megaAVR Microcontrollers

AT13041: AVR for IoT - Using megaAVR with ATWINC1500 802.11 b/g/n Wi-Fi Network Controller Module

APPLICATION NOTE

Description

This application note describes how to use the Atmel® ATmega328P Xplained Mini with the ATWINC1500 Xplained Pro Wi-Fi® network controller to control a LED via an Android application.

Table of Contents

1. Introduction

This application note and the related example project will show the software and hardware implementation of a Wi-Fi controlled LED using the ATmega328P Xplained Mini with the ATWINC1500 Xplained Pro. It will also be described how the application can be extended to transmit temperature measurements.

The related example project is available as a .zip-file from atmel.com and is implemented by using Atmel Studio and Atmel Software Framework (ASF).

1.1. Prerequisites

Running the example described in this application note requires the following:

- Atmel Studio 6.2 Service Pack 2
- ATmega328P Xplained Mini
- ATWINC1500 Xplained Pro with FW19.3
- I/O1 Xplained Pro
- Device to run Android application
- Soldering iron and solder
- Wires
- 0Ω resistor
- Two angled 100 mil headers
- Single pin header

2. Example Overview

The following sections will show how to modify the ATmega328P Xplained Mini to connect the ATWINC1500 Xplained Pro and I/O1 Xplained Pro extension wings. The example project and Android application will also be detailed.

The example project will transmit UDP (User Datagram Protocol) packets at given intervals for the Xplained Mini to be discovered by the Android application. The application can then connect via TCP (Transmission Control Protocl) and send a message to toggle the LED on the I/O1 Xplained Pro.

2.1. Hardware

The following sections will describe the different Xplained Pro modules the example uses, as well as explain how to modify the Xplained Mini board to connect these to each other.

2.1.1. ATmega Xplained Mini

The Atmel ATmega328P Xplained Mini evaluation kit is a hardware platform for evaluating the ATmega328P microcontroller. The evaluation kit comes with a fully integrated debugger that provides seamless integration with Atmel Studio 6.2 or later. All other megaAVR® Xplained Mini kits can also be used for the demo application described in this application note.

2.1.2. I/O1 Xplained

The I/O1 Xplained Pro provides a light sensor, temperature sensor, and microSD card. It connects to the extension headers of any Xplained Pro board.

2.1.3. ATWINC1500 Xplained

The Atmel ATWINC1500 Xplained is a hardware platform for evaluating the ATWINC1500 low cost, low power 802.11 b/g/n Wi-Fi network controller module. Supported by the Atmel Studio integrated development platform, the kit provides easy access to the features of the ATWINC1500 and explains how to integrate the device in a custom design.

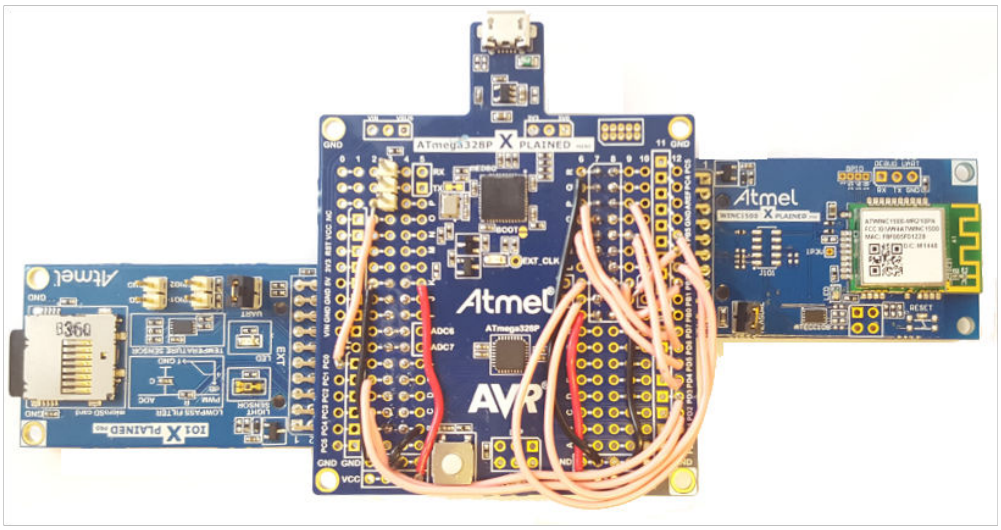
Atmel SmartConnect-WINC1500 is an IEEE® 802.11 b/g/n IoT network controller SoC. It is the ideal add-on to existing MCU solutions bringing Wi-Fi and Network capabilities through UART , I^2C, or SPI-to-Wi-Fi interface. The ATWINC1500 connects to any Atmel AVR® or SMART MCU with minimal resource requirements. The ATWINC1500 most advanced mode is a single stream 1x1 802.11n mode providing up to 72Mbps PHY throughput. ATWINC1500 features fully integrated Power Amplifier, LNA, Switch, and Power Management. The ATWINC1500 provides internal Flash memory as well as multiple peripheral interfaces including UART, SPI, and I^2C.

2.2. Hardware Modifications

To connect the ATWINC1500 Xplained Pro and the I/O1 Xplained Pro to the Xplained Mini, some modifications needs to be done to the hardware. To do the modifications you will need wires, one 0Ω resistor, as well as two angled 100 mil headers and one pin header to solder to the Xplained Mini board. The single pin header can be used to connect the board to any external component that can be controlled by a pin, and is thus optional to add.

