Smart Technology. Delivered.™

Reference Guide

WB50NBT

Version 1.3

Revision History

Version	Date	Notes	Approver
1.0	04 Apr 2016	Initial Release	Steve deRosier
1.1	10 Apr 2017	Changed *Hardware Integration Guide* references to *Datasheet* Added *WB Security* section	Andrew Dobbing
1.2	1 May 2017	OS Support	Jay White
1.3	17 Nov 2017	Updated DCAS authentication information	Andrew Dobbing

CONTENTS

1 Introduction

1.1 Overview

The Laird WB50NBT wireless bridge module is a wireless communications subsystem that may be integrated into a variety of host devices via several available electronic and logical interfaces.

Interfaces	Features	Specifications
▪ Fast Ethernet ▪ Serial UART ▪ USB ▪ SPI (master) ▪ I2C ▪ MMC/SDIO ▪ GPIO ▪ Bluetooth PCM ▪ 120-pin board with mating options	▪ Cortex A5 processor (536MHz) ▪ 64 MB of LPDDR (lower power DDR) memory ▪ 128 MB of NAND flash storage	▪ Length: 47 mm ▪ Width: 37 mm ▪ Height: 4.9 mm

1.2 Product Description

The Laird WB50NBT provides complete enterprise-class Wi-Fi connectivity with an integrated TCP/IP stack, full support for 2 x 2 MIMO IEEE 802.11a/b/g/n, Bluetooth 4.0 dual-mode wireless standards, three-wired coexistence scheme, a fully integrated security supplicant providing 802.11i/WPA2 Enterprise authentication and data encryption, and a BT protocol stack.

The system is a full Linux 4.1.13 kernel based system with modifiable file system allowing custom applications to run alongside system applications. We provide a full source and build system to facilitate customer modifications including new drivers and board support.

The WB50NBT is a fully integrated module with RF shielding and two U.FL type antenna connectors providing two stream MIMO operation for maximum data rate. The main antenna (for Wi-Fi only) and the auxiliary antenna (for Wi-Fi and Bluetooth) work together with the three-wired coexistence scheme to provide the best coexistence performance.

Note: For additional information on the hardware aspects of the WB50NBT, please refer to the *WB50NBT Datasheet* (Hardware Integration Guide) available from the WB50NBT product page of the Laird website.

1.3 Flash Storage

The WB50NBT has 128 MB of NAND flash memory that is divided into partitions (see Table 1). It uses 8-bit ECC per 512-bit sector in a 64 bit OOB area in each 2112 bit page.

Table 1: Flash memory partitions

Image	Partition	Start	End	Size	MTD	Type
at91bs.bin	at91bs	0x00000000	0x0001FFFF	128 KB	/dev/mtd0	Raw binary
u-boot.bin	u-boot	0x00020000	0x0009FFFF	512 KB	/dev/mtd1	Raw binary
-	u-boot-env	0x000A0000	0x000BFFFF	128 KB	/dev/mtd2	U-boot env
-	redund-env	0x000C0000	0x000DFFFF	128 KB	/dev/mtd3	U-boot backup env
kernel.bin	kernel-a	0x000E0000	0x005CFFFF	5 MB	/dev/mtd4	Kernel image

Image	Partition	Start	End	Size	MTD	Type
kernel.bin	kernel-b	0x005E0000	0x00ADFFFF	5 MB	/dev/mtd5	Kernel image
rootfs.bin	rootfs-a	0x00AE0000	0x03ADFFFF	49 MB	/dev/mtd6	UBI
rootfs.bin	rootfs-b	0x03AE0000	0x06ADFFFF	49 MB	/dev/mtd7	UBI
N/A	user	0x06AE0000	0x07FBFFFFF	20.6 MB	/dev/mtd8	Raw
N/A	logs	0x07FC0000	0x07F7FFFF	256 KB	/dev/mtd9	Raw

There are four basic types of binary images that can be programmed into the flash memory:

- Bootstrap loader
- U-boot boot loader
- Linux kernel
- Root filesystem

The flash partition layout allows for two kernel images (kernel-a and kernel-b) as well as two file system images (rootfs-a and rootfs-b). This allows an update to an alternate image without disturbing the currently-running system. Additionally, if something goes awry with the image update process, the original image is still available. The update program marks the newly-updated partition as active at completion and it is used on the next boot.

The filesystem is stored using Unsorted Block Images (UBI) format. This filesystem lies on top of the Memory Technology Device (MTD) layer. The MTD layer handles bad block mapping. When a bad block is encountered, it is simply skipped and not used. As new bad blocks occur, they are marked as such and handled properly between the MTD and UBI file system. The UBIFS handles wear-leveling.

1.4 WB50NBT Usage

One of the main applications of the WB is to enable Wi-Fi on your host device using one of the available interfaces of the WB module. This section describes how to do this from a Linux host environment.

The quickest way to start controlling the WB is through the built-in Linux command line interface (CLI). The WB's CLI can be accessed via the DEBUG UART (settings: 115200 8N1) or via Secure Shell (ssh). ssh logins can be accepted on Ethernet (IP address assigned via DHCP – see your local DHCP server for address) or via USB Ethernet at IP address 192.168.3.1 port 22.

Once CLI access is available, configuration files on the device can be examined or modified.

2 Configuring IP Based Connectivity

This section covers enabling your host device to communicate over Wi-Fi via the WB. The predominant use cases involve using the WB as Ethernet to Wi-Fi, USB to Wi-Fi, or Serial PPP (RS232) to Wi-Fi peripherals. These use cases allow normal IP network connectivity via the WB’s Wi-Fi interface. We’ll explain these use cases in this guide, but they aren’t the only ones available; more complex configurations are also possible. Please contact Laird support if you have a use case this is not covered in this document.

