Application Note

SPI Slave interface Porting Guide for ATWINC1500

AN-XXXXXB

Prerequisites

- **Hardware Prerequisites**
  - Atmel SAMD21 Xplained Evaluation Kit [1]
  - Atmel WINC1500 board [2]
  - Micro USB Cable (TypeA / MicroB)
- **Build Prerequisites**
  - Atmel Studio 6 [3]
  - WINC1500 software release

Introduction

This application note describes how to integrate the Atmel WINC1500 to communicate with the Atmel MCU via SPI (Serial Peripheral Interface). This application note also includes an appendix to work on SPI integration with Atmel MCU.

Table of Contents

1. Introduction

ATWINC1500 external interfaces include I2C slave for control, SPI slave and SDIO slave for control and data transfer. This application note focuses on the SPI that operates as a SPI slave. The Appendix A shows how to get simple example to study porting the SPI slave interface of WINC1500 with SAMD21 Xplained board. For more information on WINC1500, see the WINC1500 datasheet [4].

2. SPI Slave Interface

ATWINC1500 provides a Serial Peripheral Interface (SPI) that can be used for control and for serial I/O of 802.11 data. The SPI slave pins are mapped as shown in Table 2-1. The RXD pin is same as Master Output, Slave Input (MOSI), and the TXD pin is same as Master Input, Slave Output (MISO). The SPI slave is a full-duplex slave synchronous serial interface that is available immediately following reset when pin 9 (SDIO_SPI_CFG) is tied to VDDIO.

Table 2-1 ATWINC1500 SPI Slave Interface Pin Mapping

Pin No.	SPI function
9	CFG: Must be tied to VDDIO
16	SSN: Active Low Slave Select
18	SCK: Serial Clock
13	RXD: Serial Data Receive
17	TXD: Serial Data Transmit

When the SPI is not selected (SSN is high), the SPI will not interfere with data transfers between the serial-master and other serial-slave devices. When the serial slave is not selected, its transmitted data output is buffered, resulting in a high impedance driver onto the serial master receive line. The SPI slave interface responds to a protocol that allows an external host to read or write any register in the chipset as well as initiate DMA transfers.

The SPI Slave interface supports for standard modes as determined by the CPOL (Clock Polarity) and CPHA (Clock Phase) configuration. These modes are illustrated in Table 2-2.

Table 2-2 SPI Slave Modes

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

The red lines in Figure 2-1 correspond to CPHA=0 and the blue lines correspond to CPHA=1. The Figure 2-2 shows ATWINC1500 SPI timing diagram.