After the modifications are done, the I/O1 Xplained Pro can be connected to the left extension header and the ATWINC1500 Xplained Pro to the right extension header as shown in the following figure.

Figure 2-1 Xplained Mini Board with HW Modifications

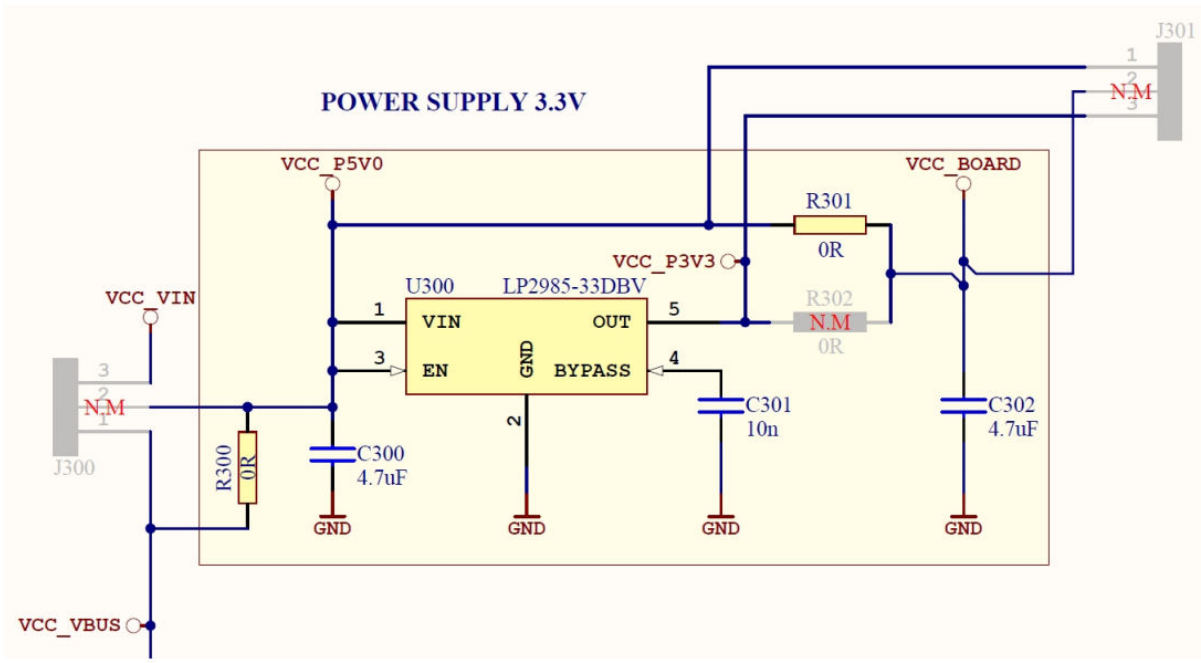


2.2.1. Power Supply

The ATWINC1500 has an operating voltage from 3.0 to 4.2V. As the Xplained Mini board has a supply voltage of 5V, it is important that the voltage is changed to the more appropriate 3.3V before we connect the ATWINC1500 Xplained Pro to the board.

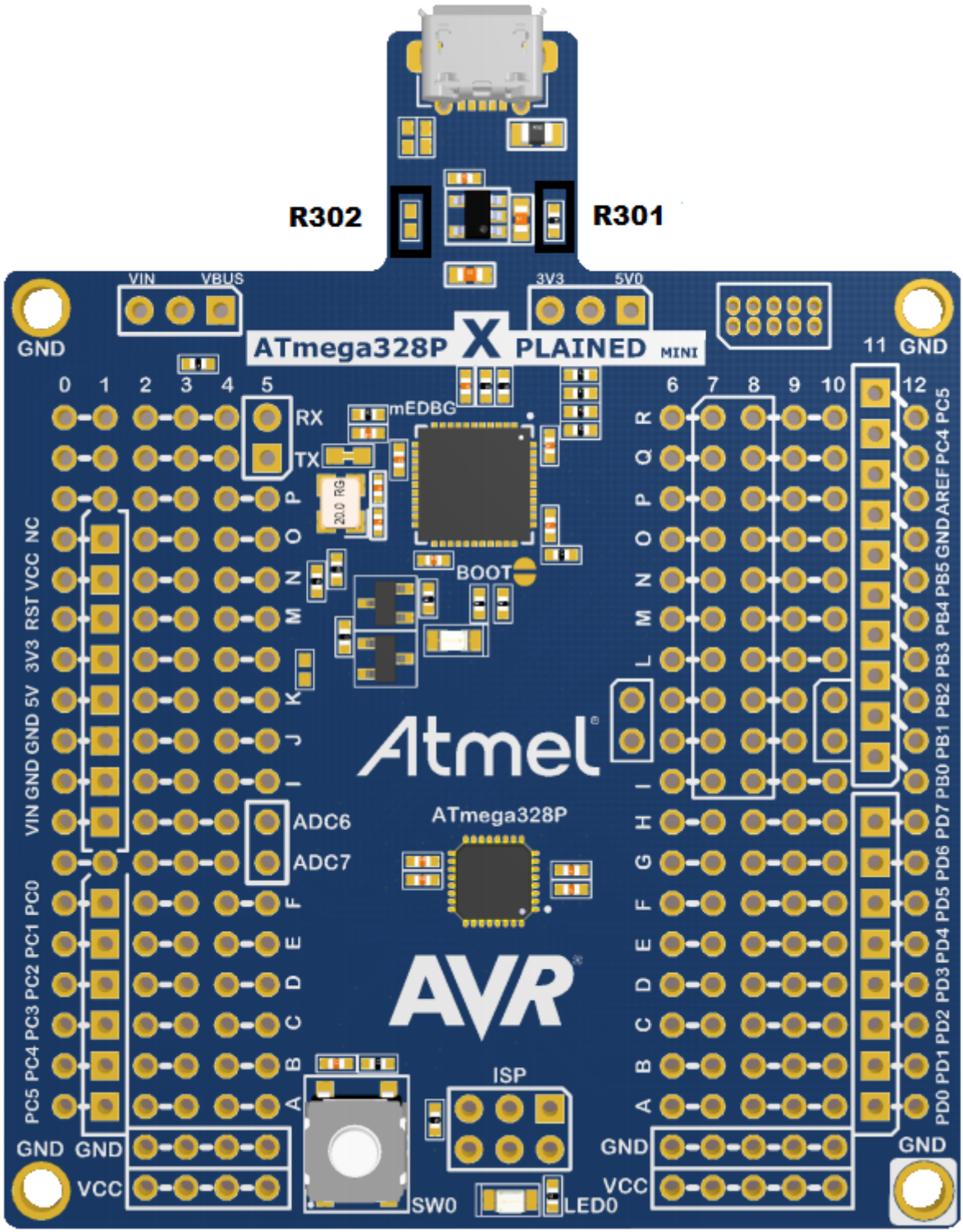
The figure below shows the 3.3V power supply schematics.