2.1 Choosing an Interface to the WB

The first consideration in configuring the Wi-Fi connectivity is what interface will be used between the host device and the WB. The most common three options are Ethernet, USB Ethernet, and Serial (RS232). Each of these cases has the host device sending and receiving IP packets and the WB forwarding packets to and from the Wi-Fi interface once connected.

- Ethernet – Refers to the standard 802.3 Ethernet.

- USB connectivity – We recommend setting up the WB to enable USB CDC Ethernet (also known as USB Gadget Ethernet). Then installing the proper drivers on the host device. This allows the WB to appear as a standard Ethernet network device.
- Serial (RS232) – We recommend setting up PPP over serial on the WB and the host device and enabling IP over PPP. Optionally, the host device can send raw byte data to a UART on the WB and the WB can forward that data bi-directionally to a TCP socket via the built in socat program.

2.2 Configuring the WB to Use Layer 2 Bridging or Layer 3 NAT

The second consideration in configuring Wi-Fi connectivity is whether to configure the WB to act as a Layer 2 Bridge or a Layer NAT device.

2.2.1 Layer 2 Bridge

When configured as a Layer 2 bridge, the WB acts as a transparent bridge for Layer 2 packets carrying an IP-based packet from the host device. The WB expects the host to do all Layer 3 IP configuration such as static addressing or DHCP for the Wi-Fi network. This mode applies to *Ethernet to Wi-Fi* and *USB Ethernet to Wi-Fi*. In the Layer 2 Bridge configuration, it appears to the host device that it is connected to a physical network.

2.2.2 Layer 3 NAT

When configured to do Layer 3 NAT, the WB is configured to run a DHCP client or have a static IP address on the WB's Wi-Fi interface. In this configuration, a static non-routable IP address is assigned on the host device's communication interface and the corresponding interface on the WB. Port NAT and IP masquerading would then enable seamless communication from the host interface to the Wi-Fi network. This mode applies to *Ethernet to Wi-Fi*, *USB Ethernet to Wi-Fi*, and *Serial PPP to Wi-Fi*. Seven common use cases are detailed that describe the WB configured as a bridge or NAT device.

Note: Please contact Laird support if you have a use case not covered here in this document.

2.3 Choosing Layer 2 Bridge or Layer 3 NAT – Use Cases

2.3.1 Use Case 1

A single host device with a DHCP client for Wi-Fi on the host device's Ethernet or USB interface to the WB.

This is often used when the WB if functioning as an Ethernet to Wi-Fi dongle on an existing product where the Ethernet is set up to use DHCP.

- **Solution:** Bridging
  The WB acts as a transparent bridge between the Ethernet or USB and Wi-Fi.
- **Caveats:**
  It should send a ping packet or other IP packet from the host device's interface to the WB once a new IP address has been assigned to register the host device with the WB's bridge code. Otherwise, the bridge may not forward incoming Wi-Fi packets to the host device.

In addition, this configuration is not able to renew DHCP on a roam without software using SDK Events to alert the host device to trigger DHCP on a roam. This is only an issue for networks configured with multiple subnets on the same SSID (determine if your customers do this).

See Use Case 3 and Use Case 5 to solve this issue.

Use Case 1 Example

2.3.2 Use Case 2

A single host device with a static IP for Wi-Fi on the host device's Ethernet or USB interface to the WB

The IP address that is used on the Wi-Fi network is statically assigned by the host device. This is often the case when using the WB as an *Ethernet to Wi-Fi* dongle on an existing product where the Ethernet was set up to use static IP addresses.

- **Solution:** Bridging
  The WB acts as a transparent bridge between the Ethernet or USB Ethernet and Wi-Fi.
- **Caveats:**
  It should send a ping packet or other IP packet from the host device's interface to the WB using static IP address to register host device with WB's bridge code. Otherwise the bridge may not forward incoming Wi-Fi packets to the host device.

Use Case 2 Example

2.3.3 Use Case 3

A single host device with DHCP running on the WB's Wi-Fi interface

A single device is connected to the Ethernet, USB Ethernet, or PPP serial; a DHCP client is running on the WB to handle the Wi-Fi DHCP assignment.

- **Solution:** NAT
  The WB does IP NAT and port masquerading to forward traffic from the Wi-Fi to host device. Host device interface to the WB along with the corresponding WB interface should be assigned a non-routable 169.254.x.x IP address. The WB does a DHCP request on roam to support networks configured with multiple subnets on the same SSID (determine if your customers do this).

Use Case 3 Example

2.3.4 Use Case 4

A single host device with a static IP on the WB's Wi-Fi interface

- **Solution:** NAT
  The WB does IP NAT and port masquerading to forward traffic from Wi-Fi to the host device. The host device interfaces to the WB, along with the corresponding WB interface, should be assigned a non-routable 169.254.x.x IP address.

Use Case 4 Example

2.3.5 Use Case 5

Multiple host devices connected via Ethernet with DHCP running on the WB's Wi-Fi interface

- **Solution:** NAT
  The WB does IP NAT and port masquerading to forward traffic from the Wi-Fi to each host device. Host device interfaces to the WB along with the corresponding WB interface should be assigned non-routable 169.254.x.x IP addresses. The WB does a DHCP request on roam to support networks configured with multiple subnets on the same SSID (determine if your customers do this).

Use Case 5 Example

2.3.6 Use Case 6

Multiple host devices connected via Ethernet with a static IP on the WB's Wi-Fi interface

- **Solution:** NAT
  The WB does IP NAT and port masquerading to forward traffic from Wi-Fi to each host device. Host device interfaces to the WB, along with the corresponding WB interface, should be assigned non-routable 169.254.x.x IP addresses.

Use Case 6 Example

2.3.7 Use Case 7

Multiple host devices connected via Ethernet with DHCP running or a static IP on each host device interface to the WB

- **Solution:** Use NAT instead
  We recommend that you use a NAT-based configuration instead. The WB does not currently support this use case in the standard release. If this use case is required, please contact Laird with complete details of the use case for potential customized options.