Figure 2-1 SPI Slave Clock Polarity and Clock Phase Timing

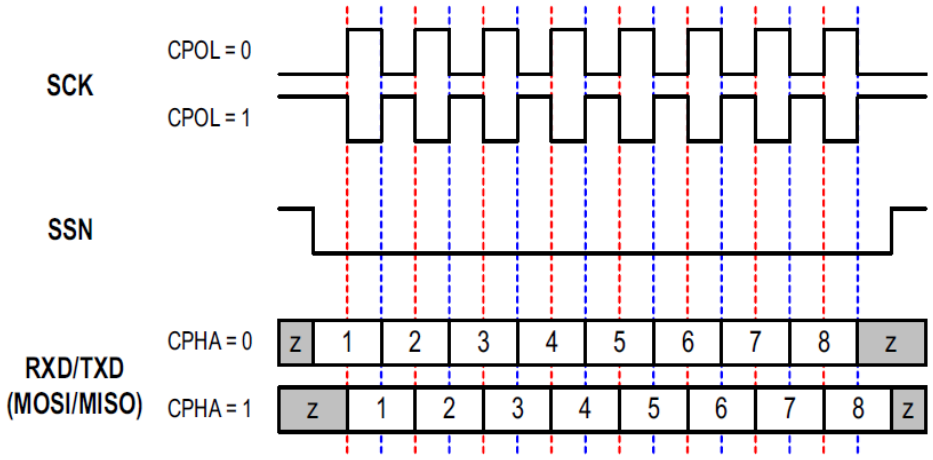
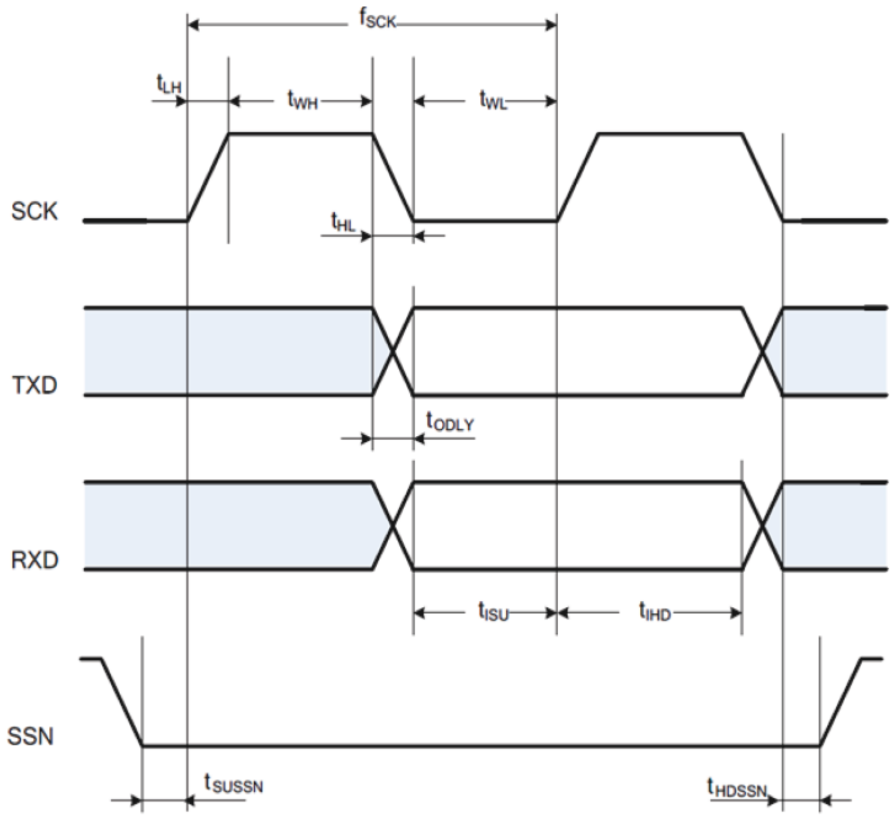


Figure 2-2 ATWINC1500 SPI Timing Diagram



TIPS For more information on WINC1500, refer to the application notes available from Atmel SmartConnect website.

.

3. WINC1500 host interface driver

This section introduces WINC1500 host interface layer to communicate with a host MCU via serial interfaces supported by WINC1500 software. The WINC1500 software architecture consists of three components, which are IoT Application layer, WINC1500 firmware and WINC1500 ASIC driver. The WINC1500 ASIC part includes the host interface layer to communicate with the host MCUs, for example, SAMD21 via SPI or I2C.

3.1 Host driver configuration for SPI interface

For communicating with a host application CPU, **USE_SPI** macro should be defined for SPI interface. Find **USE_SPI** macro in the driver source codes to see more information on the SPI uses. This macro can be defined in the Atmel Studio compiler symbols.

3.2 Platform dependent driver

The WINC1500 driver consists of platform independent and dependent parts. The dependent part of WINC1500 driver should be ported to specific platforms. The WINC1500 driver defines which functions should be ported to communicate with a host CPU via SPI. The section 3.3 lists the declarations defined in **nm_bus_wrapper.h**, which provides wrapper functions to specific platform.

TIPS The **nm_bus_wrapper.h** is located in the /src/host_drv/bus_wrapper/include.