Figure 2-2 Xplained Mini Power Supply 3.3V



To modify the Xplained Mini's supply voltage from 5 to 3.3V, remove the 0Ω resistor R301 and add a 0Ω resistor R302. These resistors are located near the USB connector on the board, and are marked in the figure below.

Figure 2-3 Xplained Mini Board with Power Supply Resistors

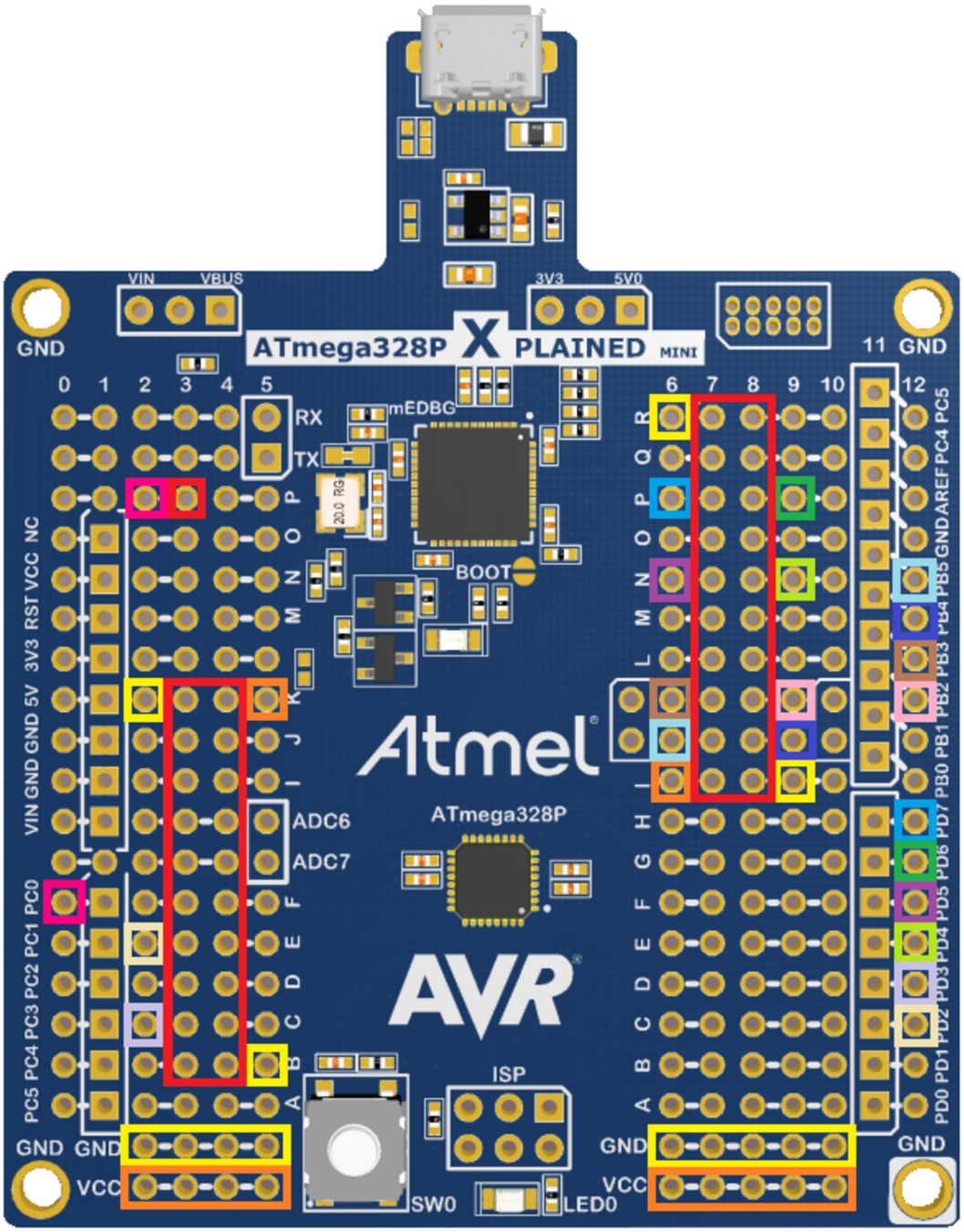


2.2.2. Extension Headers

Two extension headers needs to be soldered to the Xplained Mini board. This should be 10x2 angled 100 mil headers, and they must be soldered to the bottom of the board. The optional single pin header should be soldered to the top of the board.