Use Case 7 Example

2.4 WB Configurations – Use Case Examples

Each example configuration uses a combination of the sdc_cli command line utility, the ifrc command line utility, and editing the /etc/network/interfaces file. These can be done once console access has been established with the WB as described in the *WB50NBT Quick Start Guide* (available from the Documentation tab of the Laird WB50NBT product page). These configurations are only examples for the WB's configuration. Example host test device configurations are described in the following section to complete early testing.

Note: Editing of the /etc/network/interfaces file should be done with ***vi***. To learn how to use ***vi*** as an editor, there are many online tutorials.

To begin editing, execute the following:

```
# vi /etc/network/interfaces
```

Note: The following examples describe how to enable *Ethernet to Wi-Fi* connectivity. In these cases, the Ethernet interface is referred to as ***eth0***. To enable *USB Ethernet to Wi-Fi*, substitute ***eth0*** with ***usb0*** and for *PPP Serial to Wi-Fi* substitute ***eth0*** with ***ppp0***. The Wi-Fi interface is ***wlan0*** in the examples.

2.4.1 Use Case 1 Example

A single host device with a DHCP client for Wi-Fi on the host device's Ethernet or USB interface to the WB.

Enable the Ethernet interface and make sure it doesn't use an IP configuration:

```
# sdc_cli iface set auto eth0 on
# sdc_cli iface set method eth0 manual
```

Enable the Wi-Fi interface and make sure it doesn't use an IP configuration:

```
# sdc_cli iface set auto wlan0 on
# sdc_cli iface set method wlan0 manual
```

Enable bridging between Ethernet and Wi-Fi:

```
# sdc_cli iface set auto br0 on
# sdc_cli iface set method br0 manual
# sdc_cli iface set bridge_ports br0 eth0 wlan0
```

Optional: If IP communication between the WB and host device for configuration or otherwise is desired, then additional IP configuration should be enabled on the bridge interface (br0):

```
# sdc_cli iface set method br0 static
# sdc_cli iface set address br0 169.254.0.1
# sdc_cli iface set netmask br0 255.255.255.0
# sdc_cli iface set broadcast br0 169.254.0.255
```

2.4.2 Use Case 2 Example

A single host device with a static IP for Wi-Fi on the host device's Ethernet or USB interface to the WB

Enable the Ethernet interface and make sure it doesn't use an IP configuration:

```
# sdc_cli iface set auto eth0 on
# sdc_cli iface set method eth0 manual
```

Enable the Wi-Fi interface and make sure it doesn't use an IP configuration:

```
# sdc_cli iface set auto wlan0 on
# sdc_cli iface set method wlan0 manual
```

Enable bridging between Ethernet and Wi-Fi:

```
# sdc_cli iface set auto br0 on
# sdc_cli iface set method br0 manual
# sdc_cli iface set bridge_ports br0 eth0 wlan0
```

Optional: If IP communication between the WB and host device for configuration or otherwise is desired, then additional IP configuration should be enabled on the bridge interface (br0):

```
# sdc_cli iface set method br0 static
# sdc_cli iface set address br0 169.254.0.1
# sdc_cli iface set netmask br0 255.255.255.0
# sdc_cli iface set broadcast br0 169.254.0.255
```

2.4.3 Use Case 3 Example

A single host device with DHCP running on the WB's Wi-Fi interface

Enable the Ethernet interface and configure a non-routable IP configuration:

```
# sdc_cli iface set auto eth0 on
# sdc_cli iface set method eth0 static
# sdc_cli iface set address eth0 169.254.0.1
# sdc_cli iface set netmask eth0 255.255.255.0
# sdc_cli iface set broadcast eth0 169.254.0.255
```

Enable the Wi-Fi interface with DHCP client support:

```
# sdc_cli iface set auto wlan0 on
# sdc_cli iface set method wlan0 dhcp
```

Enable NAT between Ethernet and Wi-Fi by editing **/etc/network/inferfaces**. The stanza starting with *auto eth0* should have an uncommented out **post-cfg-do and pre-dcfg-do** sections like the following:

```
## Wired
auto eth0
iface eth0 inet static
  …
  …
    post-cfg-do /etc/network/wifi-nat.conf
    pre-dcfg-do /etc/network/wifi-nat.conf
```

2.4.4 Use Case 4 Example

A single host device with a static IP on the WB's Wi-Fi interface

Enable the Ethernet interface and configure a non-routable IP configuration:

```
# sdc_cli iface set auto eth0 on
# sdc_cli iface set method eth0 static
# sdc_cli iface set address eth0 169.254.0.1
# sdc_cli iface set netmask eth0 255.255.255.0
# sdc_cli iface set broadcast eth0 169.254.0.255
```

Enable the Wi-Fi interface with static IP support:

```
# sdc_cli iface set auto wlan0 on
# sdc_cli iface set method wlan0 static
# sdc_cli iface set address eth0 x.x.x.x
# sdc_cli iface set netmask eth0 x.x.x.x
# sdc_cli iface set broadcast eth0 x.x.x.x
# sdc_cli iface set nameserver eth0 x.x.x.x
```

Enable NAT between Ethernet and Wi-Fi by editing **/etc/network/inferfaces**. The stanza starting with *auto eth0* should have an uncommented out **post-cfg-do and pre-dcfg-do** sections like the following:

```
## Wired
auto eth0
iface eth0 inet static
  …
  …
    post-cfg-do /etc/network/wifi-nat.conf
    pre-dcfg-do /etc/network/wifi-nat.conf
```

2.4.5 Use Case 5 Example

Multiple host devices connected via Ethernet with DHCP running on the WB's Wi-Fi interface

