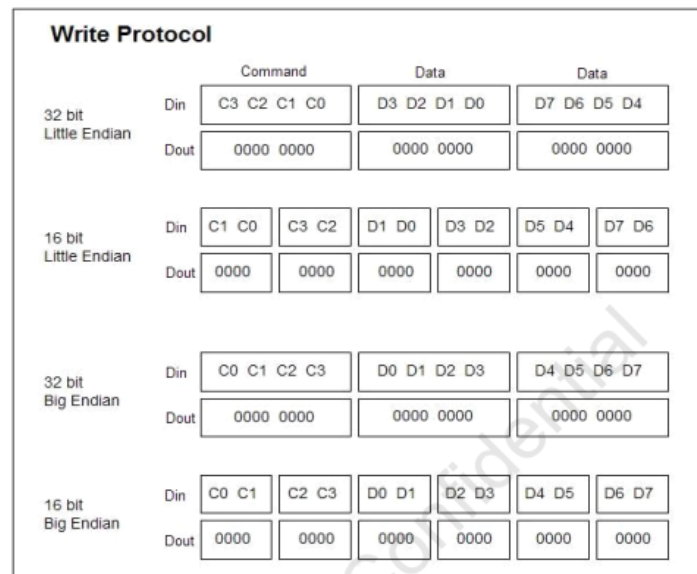


Figure 1: gSPI Write protocol

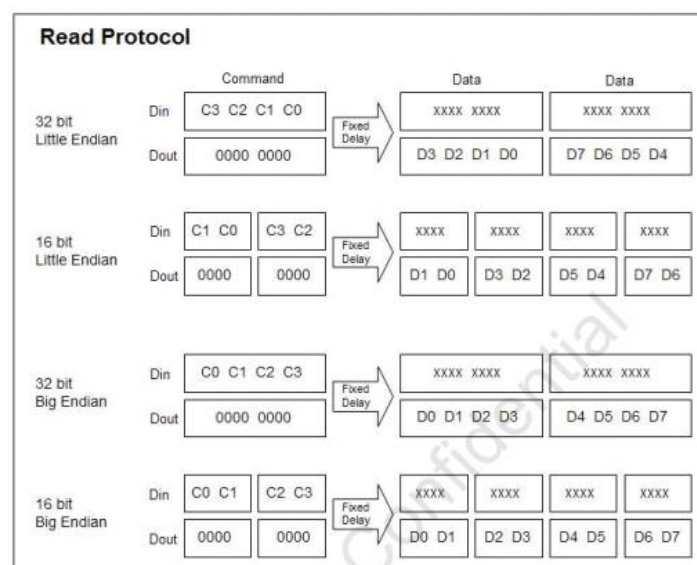


Figure 2: gSPI read protocol

2. Boot-Up Sequence

After power-up, the gSPI host needs to wait 50 ms for the device to be out of reset. For this, the host needs to poll with a read command to F0 addr 0x14. Address 0x14 contains a predefined bit pattern of 32 bits and is only available in reading.

After the reset setting, it is necessary to wait at least 3 ms before starting to send the reading frame. The default module is in 16 bit and little Indian. If we can read the pattern, then the SPI works.

we can see this boot-up sequence in the figure below:

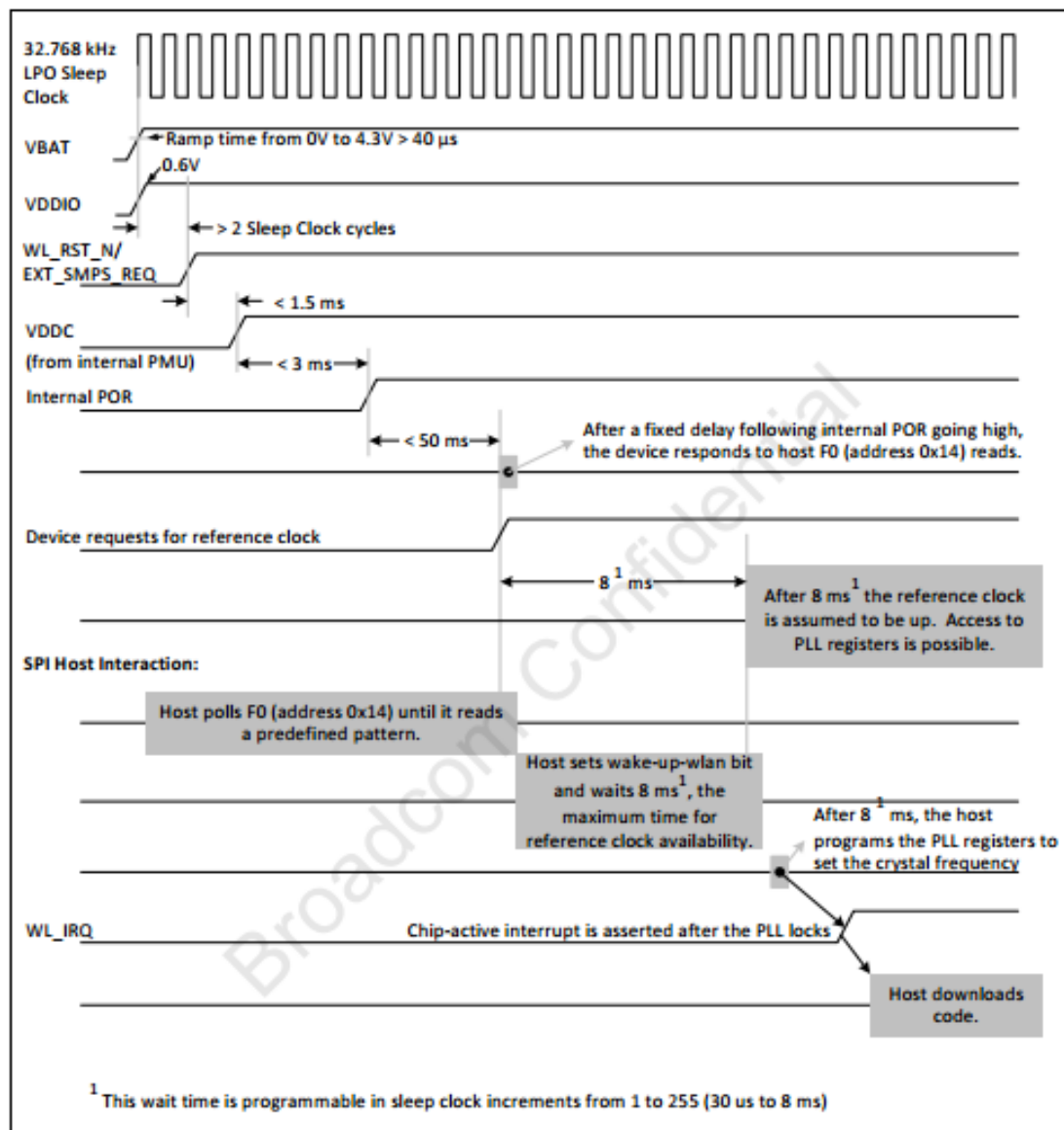


Figure 3: WLAN boot-up sequence

3. Command Structure

The gSPI command structure is 32 bits. We can show the bit positions and definitions in Figure 4.

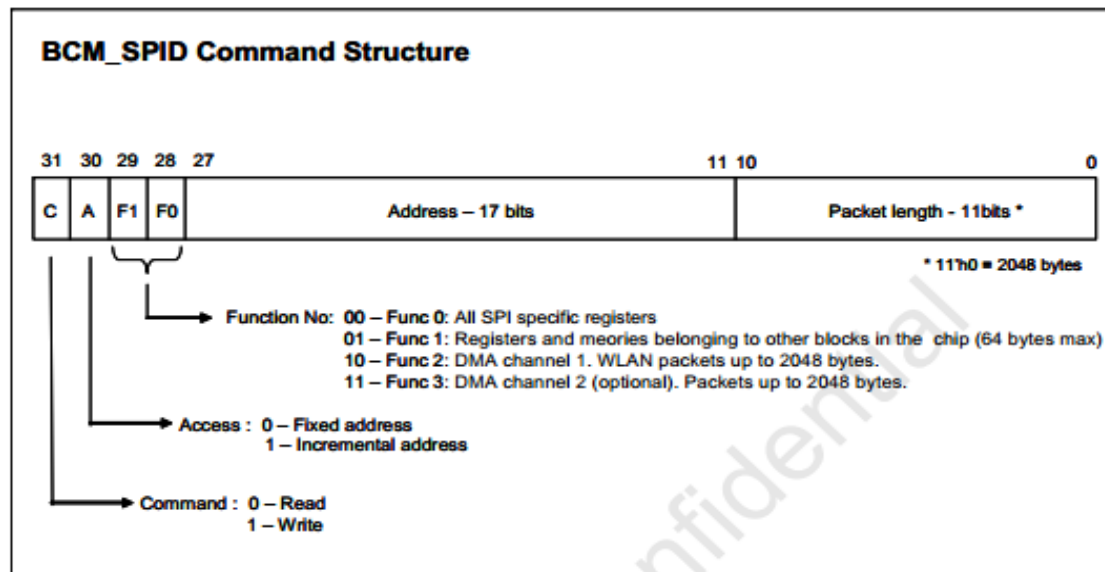


Figure 4: Command structure

The following function was developed to read the pattern 0XFEEDBEAD at address 0x14 to check the operation of the SPI:

```
SN8000_read_pattern() //Read Pattern in 16 BITS LSB (Default)
{
    PORT6.Pn.BIT.Pn13 = 0;
    data=Pekoe_RSPI1_receive16(0xA000); //Read command: begin at 0x0014 to 0x0017 for Test Pattern
    data=Pekoe_RSPI1_receive16(0x4000); //incremental option
    PORT6.Pn.BIT.Pn13 = 1;
    for(j=0;j<40;j++){__asm__("nop");}
    PORT6.Pn.BIT.Pn13 = 0;
    data1=Pekoe_RSPI1_receive16(0xFFFF); //Read 32 bits (Test Pattern)
    data2=Pekoe_RSPI1_receive16(0xFFFF);
    PORT6.Pn.BIT.Pn13 = 1;
    for(j=0;j<44000;j++){__asm__("nop");} //wait
}
```