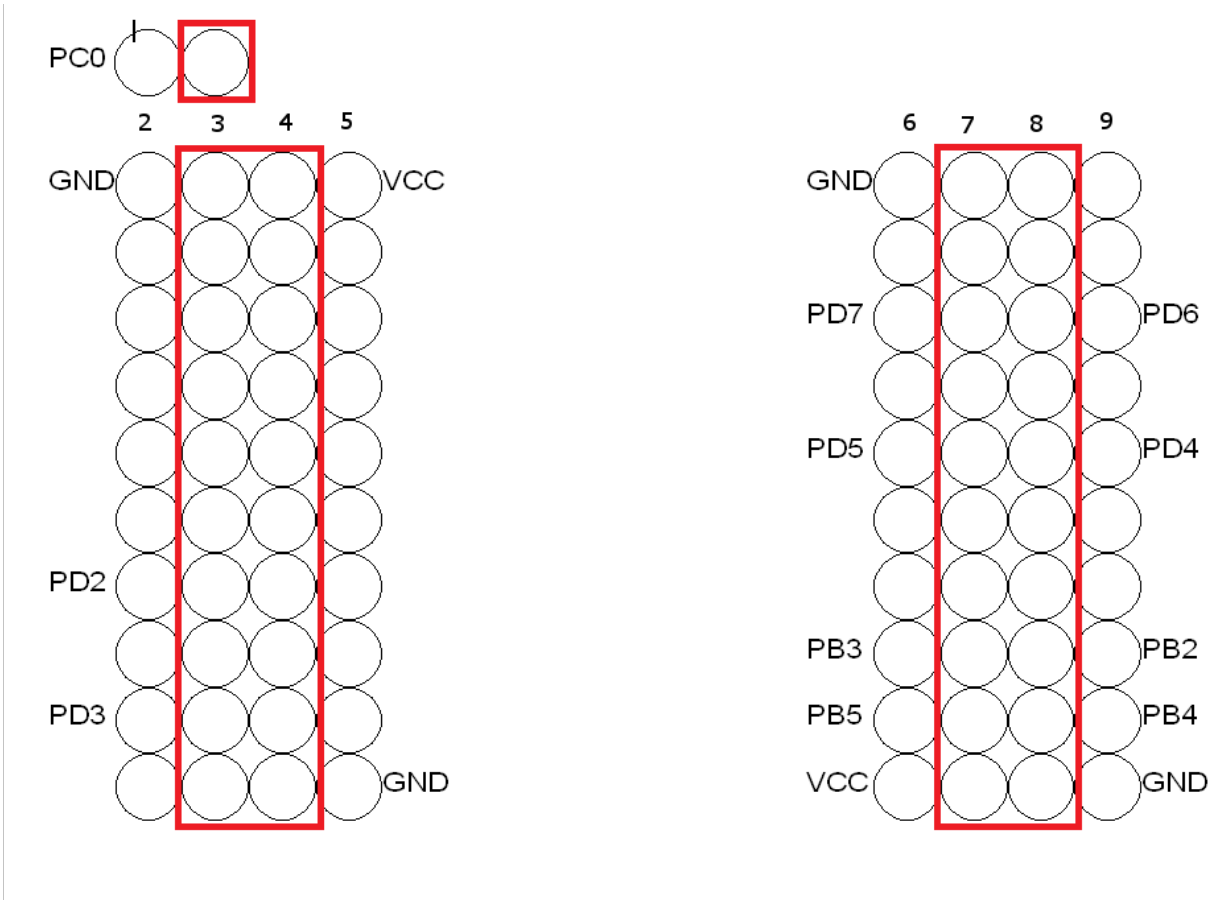
In the figure below, the connections that need to be made are color coded. The red markings show where the headers should be placed. The connections should be made from the hole next to the pin header to the hole marked with the corresponding color on the sides of the board. There are several connections to GND and VCC, marked with yellow and orange. These connections should go from the hole next to the pin headers to any of the GND and VCC holes marked at the bottom of the board.

Figure 2-4 Xplained Mini Board with Connections



In the following figure, which pin that should be wired to which pin header is marked with the name of the pin. Note that even though the extension headers should be soldered to the bottom of the board, the drawings are from the top. To make the soldering process easier, it is recommended to solder the wires to the top of the board and to solder the wires before the extension headers.

Figure 2-5 Pin Connections



2.3. Software

The software used in this demo application consists of three parts which will be described in the following sections:

- megaAVR application
- ATWINC1500 firmware
- Android application

A general description of the application and how to use it will be given after this.

2.3.1. megaAVR

The Atmel Studio project can be found in the Studio Project folder in the zip-file accompanying this application note, and is called WINC_mega. By changing the device in the Device tab of the project properties (Alt+F7), it can be recompiled for any megaAVR Xplained Mini board.

The megaAVR communicates with the ATWINC1500 module via SPI and uses Timer/Counter1 to time when to send UDP packets.

The Xplained Mini board can be programmed and debugged with the onboard embedded debugger and Atmel Studio. The WINC_mega_328p.hex file found in the Studio Project\programmable folder has been compiled for the ATmega328 and can be programmed directly via ISP if you are using an ATmega328 device.