3.3 Bus wrapper APIs

This section describes host bus wrapper functions. The independent parts of WINC1500 driver call them in proper time. For example, **nm_bus_init** will be called during driver initial stage. In similar, **nm_bus_deinit** will be called when de-initializing the driver. The **nm_bus_ioctl** is called when the driver reads or writes the control packets or 802.11 data.

nm_bus_init

Declaration

sint8 nm_bus_init (void *)

Description

Initialize the bus wrapper

Return

M2M_SUCCESS in case of success and M2M_ERR_BUS_FAIL in case of failure

nm_bus_deinit

Declaration

sint8 nm_bus_deinit (void)

Description

De-initialize the bus wrapper

Return

ZERO in case of success and M2M_ERR_BUS_FAIL in case of failure

nm_bus_ioctl

Declaration

sint8 nm_bus_ioctl (uint8 u8Cmd, void* pvParameter)

Description

send/receive from the bus

Parameters

U8Cmd: IOCTL command for the operation

pvParameter: Arbitrary parameter depending on IOCTL

Return

M2M_SUCCESS in case of success and M2M_ERR_BUS_FAIL in case of failure

The chapter 4 will show how to port the bus wrapper functions on the SAMD21 Xplained board.

4. Porting on SAMD21 Xplained board

This chapter describes how to port the WINC1500 SPI bus wrapper with SAMD21 Xplained board. For more details on SAMD21, refer to SAMD21 Xplained Pro User Guide. See also SAM D MCUs for more information on SAM D MCU series from Atmel. The Appendix A shows simple example from Atmel Studio reading chip id with WINC1500. This example can be used to verify porting SPI slave bus wrapper is ok.

4.1 Writing nm_bus_wrapper.c file

The WINC1500 release package includes **nm_bus_wrapper.h** to declare which functions are required by the driver, dependent part of the WINC1500 ASIC driver. The section 4.2 describes how to write the bus wrapper C-file to define them declared in **nm_bus_wrapper.h**. It's required to make new file to define declarations, for example, **nm_bus_wrapper_samd21.c** file, which intentionally indicates it belongs to SAM D21 board. Make sure new file should be included in compile lists.

4.2 Defining SPI wrapper functions

The following sample code snapshot shows how to initialize the SPI driver to provide the wrapper function, **nm_bus_init** to WINC1500 ASIC driver. This function's goal is to initialize the SPI driver of SAM D21. The WINC1500 ASIC driver needs host interface layer to communicate with a host MCU so calls this function during WINC1500 initial time. Then, the WINC1500 ASIC driver tries to read chip id to check if the SPI communication has no problem.

```
sint8 nm_bus_init(void *pvInitValue)
{
        sint8 result = M2M_SUCCESS;
        /* Structure for SPI configuration. */
        struct spi_config config;
        struct spi_slave_inst_config slave_config;
        /* Select SPI slave CS pin. */
        /* This step will set the CS high */
        spi_slave_inst_get_config_defaults(&slave_config);
```

```
                slave_config.ss_pin = CONF_WIFI_M2M_SPI_CS_PIN;
                spi_attach_slave(&slave_inst, &slave_config);

                /* Configure the SPI master. */
                spi_get_config_defaults(&config);
                config.mux_setting = CONF_WIFI_M2M_SPI_SERCOM_MUX;
                config.pinmux_pad0 = CONF_WIFI_M2M_SPI_PINMUX_PAD0;
                config.pinmux_pad1 = CONF_WIFI_M2M_SPI_PINMUX_PAD1;
                config.pinmux_pad2 = CONF_WIFI_M2M_SPI_PINMUX_PAD2;
                config.pinmux_pad3 = CONF_WIFI_M2M_SPI_PINMUX_PAD3;
                config.master_slave_select_enable = false;

                config.mode_specific.master.baudrate = CONF_WIFI_M2M_SPI_BAUDRATE;
                if (spi_init(&master, CONF_WIFI_M2M_SPI_MODULE, &config) != STATUS_OK) {
                  return M2M_ERR_BUS_FAIL;
                }
                /* Enable the SPI master. */
                spi_enable(&master);
                return result;
}
```