Enable the Ethernet interface and configure a non-routable IP configuration:

```
# sdc_cli iface set auto eth0 on
# sdc_cli iface set method eth0 static
# sdc_cli iface set address eth0 169.254.0.1
# sdc_cli iface set netmask eth0 255.255.255.0
# sdc_cli iface set broadcast eth0 169.254.0.255
```

Enable the Wi-Fi interface with DHCP client support:

```
# sdc_cli iface set auto wlan0 on
# sdc_cli iface set method wlan0 dhcp
```

Enable NAT between Ethernet and Wi-Fi by editing **/etc/network/inferfaces**. The stanza starting with *auto eth0* should have an uncommented out **post-cfg-do and pre-dcfg-do** sections like the following:

```
## Wired
auto eth0
iface eth0 inet static
  …
  …
    post-cfg-do /etc/network/wifi-nat.conf
    pre-dcfg-do /etc/network/wifi-nat.conf
```

2.4.6 Use Case 6 Example

Multiple host devices connected via Ethernet with a static IP on the WB's Wi-Fi interface

Enable the Ethernet interface and configure a non-routable IP configuration:

```
# sdc_cli iface set auto eth0 on
# sdc_cli iface set method eth0 static
# sdc_cli iface set address eth0 169.254.0.1
# sdc_cli iface set netmask eth0 255.255.255.0
# sdc_cli iface set broadcast eth0 169.254.0.255
```

Enable the Wi-Fi interface with static IP support:

```
# sdc_cli iface set auto wlan0 on
# sdc_cli iface set method wlan0 static
# sdc_cli iface set address eth0 x.x.x.x
# sdc_cli iface set netmask eth0 x.x.x.x
# sdc_cli iface set broadcast eth0 x.x.x.x
# sdc_cli iface set nameserver eth0 x.x.x.x
```

Enable NAT between Ethernet and Wi-Fi by editing **/etc/network/inferfaces**. The stanza starting with *auto eth0* should have an uncommented out **post-cfg-do and pre-dcfg-do** sections like the following:

```
## Wired
auto eth0
iface eth0 inet static
  …
  …
    post-cfg-do /etc/network/wifi-nat.conf
    pre-dcfg-do /etc/network/wifi-nat.conf
```

2.4.7 Use Case 7 Example

Multiple host devices connected via Ethernet with DHCP running or a static IP on each host device interface to the WB

Laird recommends to use a NAT-based configuration instead. The WB does not currently support this use case in the standard release. If this use case is required, please contact Laird with complete details of the use case for potential customized options.

2.5 Activating the New WB Configuration

To activate the new configuration, two method can be used:

- Activate via a reboot

```
# reboot
```

- Activate without a reboot

```
# ifrc restart
```

3 Set Up – USB Ethernet on a Host PC

From a PC running Ubuntu 14.04, you must configure the usb-ethernet host device settings.

Note: The IP address of the USB Ethernet connection is 192.168.3.1. When configuring the host interface the IP address assigned to the host must be on the same subnet.

Once the USB cable is plugged in, the Linux OS should recognize and load the USB Ethernet driver. Ensure that the USB Ethernet interface is being used for bridging the Wi-Fi on the WB50 to the host by removing any Ethernet cables and disabling any Wi-Fi interfaces on your host PC. To do this, left-click the network icon in the upper right corner on the task bar and un-check **Enable Wi-Fi**.

To set the IP address on the USB Ethernet device to be compatible with the WB50, follow these steps:

1. Left-click the Network Manager icon on the task bar and select **Edit connections**.
2. From the Ethernet list, select the **Wired connection** interface.
3. Click **Edit**..
4. In the Editing Wired connection window, select the IPv4 Settings tab.
5. From the Method list, select **Manual**.
6. Click **Add**.
7. Change the address settings to the following:

   Address: 192.168.3.2

   Netmask: 255.255.255.0

   Gateway: 192.168.3.1

   DNS servers: 8.8.8.8

8. Click **Save** and **Close**.

If your WB was set up with a profile and connected to a local AP, you should now be connected to the network via the Wi-Fi. Verify by opening your browser and navigating to a resource on the network or, if the network has access to the Internet, navigate to our website: http://www.lairdtech.com/

4 Setting Up a PPP Link Over RS232

The WB50NBT may be configured as a Wi-Fi peripheral using a PPP (point-to-point) link over RS232. The USB to RS232 cable is required for this example. On the WB, use the sdc_cli to create and activate a wireless profile. Once a Wi-Fi connection has been established, attach the DB9 connector of the cable to the BB40 (use the port on the BB40 labeled **UART0** which is **/dev/ttyS2**) and the USB connector to the host.

See the examples for use cases 3 and 4; substitute ***ppp0*** to configure the WB for PPP over RS232.

Next, you must set up your host machine.

Enter the following into the Linux terminal:

```
# /usr/sbin/pppd 192.168.0.2:192.168.0.1 /dev/ttyUSB1 230400
```

Note: Your USB interface may be different.

4.1.1 Forwarding UART Data Over a TCP Socket with Socat

In many instances, you may want to directly connect a device or file to a socket. This may be done with a tool called Socat. Socat allows the creation of bidirectional byte streams that may be connected to sockets and/or devices within Linux.

Below is a simple application (using Socat) to connect a TCP socket to an external UART. The following example command demonstrates how to establish the Socat link between the device so that Linux forwards all incoming UART data (on ttyS1) over the socket connection and vice versa.

To use Socat to connect TCP port 1234 to the UART running on **/dev/ttyS1**, use the following:

```
# socat tcp-l:1234,reuseaddr,fork file:/dev/ttyS1
```

5 Making Modifications to the WB50 Software

The WB50 system can be built from source in order to add or modify the system as desired.

Details on obtaining the source can be found at in the Laird WB source *How To* (https://github.com/LairdCP/wb-manifests/blob/master/README.md) in the sections covering *Preparation* and *Getting the WB Project*.