The fuse settings must be set as follows:

- BOOTSZ = 256W_3F00
- BOOTRST = [ ]
- RSTDISBL = [ ]
- DWEN = [ ]
- SPIEN = [X]
- WDTON = [ ]
- EESAVE = [ ]
- BODLEVEL = DISABLE
- CKDIV8 = [ ]
- CKOUT = [ ]
- SUT_CKSEL = EXTCLK_6CK_14CK_65MS

2.3.2. ATWINC1500

The ATWINC1500 firmware includes a Host Interface Layer, running on top of a set of components, including a real-time operating system (RTOS), a light-weight network stack, middleware services and a MAC control driver. The ATWINC1500 firmware version supported for this application is 19.3.

The ATWINC1500 can interface a Host MCU using a serial interface allowing a Host MCU application with limited memory and CPU capabilities to access a wide range of connectivity features from peer -to-peer connections, LAN communication and communication with cloud servers.

The ATWINC1500 firmware exchanges binary messages with Host MCU application via SPI interface. The firmware encapsulates several types of Host InterFace (HIF) messages such as device-specific initialization, WLAN configuration (scan, connection, disconnection) and network traffic using a BSD-like socket interface (socket(), close(), bind(), listen(), accept(), send(), sendto(), recv(), recvfrom()).

The ATWINC1500 HIF uses a “push” architecture, where data and events are pushed from ATWINC1500 firmware to the Host MCU in First-come, first-served manner (FCFS). The ATWINC1500 firmware will interrupt the Host MCU using a dedicated GPIO line whenever one or more events occur. The above architecture allows the ATWINC1500 Firmware driver to be flexible enough to run in the following configurations:

- **No operating system configuration** – The Host MCU application main loop is responsible for updating the WINC host driver state machine by calling a specific function to handle pending events and callbacks.
- **Operating system configuration** – A dedicated task can be implemented to handle deferred work from the interrupt handler by calling a specific function to handle pending events and callbacks.

2.3.2.1. Wi-Fi Software API

The ATWINC1500 Wi-Fi module comes with a Wi-Fi software API for host MCU. The purpose of this API is to provide an abstraction of the binary protocol used between the host MCU and the ATWINC1500 Wi-Fi module while keeping an easy and reliable solution to add wireless capabilities to any user application.

The Host MCU application should be based on a state machine which processes received data and events from the ATWINC1500 module. The ATWINC1500 Host Driver requires the following software components to run on the Host MCU beside the application code:

- **Board Support Package (BSP)** – with a Host MCU serial SPI bus driver and interrupt logic for ATWINC1500 module.
- **ATWINC1500 Controller driver** – which handles the communication with the ATWINC1500 chip via HIF messages. It also provides a C API interface to Host Application.

2.3.2.2. Porting

By changing or adding a few files and implementing the SPI bus, the ATWINC1500 API can be used with any MCU. The following files have been changed or added to use the ATWINC1500 with ATmega328P:

- **winc/bsp/include/nm_bsp.h** - Must be updated to add a new device, and include the nm_bsp_* header file for the particular device.
- **winc/bsp/include/nm_bsp_mega.h** - Must be added with device-specific configuration.
- **winc/bsp/source/nm_bsp_mega.c** - Must be added with implementation of the function declarations from nm_bsp.h. This includes setting up the output pins used for resetting and enabling the ATWINC1500, as well as the interrupt input pin used by the ATWINC1500 to signal to the MCU that an even needs to be handled.
- **winc/bus_wrapper/source/nm_bus_wrapper_mega.c** - Must be added to implement the function declarations from nm_bus_wrapper.h. This includes functions to initialize and use the SPI bus to communicate with the ATWINC1500.
- **config/conf_winc.h** - Must be added with device- and hardware-specific configuration.

2.3.2.3. FW Upgrade

The ATWINC1500 module must be upgraded to firmware version 19.3 for the application to work.

The firmware upgrade process consists of first programming a serial bridge application to the megaAVR on the Xplained Mini board. This will then act as an interface between the virtual com port and the ATWINC1500 module and transfer the firmware from the computer to the ATWINC1500.

To do this, make sure that the Xplained Mini is connected to the computer with the USB cable, and that the ATWINC1500 is connected to the pin headers on the right side of the board. By running the .bat script found in the WINC1500_FW_UPDATE folder in the .zip file accompanying this application note, the serial bridge will be programmed to the megaAVR and the new firmware will be transferred to the ATWINC1500. For the ATmega328P, the script to run is "mega328p_xplained_mini_firmware_update.bat". Update scripts for other Xplained boards are also included in this folder.

Troubleshooting

If there are any issues during the FW upgrade process, check the following points:

- Note that the upgrade process will fail if DebugWire is enabled on the megaAVR. Make sure the DW fuse is un-programmed, and that the ISP fuse is programmed.
- The mEDBG Virtual COM Port must not be opened in any terminal programs, such as PuTTY or TeraTerm
- Make sure that the mEDBG Virtual COM Port driver is installed and shows up in the device manager. If it is not shown, download the drivers via the Extension Manager in Atmel Studio: Tools → Extension Manager and search for the Atmel USB Installer.

2.3.3. Android Application

The Android application AVR for IoT is found as an apk-file in the zip-file accompanying this application note. To install the application, save the apk-file to an Android device and open it from the device. This will install the application. If the install is blocked, go to Settings → Lock screen and security and turn on "Unknown sources".

The application logo is shown in the figure below.