The **nm_bus_ioctl** is called when the WINC1500 ASIC driver reads or writes the data. In case of SPI interface, only **NM_BUS_IOCTL_RW** command is used.

```
sint8 nm_bus_ioctl(uint8 u8Cmd, void* pvParameter)
{
        sint8 s8Ret = 0;
        switch(u8Cmd)
        {
                case NM_BUS_IOCTL_RW: {
                tstrNmSpiRw *pstrParam = (tstrNmSpiRw *)pvParameter;
                s8Ret = spi_rw(pstrParam->pu8InBuf, pstrParam->pu8OutBuf, pstrParam->u16Sz);
                }
                break;
                default:
                        s8Ret = -1;
                        M2M_ERR("invalide ioclt cmd\n");
                        break;
        }

        return s8Ret;
}
```

As shown, **spi_rw** function is implemented to read or write data in the same file, **nm_bus_wrapper_samd21.c** like the following.

```
static sint8 spi_rw(uint8* pu8Mosi, uint8* pu8Miso, uint16 u16Sz)
{
        uint8 u8Dummy = 0;
        uint8 u8SkipMosi = 0, u8SkipMiso = 0;
        uint16_t txd_data = 0;
        uint16_t rxd_data = 0;
        if (!pu8Mosi) {
                pu8Mosi = &u8Dummy;
                u8SkipMosi = 1;
        }
        else if(!pu8Miso) {
                pu8Miso = &u8Dummy;
                u8SkipMiso = 1;
        }
        else {
                return M2M_ERR_BUS_FAIL;
        }
        spi_select_slave(&master, &slave_inst, true);
        while (u16Sz) {
                txd_data = *pu8Mosi;
                while (!spi_is_ready_to_write(&master))
                        ;
                while(spi_write(&master, txd_data) != STATUS_OK)
                        ;

                /* Read SPI master data register. */
                while (!spi_is_ready_to_read(&master))
                        ;
                while (spi_read(&master, &rxd_data) != STATUS_OK)
                        ;
                *pu8Miso = rxd_data;

                u16Sz--;
                if (!u8SkipMiso)
                        pu8Miso++;
                if (!u8SkipMosi)
                        pu8Mosi++;
        }
        while (!spi_is_write_complete(&master))
                ;
        spi_select_slave(&master, &slave_inst, false);
        return M2M_SUCCESS;
}
```

The wrapper function provides **nm_bus_deinit** as well. If required in specific platform, the function should also be implemented in the wrapper file. If not required, just return zero like the following sample code.

```
sint8 nm_bus_deinit(void)
{
    return 0;
}
```

5. Conclusion

This application note described how to port SPI interface of the Atmel WINC1500 wing board on the SAMD21 board. The sample codes can be downloaded from Atmel Studio. It will be very helpful to port SPI slave driver in any MCUs. The Appendix A shows how to download WINC1500 examples from Atmel Studio. The simple example is introduced to verify the SPI slave interface.

6. Revision History

Doc. Rev.	Date	Comments
XXXXXA	10/2015	Initial document release

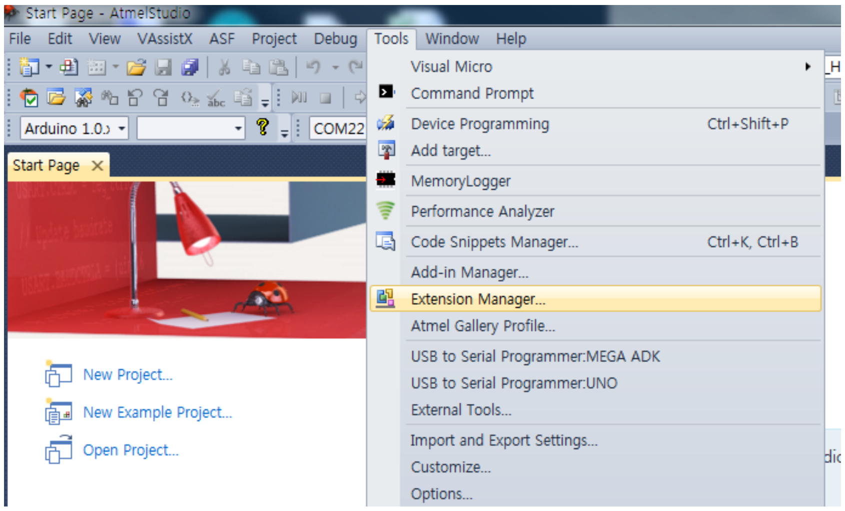
7. Appendix A: Simple Example of ATWINC1500