5.1 Building the WB from Source

The WB50NBT utilizes Buildroot to build the entire system. Laird's WB50NBT source release package provides everything necessary to build and customize a WB image for your application and hardware system. WB source releases are distributed on GitHub.

Please go to GitHub for both the source release and the instructions for working with it:
https://github.com/LairdCP/wb-manifests

Requirements:

- Linux system capable of building Buildroot
- Internet access

Recommended:

- Ubuntu 14.04 LTS

Details on the system requirements for using Buildroot can be obtained from the Buildroot User Manual.
(http://buildroot.uclibc.org/downloads/manual/manual.html#requirement)

Complete information on Buildroot is also available with the Buildroot User Manual.
(http://buildroot.uclibc.org/downloads/manual/manual.html)

6 Updating the WB50NBT Software

There are various methods of updating the software on the WB. Once you've completed a build with the buildroot framework or have downloaded the binary image files, there are three options for programming them onto the WB50NBT:

- Linux: Use the fw_update utility on the WB.
- UBoot: Flash the images from the u-boot environment shell.
- SAM-BA: Use the external sam-ba utility if there is no image present on the WB or the system has become corrupted.

6.1 Flash Programming Using fw_update

The fw_update program is run via the Linux shell in either of two ways:

- **Automatic** – During bootup, if using a DHCP server that also sends a *Bootfile-Name* option along with the IP address. The option value is set to the URL of the images server.
- **Manual** – The fw_update program can be invoked manually over the WB's Linux command line interface (CLI).

In all cases, fw_update requires an fw.txt file to describe the image files available via locally-attached media (such as a USB flash drive or SD card); or via remote network server (anonymous https, http, and ftp are supported). The program has built-in help and examples which can be viewed with:

```
# fw_update -h
# fw_update --usage
# fw_update --usage fw.txt
```

The fw_update program can be invoked from the CLI. You can obtain access to the CLI either through the console serial port or over a network connection, via Ethernet or a USB Ethernet gadget, with the use of SSH. By default, an SSH server is enabled on the WB50NBT. The username is *root* and the password is *summit*. It's important that the password is changed to something different in the filesystem of the end product.

For example: If updating from a USB flash drive:

Load the image components and the fw.txt file onto a USB or mmc/SD drive. Insert the drive into the appropriate port on the BB50 breakout board. The media drive is auto-mounted to the directory /media/usb0.

```
# fw_update /media/usb0/fw.txt
```

The path of the image files on the USB drive can be specified on the fw_update command line; a path with spaces is not supported. For example:

```
# fw_update /media/usb0/directory_path/fw.txt
```

... when the files are stored on the USB drive in /directory_path/.

6.1.1 fw_update via Network

fw_update can also be used over a wireless connection if the image files are placed on a web or ftp server. For example:

```
# fw_update http://<url of server>/<path>/fw.txt
```

6.2 Remote Update Requirements

To use the Remote Update capability of the WB50NBT via Wi-Fi, three pieces of hardware must be in place:

- Wi-Fi access point which supplies a network connection
- Linux server running DHCP and either HTTP or FTP
- Laird WB50NBT module

Wi-Fi Access Point

To connect to a Wi-Fi access point, the WB50NBT module must be configured using the CLI to create the corresponding security profile. The network credentials used to create the profile should match those of the AP to which the WB5NBT will be attached.

Linux Server

The server must offer DHCP and either HTTP or FTP file services.

The DHCP server must be configured to send an additional parameter called a DHCP Option. The DHCP Option specifies a Bootfile-Name (DHCP option #67). The Bootfile-Name is a URL of a text file that further describes the images to be updated (the Bootfile). More information on the bootfile can be found later in this document.

The update program uses the *wget* utility to fetch the remote updates from the file server over the network. It supports both HTTP and FTP protocols.

Laird WB50NBT

The WB50NBT must be connected to a power source and have a Wi-Fi antenna connected to it.

6.3 Bootfile Description

The bootfile (called fw.txt) is a list of all images that are to be updated on the WB50NBT. It resides on the file server and is pointed to by the Bootfile-Name option in the DHCP server's configuration.

The following is the format of the bootfile:

```
06c73bb2c38f6bd0ecde4b2308595adc  at91bs.bin  11444
c55870dce9de3347950987b135a9fcd3  u-boot.bin  359032
5f7f5ebb4717250cdf565e20049debbe  kernel.bin  2091686
479d3828243c3712efd9a14ecfefb96e  rootfs.bin  23592960
```

You do not need to list all four images in this file. If only a subset of these images is required to be upgraded, then only list those particular image files. Each line has three elements:

- MD5SUM of image file
- Name of the image file
- Length of the image file in bytes

For complete fw.txt details and usage, enter the following:

```
fw_update --usage fx.txt
```

6.3.1 Remote Update Methodology

The remote update can be performed automatically using DHCP or manually, using the fw_update program on the WB50NBT. Both methods are described in the following sections.

6.3.2 Automatic Remote Update

This section describes the default behavior of Laird's WB50NBT firmware.

When the WB50NBT is powered on, it connects to the AP for which it has been provided a valid network security profile. After associating with an AP, the WB50NBT sends out a DHCP request. The DHCP server must respond with an IP address and a Bootfile-Name. The WB50NBT detects if it has received this optional DHCP information. If so, it proceeds with the fw_update program using the information in the bootfile to update its flash memory. If the MD5SUM of the components listed in the bootfile match what is already on the device, then nothing is updated and the WB50NBT continues to run normally.

After all the updates are complete, the bootloader is made aware of the new kernel and/or filesystem and loads these new images on the next reboot. The WB50NBT reboots into the newly-updated system automatically after the update process is complete.