Figure 2-6 AVR for IoT Application Logo

2.3.4. Program Flow

When running the application for the first time, the ATWINC1500 needs to be connected to a wireless network. To provide the SSID and password of the available Wi-Fi to the ATWINC1500, it must be placed in provisioning mode. This is done by holding down the push-button on the board when powering it up. When the LED on the I/O1 Xplained is lit, the button can be let go. Now that the ATWINC1500 is placed in provisioning mode, it will create its own access point called AVR_for_IoT.

Provisioning mode is built-in functionality of the ATWINC1500, and can be started with a function call in software. In this mode, the ATWINC1500 acts as an access point with the sole purpose of displaying a web page, which can be used to type in the credentials to a network. The application will get a callback when the ATWINC1500 has received the credentials, and this can then be used to connect to the network.

To supply the user-name and password of the available network, there are two options. The first is to use a phone or a computer to connect to the AVR_for_IoT access point and open a web browser, which will open a provisioning page as shown in the figure below.

Figure 2-7 ATWINC1500 Provisioning Page

Type in the SSID and password of the network and press the "Connect" button, which will send the credentials to the ATWINC1500. Note that the Device Name field is unused. The ATWINC1500 firmware

will disable the access point and send the information about the SSID and password to the application. As the access point is disabled, the phone or computer will be disconnected from the Wi-Fi, and an error message might be shown in the browser. When the credentials is received in the application, the LED will be turned off. The application will then try to connect to the provided Wi-Fi network.

Instead of manually connecting to the AVR_for_IoT access point, the other option is to use the Android application. Open the Android app (AVR for IoT), and press the "WIFI TO WINC" button as shown in the figure below. When doing this, the phone should be connected to the normal Wi-Fi network, which the ATWINC1500 should connect to.

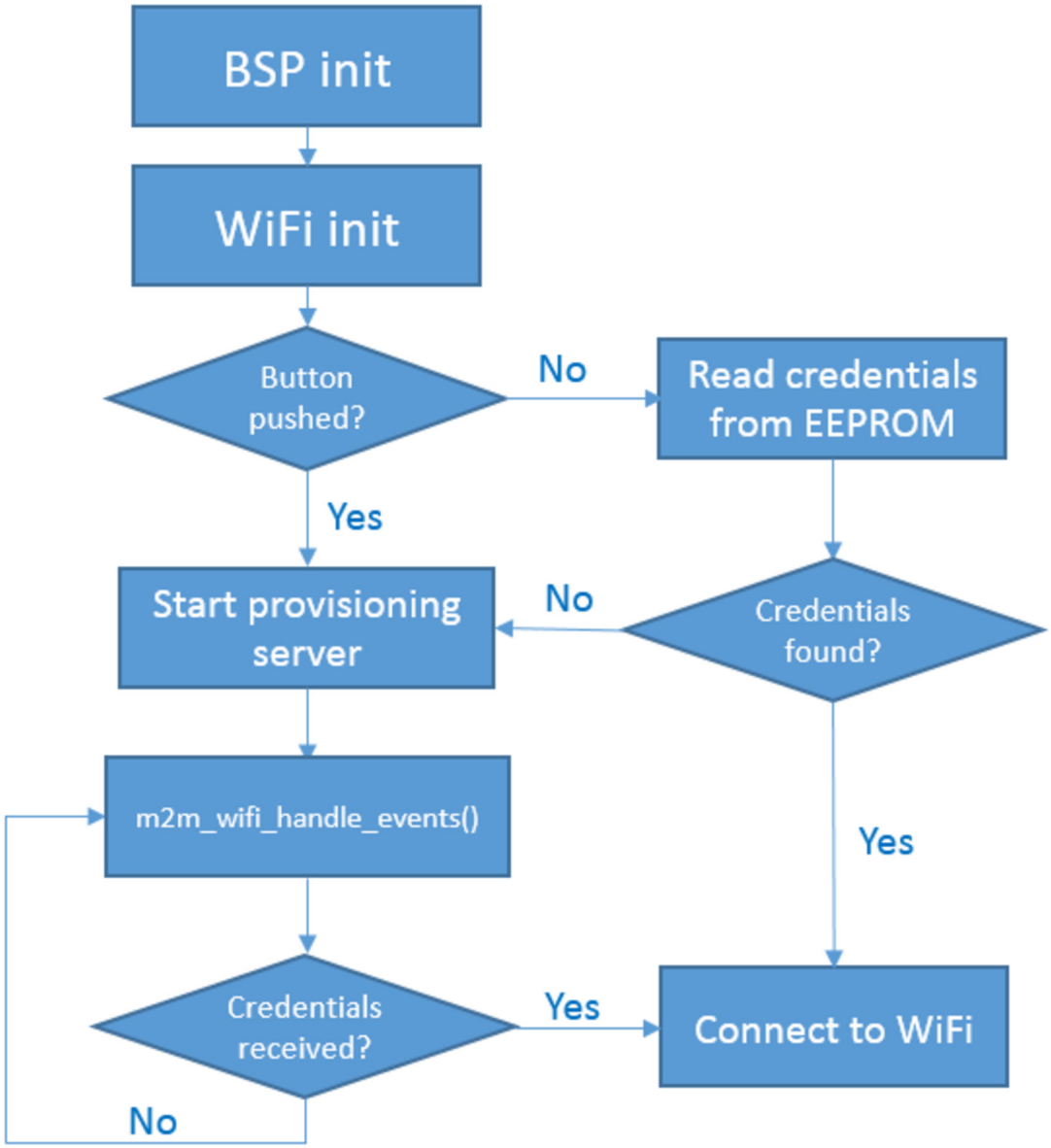
Figure 2-8 Atmel ToggleDemo Main Screen

This will let you write the SSID and password of the network the ATWINC1500 should connect to. When the "WIFI TO WINC" button is pressed again, the Android phone will disconnect from any Wi-Fi currently connected to, connect to the AVR_for_IoT access point, and send the credentials to the ATWINC1500. When this is done, it will reconnect to the network it was previously connected to.