This appendix shows how to refer to the WINC1500 examples from Atmel Studio. The Atmel Studio is essential to work on WINC1500 with SAM D21 board. Download and install Atmel Studio [3]. Then, refer to the Atmel Studio online help [5] for more instructions.

The followings describe how to find WINC1500 examples in Atmel Studio.

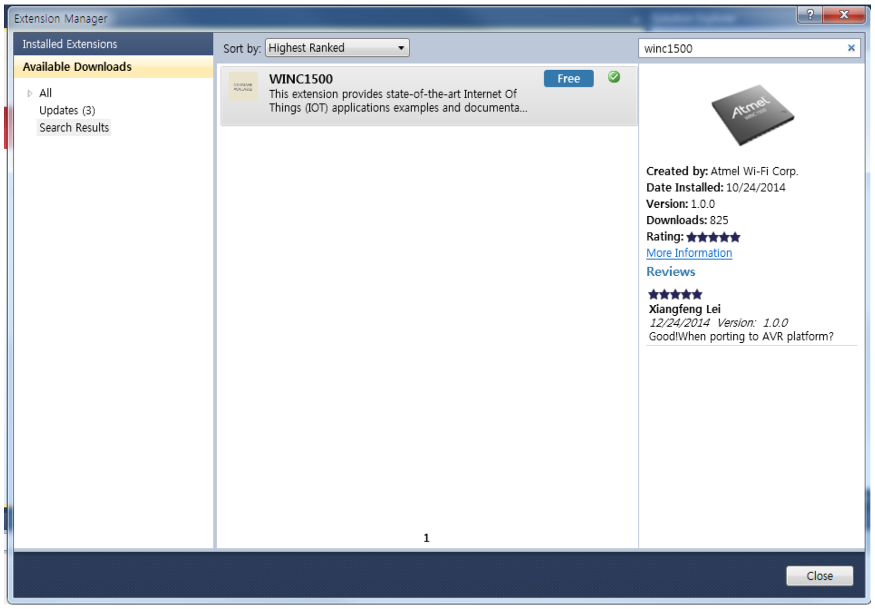
- Launch the Atmel Studio
- Click Tools menu and select Extension Manager

Figure 7-1 Atmel Studio Selecting Extension Manager



- Search WINC1500 in Extension Manager and install WINC1500.