6.3.3 Manual Update Remote

When manually starting the fw_update program, the URL for the *fw.txt* file must be supplied as an argument to the fw_update program. For example:

```
# fw_update http://192.168.1.10/wb50n/fw.txt
```

Note: WB50NBT requires a valid IP address before it can fetch the image files from the network. If the WB50NBT is configured as a bridge (which may not have an IP address), then it must be configured with a valid IP address on its network interface before running the fw_update program. This can be done by stopping the bridging (/etc/network/bridge.sh stop) and then using udhcpc -i wlan0 or the ifrc utility to configure the IP address settings of the wireless interface.

After the update program has completed its process, the bootloader becomes aware of the new kernel and/or filesystem. The new images are loaded on the next reboot. Issue the reboot command to reboot the WB50NBT. When invoked manually, the fw_update program does not automatically reboot the WB50NBT after it has completed updating the device.

6.3.4 Limitations

These methods of updating are destructive to existing data on the device because existing data in the flash memory of the WB50NBT is overwritten by the new image.

Boot failures are not automatically handled by the software.

6.4 Flash Programming using U-boot shell

If the WB50NBT module already has both the bootstrap and U-Boot programmed into it, then U-Boot may be used to load binary images. To use U-Boot to program flash memory, an Ethernet connection must be available and the binary images must be available over the network from a TFTP server.

U-Boot supports the Ethernet interface (not the wireless nor USB interfaces) for downloading images. For this reason, it is necessary to have the WB50NBT plugged into the breakout board so that the RJ45 Ethernet connection is available. This method of programming also requires that a TFTP server exists on the local network and that it has the binary images that are required for the programming procedure.

To get into the command prompt of U-Boot, connect a serial line from the Debug UART to a PC running a terminal program such as Tera Term. Tera Term may be obtained from: http://ttssh2.sourceforge.jp/

The serial interface should be configured for 115200 baud, no parity, 8 data bits, 1 stop bit (N81), and no handshaking. When the WB50NBT is powered up, the serial terminal displays some bootloader messages. At this point, press **Enter** several times; the following command prompt appears:

```
U-Boot>
```

Note: If U-Boot does not see any key press, then it automatically boots the operating system after a one second delay.

At this point, U-Boot may be used to update itself, the bootstrap loader, the Linux kernel, or the file system. Any one of these items may be updated and they do not all necessarily need to be updated at the same time. However, because of compatibility, it's good to ensure that the Linux kernel and the file system are intended to work together (from the same release version).

The process of updating a flash image using U-Boot is comprised of downloading the image to be programmed into local SDRAM of the WB50NBT, erasing the flash at the target memory location, and then copying the contents of SDRAM to flash memory.

Preparing for using network and TFTP:

```
dhcp

set serverip <ip-address-of-tftp-server>
```

To update the bootstrap loader, follow these commands:

```
tftp at91bs.bin

nand erase 0x00000000 0x00020000
nand write 0x22000000 0x0 ${filesize}
```

To update U-Boot, follow these commands:

```
tftp u-boot.bin
nand erase 0x20000 0x80000
nand write 0x22000000 0x20000 ${filesize}
```

To update the Linux kernel, follow these commands:

```
tftp kernel.bin
nand erase 0x000e0000 0x00500000
nand write 0x22000000 0x000e0000 ${filesize}
```

To update the file system, follow these commands:

```
tftp 0x20000000 rootfs.ubi
nand erase 0x00ae0000 0x03000000
nand write.trimffs 0x20000000 0x00ae0000 ${filesize}
```

To reset the system from the U-Boot prompt, type **reset** and press **Enter**.

6.5 Troubleshooting U-Boot

If it is not known whether or not a WB50NBT module has a boot loader programmed into it, the presence of a boot loader can be determined by examining the serial data from the Debug UART at power-on. This serial interface runs at 115200 baud, no parity, 8 data bits, 1 stop bit, and no hardware handshaking. The boot loader displays text that is similar to the following:

```
AT91Bootstrap 3.7.1-laird07 (Thu Mar 17 00:08:09 EDT 2016)

NAND: ONFI flash detected

NAND: Manufacturer ID: 0x2c Chip ID: 0x31

NAND: Initialize PMECC params, cap: 0x8, sector: 0x200

NAND: Image: Copy 0x80000 bytes from 0x20000 to 0x23f00000

NAND: Done to load image


U-Boot 2014.07-laird04 (Mar 17 2016 - 00:08:13), Build: jenkins-
wb50n-laird_fip1


CPU: SAMA5D31
Crystal frequency:       12 MHz
CPU clock        :      528 MHz
Master clock     :      132 MHz
DRAM:  64 MiB
NAND:  128 MiB
In:    serial
Out:   serial
Err:   serial
Net:   macb0, usb_ether
Hit any key to stop autoboot:  0
```

If only *RomBOOT* displays, then the boot loader is not installed (flash is empty or the bootstrap loader is corrupt). If nothing displays, check the serial connection and verify that the WB50NBT is properly powered.

6.6 Flash Programming Using the Atmel SAM-BA Utility

It is possible to program the bootloader images only into the WB50NBT using SAM-BA. Details on this procedure are in the application note *Flash Programming using the Atmel SAM-BA Utility*. This application note is available from the Documentation tab of the Laird WB50NBT product page.

7 Breakout Board Schematic and BOM

The breakout board schematic and BOM (Bill of Materials) are available from the Laird WB50NBT product page.

8 Developing and Integrating Using the Development Kit

There are many helpful tools and hints to developing software and integrating hardware with the WB50N.

8.1 Software Tools and Techniques

8.1.1 devmem

The program *devmem* can be used to read or set any arbitrary physical address on the system. It is particularly useful for inspecting the state of the Atmel processor's special function registers when trying to debug drivers, device trees, or other kernel code.

```
# devmem
BusyBox v1.21.1 (2013-07-29 17:17:51 PDT) multi-call binary.

Usage: devmem ADDRESS [WIDTH [VALUE]]

Read/write from physical address

ADDRESS     Address to act upon
WIDTH Width (8/16/...)
VALUE Data to be written

# devmem 0xFFFFF818 32
0x00004100
```

The above example shows the PIO_OSR (Output Status Register) for PIOC. Note that pc8 and pc14 are configured as GPIO outputs.