When the Wi-Fi credentials have been received, the MCU will store it in EEPROM, so that it is not necessary to repeat the provisioning process each time the MCU is powered on. If the push-button on the Xplained Mini is not pressed during power on, the SSID and password will be read from EEPROM if present, and the application will try to connect with these credentials directly.

The figure below shows a flowchart of the initialization and provisioning process.

Figure 2-9 Flowchart of Initialization and Provisioning

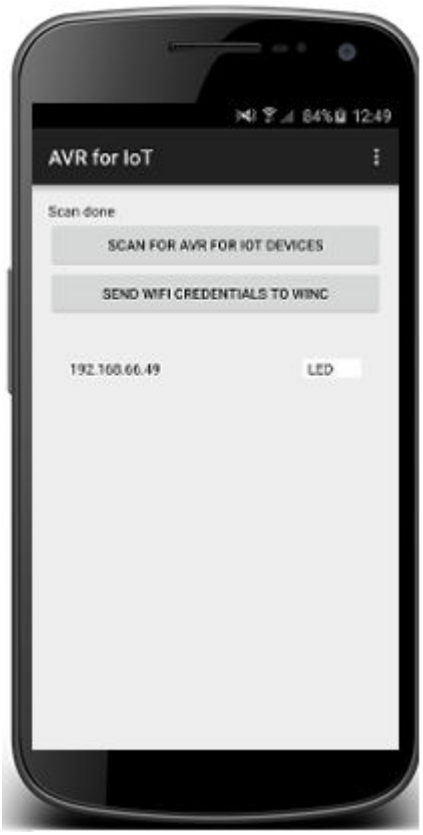


When the application is connected to the access point and has gotten an IP address, it will configure a timer and use this to broadcast UDP packets every four seconds. As these packets are broadcasted on the network, any Android phone or computer connected to the same access point will receive these packets. To let the Android application listen for UDP packets from the Xplained Mini, press the "SCAN" button.

This will initiate a scan on the network for approximately six seconds, and then show any nearby Xplained Mini with ATWINC1500 devices connected to the same wireless network as the Android phone. The IP address of any available device is shown, and when this is pressed, the LED on the I/O1 Xplained will toggle.

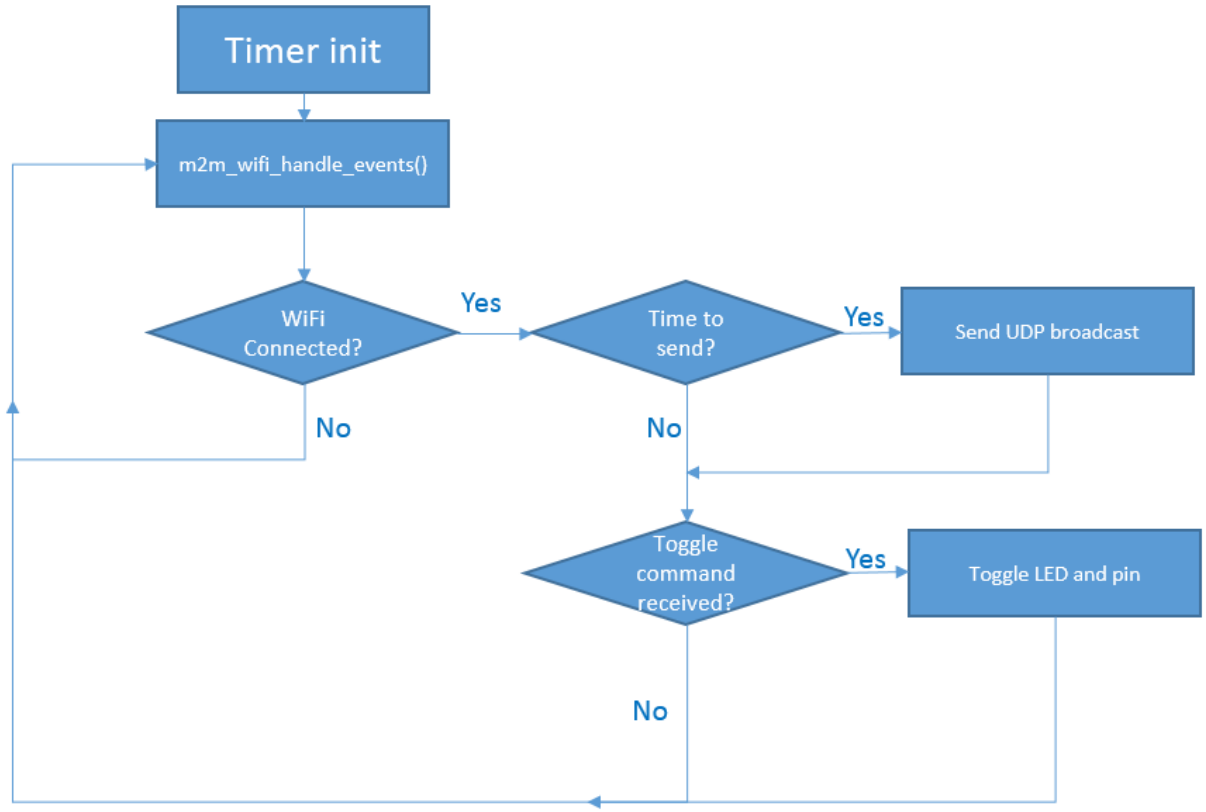
The pin header on the top left side of the Xplained Mini board will also toggle, allowing the demo to be connected to anything external that can be controlled by a pin.