A new function has been developed to write a pattern 0xAABBCCDD in a register and to read the pattern written to validate the SPI communication:

```
SN8000_wirte_pattern() //Write Pattern in 16 BITS LSB (Default)
{
    //WRITE
    PORT6.Pn.BIT.Pn13 = 0;
    data=Pekoe_RSPI1_receive16(0xC000); //Write command: begin at 0x0018 to 0x001B for Test Pattern
    data=Pekoe_RSPI1_receive16(0xC000); //incremental option
    data1=Pekoe_RSPI1_receive16(0xBBAA); //Write 32 bits (Test Pattern)
    data2=Pekoe_RSPI1_receive16(0xDDCC);
    PORT6.Pn.BIT.Pn13 = 1;
    for(j=0;j<44000;j++){__asm__("nop");} //wait

    //READ
    PORT6.Pn.BIT.Pn13 = 0;
    data=Pekoe_RSPI1_receive16(0xC000); //Read command: begin at 0x0018 to 0x001B for Test Pattern
    data=Pekoe_RSPI1_receive16(0x4000); //incremental option
    PORT6.Pn.BIT.Pn13 = 1;
    for(j=0;j<40;j++){__asm__("nop");}
    PORT6.Pn.BIT.Pn13 = 0;
    data3=Pekoe_RSPI1_receive16(0xFFFF); //Read 32 bits (Test Pattern)
    data4=Pekoe_RSPI1_receive16(0xFFFF);
    PORT6.Pn.BIT.Pn13 = 1;
    for(j=0;j<44000;j++){__asm__("nop");} //wait
}
```

it is easier to put the module in 32-bit word to write frames. For doing that, we must set the configuration register. This register especially once our SPI communication is functional to completely awaken our module. Thus, there must be a bit 7 to the value 1 to wake up the module and start the oscillator and PLL that it needs to exchange data.

Once the module is awakened, the lowest level game is over. You must then use the Broadcom stack for exchanging data with the Wi-Fi module.

4. WICED Wi-Fi SDK Broadcom

WICED Wireless Internet Connectivity for Embedded Devices is for a high-level wrapper for creating embedded network applications using Broadcom chips. It is an optimized library that contains all the necessary wireless drivers to operate with the chips. In our case, we use a Broadcom 43362 chip.

So it is essential to use Broadcom Wiced SDK Wi-Fi to use the advanced functions of Wi-Fi module. For adding the Wi-Fi module to the library, it is necessary to add files to the library.

First, there is a Platform folder that contains all the source files and header of available microcontrollers. We must therefore add the files needed for the operation of RZA1L.

The RZA1L has a Cortex A9, we add a folder called ARM_CA9 that allows to operate the Cortex A9. Then we add in the MCU subfolder the RZA1L with many source files and headers needed for the operation of the microcontroller. We will talk about the important file you must name `wwd_SPI.c`. This is a file that defines the SPI functions that WICED library will use to send frames to the module. It is in this file that we must adapt our SPI functions (low level) that we developed previously.

In this file, you will find the initialization function that will adapt the SPI of the microcontroller to function with the Wi-Fi module:

```
wwd_result_t host_platform_bus_init( void )
{
    host_rtos_init_semaphore(&spi_transfer_finished_semaphore);

    /* Setup SPI */

    PORT3.Pn.BIT.Pn15 = 0; // PMOD CS
    Pekoe_Init_SPI();
    host_rtos_delay_milliseconds( (uint32_t) 1 ); //wait power up
    PORT3.Pn.BIT.Pn15 = 1; // PMOD CS
    host_rtos_delay_milliseconds( (uint32_t) 3 ); //wait internal POR //<3ms

    return WICED_SUCCESS;
}
```

And the most important function to test is the **host_platform_spi_transfer** function that will use the low level routines SPI previously developed with RZA1L to send frames to the module:

```

wwd_result_t host_platform_spi_transfer( wwd_bus_transfer_direction_t direction, uint8_t* buffer, uint16_t buffer_length )
{
    wiced_result_t result = WWD_SUCCESS;
    uint8_t trans_data = 0u;
    int k;
    long j=0;

    platform_mcu_powersave_disable();

    PORT6.Pn.BIT.Pn13 = 0; //CS
    for(k=0;k<buffer_length;k++)
    {
        if(direction==BUS_READ)
        {
            *buffer=Pekoe_RSPI1_receive8(*buffer);
            buffer++;
        }
        else
        {
            Pekoe_RSPI1_receive8(*buffer);
            buffer++;
        }
    }
    for(j=0;j<4000;j++){__asm__("nop");} //wait
    for(j=0;j<4000;j++){__asm__("nop");}
    PORT6.Pn.BIT.Pn13 = 1; //CS

    platform_mcu_powersave_enable( );

    return result;
}

```

Once the RZA1L is integrated with Wiced, we must use a function (the **wwd_management_wifi_on ()** in our case) which will call more functions for SPI. All functions used are available in the documentation section of the project.