8.1.2 sysfs

Linux can present data about the system and drivers by using *sysfs* (system file-system). The entire tree can be found mounted at **/sys** and, through various virtual files, you can read information about the kernel, drivers, and hardware of the system. By writing to various files, you can cause the drivers or kernel to take various actions. See the GPIOs section below for examples. Detailed information can be found in the following document:
https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

8.1.3 debugfs

Debugfs (debug file-system) is a useful virtual file system that presents debug information and allows you to affect various system parameters. It is found at **/sys/kernel/debug**.

8.2 Finding Version Information

The kernel version can be found via the **uname** command:

```
# uname -a
```

```
Linux summit 4.1.13-laird12 #1 PREEMPT Thu Mar 24 23:37:58 EDT 2016 armv7l
GNU/Linux
```

The rootfs Laird release can be found in **/etc/laird-release**:

```
# cat /etc/laird-release

Laird Linux wb50n-laird-3.5.2.16
```

9 Hardware Use Notes

9.1 GPIOs

Name	Linux ID	LED	Header Pin	Port	Comments
LED0	12	D18	1	pa12	
LED1	24	D17	2	pa24	
LED2	26	D16	3	pa26	
STAT0	22	D15	4	pa22	
STAT1	28	D14	5	pa28	
GPIO-2	10	D11	8	pa10	Configured as input with pull-up. May be used to wake from standby. Handled by input-event system.
IRQ	159	N/A	X	pe31	Configured as input. BB40 button, labeled WAKEUP, is broken. Handled by input-event system.

9.1.1 Use in Linux

Linux uses a standard GPIO library and all the GPIOs that are not claimed by a device may be accessed via the **sysfs** interface. To use a GPIO, you must first export it. It is exported **to /sys/class/gpio/name/**.

To work with a pin as an output, follow these steps:

1. Export the pin.
2. Set its direction as an output.
3. Set it to the desired value.

To work with the pin as an input, follow these steps:

1. Export the pin.
2. Set its direction as an input.

Note: These are set as inputs by default so this step is likely unnecessary.

3. Read the value.

For example, to change LED0 to an output and set it high:

```
# cd /sys/class/gpio
# echo 12 > export
# echo 'out' > pioD18/direction
# echo '1' > pioD18/value
# cat pioD18/value
```

```
    1
```

Note: GPIOs are set as inputs by default. The *value* reflects the input value on the pin.

9.1.1.1 Export

```
# echo 12 > export
```

9.1.1.2 Set input

```
# echo 'in' > pioD18/direction
```

9.1.1.3 Read input value

This must be first set as input.

```
# cat pioD18/value
0
```

9.1.1.4 Set output

```
# echo 'out' > pioD18/direction
```

9.1.1.5 Set pin value

This must be first set as output.

```
# echo '1' > pioD18/value
# echo '0' > pioD18/value
```

9.1.1.6 Set IRQ as GPIO interrupt

```
# echo 159 > /sys/class/gpio/export
# echo 'in' > /sys/class/gpio/pioE31/direction
# echo 'both' > /sys/class/gpio/pioE31/edge
# cat /proc/interrupts
```

9.2 USB

The WB50NBT has one device-mode USB2.0 port and two host-mode USB2.0 ports. By default, these ports are enabled and working.

The device mode port, DDM/DDP pins on the WB50 connector, are routed to the USB Type-B connector on the BB. This port is set up as a CDC Ethernet device (Ethernet over USB).

The host mode ports, HDMA/HDPA and HDMB/HDPB pins on the WB50 connector, are routed to the two USB Type-A connectors on the BB. While the BB can provide some power over these connectors, it cannot provide full USB power, so self-powered devices are recommended for devices that pull significant current.

The USB host-mode ports are set up to handle storage devices (USB flash drives) by default. A CDC-ACM driver is also included in our default images so CDC-ACM compliant serial devices may also be used. Laird also supports SMSC95xx USB-connected Ethernet controllers like the LAN9500A from Microchip and provides the driver on-board.

```
# modprobe smsc95xx

# modprobe cdc-acm
```

10 WB Security

This section addresses current security matters related to the use of the Laird WB. The WB is designed to aid a development engineer in the development of an end system. For ease of development it is not a hardened platform and the flash memory file system is unprotected. This leads to the following conditions which a development engineer should be aware of:

- A variety of system utilities are present that can manipulate flash memory.
- Networking is unrestricted; there are no defined firewall (traffic use case) rules.

Software updates are performed by *root* using *fw_update* and *fw_select*.

- fw.txt automatically transfers /root/.ssh/ which may be undesirable due to the potential transferal of out-of-date or compromised keys.
- Software bundled with the WB development platform is not cryptographically-signed.
- Because fw_update does verify and record the md5sum of each image, you can sign your software and fw_update could then check the signing via key.

10.1 Solutions to Enhance Security

The ODM should consider the following actions to enhance security on their end system:

- Change the root login to something stronger – The default root and user password is summit. You can change it with the passwd utility.
- If possible, use a non-root account for basic operations or status checking –
  - There are currently three user accounts (summit, ftp, and default). If you're not using these accounts, remove them.
  - The adduser and deluser utilities can be used to manage accounts.
- Enable login timeouts.
- Disable any unnecessary services, such as *httpd*, and *ftpd*.
  - Some services are handled via inetd (/etc/inetd.conf is the default configuration file for the inetd)
  - Other services may be managed using chmod (change mode) on specific init-scripts.
- Disable unused network interfaces – In /etc/network/interfaces, set #auto <iface> to prevent auto starting.
- Serial console
  - Disable debug console login access to deter physical attacks via the debug console port
    - Disable unnecessary serial port access by removing their entries in **/etc/inittab**.
    - The **/etc/inittab** file contains activation for the *getty* program and options.
  - Set **uboot bootdelay** to zero (0) to stop uboot serial console access.
- Install proper security certificates in **/etc/ssl/...**

  Note: There is no RTC available (no hardware clock).