Figure 2-10 Atmel ToggleDemo Scan Done



The figure below shows a flowchart of the main loop of the application.

Figure 2-11 Flowchart of Application Main Loop



2.3.5. Protocol

Every four seconds, the application running on the megaAVR sends out an UDP packet on port 6789 to the broadcast address 255.255.255.255. This packet consists of a predefined buffer, which is also known by the Android application.

```
/** Key buffer - must match with Android app */
uint8_t key_buf[] = KEY_BUFFER;
```

This makes the Android application able to distinguish between UDP broadcast packets from the demo application and any other broadcasts on the network. Broadcasts that matches the key buffer will be added to the list of available devices the Android application displays after a scan.

The megaAVR will open a TCP socket to listen for incoming requests as soon as it is connected to the Wi-Fi. When the IP address of a device is pressed in the Android application, it will connect to the Xplained Mini via a TCP socket on port 1234, and send a message including the word "toggle", which is the keyword the megaAVR will look for when receiving messages on the TCP socket. When this is received by the megaAVR, it will toggle the LED and the toggle pin. When this is done, it will send a confirmation message back to the Android application. If this message is not received, or any other error occurs during the transmission, the Android application will highlight the IP address with a red box, indicating that something went wrong.

3. Further Development of the Application

The following section will describe how the application can be further extended to also send temperature data to the Android application. The Android application is already set up for this, and it can be enabled from the app. This is done by pressing the three dots in the top right corner of the main screen, selecting settings and checking the "Enable temperature reading" checkbox as shown in the figure below.

Figure 3-1 Enable Temperature Reading in Android Application

This will add another button to each item in the list, which can be clicked to request temperature data from the Xplained Mini board. When this button is pressed, the Android application will send a TCP packet to the ATWINC1500 with the text "temp". The handling of this message needs to be implemented in the WINC_mega.c file in the Atmel Studio project. The Atmel Studio project, "WINC_mega6_2.atsln", can be found in the "Studio Project\WINC_mega" folder in the .zip file that comes with the application note.

The first thing that should be done is to add two functions to WINC_mega.c; one for initializing the ADC and one for doing an ADC conversion to read the internal temperature sensor of the megaAVR.

```
static void init_temp(void)
{
    sysclk_enable_module(POWER_RED_REG0, PRADC_bm);
    /* Init ADC to read from internal temperature sensor */
    ADMUX =  (1 << MUX3) | (1 << REFS0) | (1 << REFS1);
    /* Enable ADC */
```

```
    ADCSRA = 1 << ADEN | 1 << ADPS0 | 1 << ADPS1 | 1 << ADPS2;
}
```

```
static void read_temp(uint8_t *buffer)
{
    /* Start conversion */
    ADCSRA |= 1 << ADSC;

    /* Wait for conversion to be done */
    while(!(ADCSRA & (1 << ADIF)));
    /* Write conversion result to buffer */
    ((uint16_t*)buffer)[0] = ADCW;
}
```

We will also need a variable to signal to the main loop whether a request to send the temperature has been received from the Android application. This variable should be global and placed outside the main function:

```
/** Variable for main loop to see if a temperature read request is
received */
volatile bool temperature = false;
```

At the start of the main function, a buffer should be declared. This will be used to store the temperature that should be sent to the Android application when requested.

```
/** Buffer to store temperature */
uint8_t temp_buf[2];
```

In the socket callback function m2m_tcp_socket_handler(), we will add the logic to handle the new message from the Android application. As previously stated, the message sent is "temp", so we will need to check if this string is present in the received message. If it is, we will set the global variable to true, so that it can be handled in the main loop. Find the SOCKET_MSG_RECV case in the switch case and replace it with the following code:

```
/* Message receive */
case SOCKET_MSG_RECV:
    /* Check command */
    if (strstr(gau8SocketTestBuffer, "toggle")) {
        toggle = true;
    } else if (strstr(gau8SocketTestBuffer, "temp")) {
        temperature = true;
    }
    break;
```

After doing this, we will need to call the ADC initialization function from the main function, this must be done before the beginning of the while(1) loop.

```
init_temp();
```

The last thing to do is to add the temperature reading and sending it back to the Android application when it has been requested. The following code should be placed inside the while(1) loop, inside the if (wifi_connected == M2M_WIFI_CONNECTED) statement, right after the similar code for the LED toggling:

```
/* Temperature command received  */
if (temperature) {
    temperature = false;
```

```
    read_temp(temp_buf);
    /* Sending temperature reading back */
    send(tcp_client_socket, (void*)temp_buf, sizeof(temp_buf), 0);
}
```

The value sent from the AVR to the Android application will be the raw value from the ADC sampling the temperature sensor. This will thus not be temperature in degrees, but when heating or cooling the AVR, there will be a difference in the readings. For more information about the temperature sensor, refer to the device datasheet.

A WINC_mega.c file with all these implementations added can be found in the !Studio Project \temperature" folder in the .zip file, and can be used as a reference.

4. Revision History

Doc Rev.	Date	Comments
42552A	09/2015	Initial document release.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**