Figure 7-2 Extension Manager WINC1500

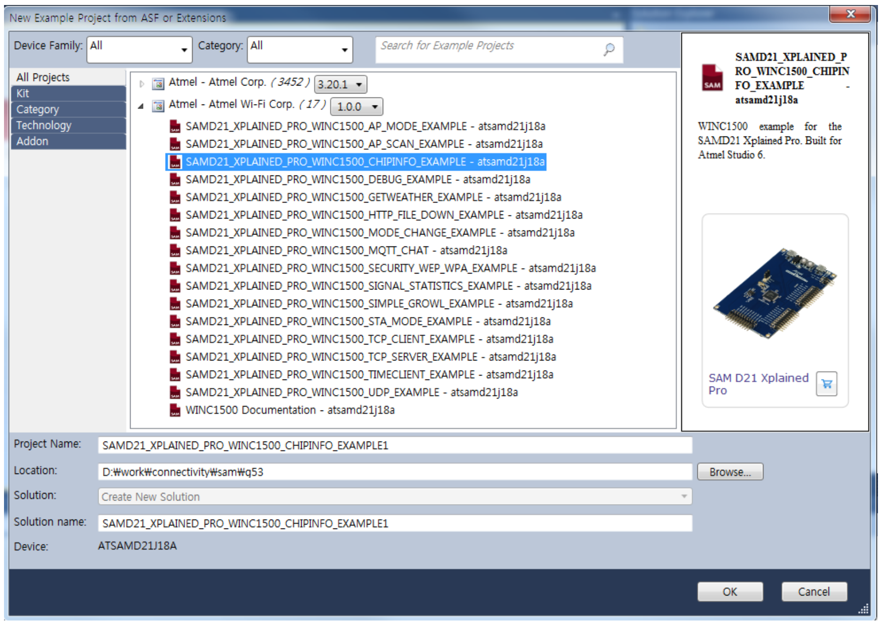


WINC1500
This extension provides state-of-the-art Internet Of Things (IOT) applications examples and documenta...

Created by: Atmel Wi-Fi Corp.
Date Installed: 10/24/2014
Version: 1.0.0
Downloads: 825
Rating: ★★★★★
More Information
Reviews
★★★★★
Xiangfeng Lei
12/24/2014 Version: 1.0.0
Good!When porting to AVR platform?

- Click File menu and select New→ Example Project.
- Then install SAMD21_XPLAINED_PRO_WINC1500_CHIPINFO_EXAMPLE.

Figure 7-3 New Example Project from ASF or Extension



New Example Project from ASF or Extensions

Device Family: All   Category: All   Search for Example Projects

All Projects
Kit
Category
Technology
Addon

- Atmel - Atmel Corp. (3452) 3.20.1
- Atmel - Atmel Wi-Fi Corp. (17) 1.0.0
  - SAMD21_XPLAINED_PRO_WINC1500_AP_MODE_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_AP_SCAN_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_CHIPINFO_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_DEBUG_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_GETWEATHER_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_HTTP_FILE_DOWN_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_MODE_CHANGE_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_MQTT_CHAT - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_SECURITY_WEP_WPA_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_SIGNAL_STATISTICS_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_SIMPLE_GROWL_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_STA_MODE_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_TCP_CLIENT_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_TCP_SERVER_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_TIMECLIENT_EXAMPLE - atsamd21j18a
  - SAMD21_XPLAINED_PRO_WINC1500_UDP_EXAMPLE - atsamd21j18a
  - WINC1500 Documentation - atsamd21j18a

SAMD21_XPLAINED_PRO_WINC1500_CHIPINFO_EXAMPLE - atsamd21j18a

WINC1500 example for the SAMD21 Xplained Pro. Built for Atmel Studio 6.

SAM D21 Xplained Pro

Project Name: SAMD21_XPLAINED_PRO_WINC1500_CHIPINFO_EXAMPLE1
Location: D:₩work₩connectivity₩sam₩q53   Browse...
Solution: Create New Solution
Solution name: SAMD21_XPLAINED_PRO_WINC1500_CHIPINFO_EXAMPLE1
Device: ATSAMD21J18A

OK   Cancel

This install will make new project in specified location for simple example to read the chip id. If the SPI slave interface is not working, it fails to read the chip id. For more information on this example, refer to the Atmel gallery [6].

The simple example has **nm_bus_wrapper_samd21.c** file in the /src/winc/bus_wrapper/source. This file is made to port the SPI slave interface driver for WINC1500 ASIC driver. This file contains three implementations for **nm_bus_init**, **nm_bus_ioctl** and **nm_bus_deinit** as explained in this application note.

The main function has very simple implementation. After calling **m2m_wifi_init**, print the chip id to check if the bus interface has no problem.

8. Appendix B: References

[1] SAM D21 Xplained Pro Evaluation Kit

[2] ATWINC1500

[3] Atmel Studio

[4] WINC1500 datasheet

[5] Atmel Studio online help

[6] Atmel Gallery for WINC1500

Atmel | Enabling Unlimited Possibilities®