- Ensure only *root* can access certain apps (such as the SMU) or remove them completely.

- Remove *athtestcmd* and manufacturing test firmware for Wi-fi (or ensure it is not accessible to the end user).

- Modify the default username/password for lighttpd.
- Modify the default root password.
- When in AP mode, disable SSID broadcasting.
  - Options are in the **/etc/hostapd/hostapd.conf** file.
- In regards to SSH security, do the following:
  - Disable root access via SSH.
  - Only allow key-based authentication over SSH (disable password authentication).
  - The SSH options are in the /etc/ssh/sshd_config file.
  - Enable SSH timouts.
- Configure the firewall.
- If you're using SSH, serial console, or lighttpd, set a timed lockout after a failed number of incorrect login attempts.
- DCAS authentication is via public key by default. Although DCAS can have username/password authentication configured in addition to public key, we do not recommend this setup unless you have a temporary provisioning algorithm in place that turns off the authentication method when it's completed.

10.2 Best Practices for Improving Wireless Network Security

The following are best practices for improving the security of the wireless network:

- For PEAP/TLS, always specify a CA certificate to match against, to prevent connecting to a hacker's AP.
- When selecting a CA certificate, rather than using a certificate from a global certification authority, use a middle certificate. This prevents your end device from just connecting to any AP that can source back to that global CA certificate.
- Matching the CA certificate is particularly important for PEAP-GTC where the password is only protected by the encrypted tunnel.
- Password and PSK should use strong password selections (long and from multiple sets of characters).
- Use the appropriate (highest) encryption and authentication – AES-CCMP.

10.3 WB inetd and WB init-scripts

10.3.1 WB inetd

Many of the standard services are managed by inetd (an internet services daemon). This utility listens for connections, as enabled in the **/etc/inetd.conf** file, and starts programs to provide the requested service.

```
# /etc/inetd.conf
# Listens to multiple ports and invokes a requested service.
# Mainly useful for state-controlled connections and reducing total load.
# A service may be disabled with a prepended '#' char.

 ssh     stream  tcp      nowait  root     /usr/sbin/sshd   sshd -i
 tftp     dgram  udp        wait  root     /usr/sbin/tftpd  tftpd -l -c /root
 ftp     stream  tcp      nowait  root     /usr/sbin/proftpd  proftpd -d1
```

Remove any undesired services or prefix them with a **#**.

```
#ssh    stream  tcp       nowait  root      /usr/sbin/sshd  sshd -i
```

You must restart inetd or reboot the WB for changes to take effect.

Note: The *inetd.conf* file may be cached as **/tmp/inetd.conf**.

Use the following to restart inetd:

```
/etc/init.d/S??inetd restart
```

10.3.2 WB init-scripts

Most of the init-scripts used for starting or stopping services and devices reside in **/etc/init.d/** and **/etc/init.d/opt/**. They are run in lexical order by *rcS start* during bootup and in reverse-lexical order by *rcS stop* during shutdown.

The *rcS only* runs scripts that have the executable bit set and are accessible from the top directory.

These scripts at the top-level are essential, while those in **opt/** are more conditional as to when or if they might be run. The optional scripts may have a symlink to them from the top-level.

Note: Some optional scripts are conditionally called by the network *init-script* or by inetd, if enabled.

The following are special init-scripts:

- rcS (run-config-superscript) – Run by init and calls all other init-scripts.
- S40network (network-init-script) – Auto-starts enabled interfaces in the /etc/network/interfaces file. This script then calls upon a network-configuration tool to apply settings.
- S50inetd (internet services daemon init-scipt) – For stateful services enabled in /etc/inetd.con. This script calls on optional service init-scripts to perform pre-checking.

The following network-specific (located in **/etc/network/**) are multi-mode and serve as system commands (and are used by ifrc):

- wireless.sh – Wi-Fi interface driver and firmware configuration. This script also launches the supplicant and optionally supports FIPS mode.
- bridge.sh – Bridging interface setup.

Example init-script:

```
#!/bin/sh
# name - and a brief description
# details, author, usage...

case $1 in

  stop)
    echo "Starting some task"
    ##
    ## steps to perform during shutdown only
```

```
     ##
     ;;

   start)
     echo "Stopping some task"
     ##
     ## steps to perform during bootup only
     ##
     ;;
 esac
```

In the above example, the first line is asking for the system shell to interpret the script. Since busybox ash is assumed, you may want to be more explicit (use **/bin/ash** or **/bin/hush**).

Next are *parse* and *handle* commands as the first argument.

The first argument is tested for match. When rcS is calling the script, it uses either *start* or *stop*. Additional commands can also be handled if the script needs to provide additional tasks such as *restart* or *status*.

```
   restart)
     echo "Restarting some task"
     $0 stop
     $0 start
     ;;
```

In this example, the script calls itself for stop and then start, providing a restart action.

Note: The init-scripts must always return within a reasonably short period; they are expected to do some series of steps and then be finished so that the rcS can move on to the next script. Otherwise the system appears to hang.

11 References

http://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xml

Source-Forge (Tera Term) website: http://en.sourceforge.jp/projects/ttssh2/releases/

Buildroot: http://buildroot.uclibc.org/downloads/manual/manual.html#requirement

Complete information on Buildroot is located at:
http://buildroot.uclibc.org/downloads/manual/manual.html

Detailed Kernel sysfs Information: https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

12 More Information and Support

Support materials, documentation, schematics, and software for the WB50NBT are available on Laird's support site. Additionally, you may contact Laird's support team from the support site as well at http://ews-support.lairdtech.com.