NetBurner
Networking in 1 Day!

NBWIFIUG-100CR
Wireless Network Interface Controller
IEEE 802.11b/g
Programmers Guide and Reference Manual

Revision 1.0
November 6, 2009

Table of Contents

1. Introduction

The NBWIFIUG-100CR is an IEEE 802.11b/g wireless network interface controller (NIC) that is based on the Realtek RTL8711 c hip set. It has the following features:

1. Compliant to IEEE 802.11 b/g standards.
2. Data rates of 1, 2, 5.5, 6, 9, 11, 12, 18, 24, 36, 48, and 54Mbps.
3. DSSS with DBPSK and DQPSK, CCK modulations and demodulations supported with long and short preamble.
4. Hardware-based IEEE 802.11 encryption/decryption engine for 64-bit/128-bit WEP.
5. Communicates with the NetBurner module via the QSPI bus.

This reference guide also covers the software library API used for programming with the NBWIFIUG-100CR.

2. Product Compatibility

2.1 Compatible NetBurner Modules

The NBWIFIUG-100CR is compatible with the modules listed below:

- MOD5234
- MOD5270
- MOD5272
- MOD5282

2.2 Default Interface Signals

The default hardware and software configuration uses the following NetBurner processor module signals:

- QSPI MOSI, QSPI MISO, QSPI CLK, QSPI CS1
- IRQ 3

2.3 Custom Interface Signals

You may use a different QSPI chip select and/or IRQ input. To use a different QSPI chip select you must use the long for of the AddWifiInterface() function that specifies the chip select and IRQ signal used. To use a different IRQ signal you must ensure the proper IRQ signal is routed to the Wifi module.

3. Additional Documentation

This document covers the wireless product only. If you need assistance in using the NetBurner development tools, TCP/IP stack or Real-Time Operating system, please refer to the following documents located in c:\nburn\docs:

- NNDK Programmer's Guide: a textbook style document on programming NetBurner devices.
- NBEclipse Getting Started Guide: a tutorial on using the NBEclipse Integrated Development Environment.
- NetBurner Runtime Libraries: API function reference documents for the TCP/IP Stack and RTOS.

4. Installation

4.1 For Development

The NNDK-NBWIFIUG-KIT development kit includes an adapter board that enables it to be installed on the J2 header of the MOD-DEV-100CR or MOD-DEV-70CR development boards, which are included with the NetBurner Modxxxx development kits.

If you have a MOD-DEV-70CR, you will need to install a 50-pin 0.1” dual row header in location J2_C.

The Wifi software libraries are automatically installed with your development kit.

4.2 For Production

The NBWIFIUG-100CR has a single row 0.1” 10-pin header with the QSPI, IRQ, Power and Ground signals.

5. Hardware

The NBWIFIUG-100CR has a 10 pin, single row connector and a board mounted reverse subminiature version A (SMA) plug for the antenna. All electrical signals have a maximum voltage of 3.3VDC.

5.1 NBWIFIUG-100CR J2 Connector Pinout

Pin No	Silkscreen	SPI Signal	Module Signal	SPI Function
1	GND	GND	GND	Ground
2	SD02	Unused		
3	SD03	SS	QSPI_[0:3]	Slave Select (Chip Select)
4	SDCMD	MOSI	QSPI_DOUT	Master Out/Slave In
5	GND	GND	GND	Ground
6	VDD	VCC	VCC	Supply Voltage 3.3V
7	SDCLK	SC(L)K	QSPI_CLK	Clock
8	GND	GND	GND	Ground
9	SD00	MISO	QSPI_DIN	Master In/Slave Out
10	SD01	IRQ	IRQ[3:7]	Interrupt request input

5.2 Development Kit Adapter Board JP1 Pinout

The JP1 header on the development board enables you to select a the desired QSPI core module chip select. However, if you change from the default QSPI CS1, you will need to use the long form of the AddWifiInterface() function to specify the new chip select number. The Wifi infrastructure example in your NNDK tools installation illustrates the use of this function call.

Please refer to the data sheet for your specific NetBurner Ethernet module for the signal descriptions on the J2 connector. All compatible MODxxxx products have QSPI CS1 on J2-40.

Shorting Jumper Location	Signal Pin on Ethernet Module J2 Connector
1-2	J2-26 (QSPI CS3)
3-4	J2-30 (QSPI CS0)
5-6	J2-32 (GPIO)
7-8	J2-35 (QSPI CS2)
9-10	J2-40 (default QSPI CS1 on all modules)

6. Wifi Interface Example

The following example is designed to add a Wifi interface to a NetBurner module with an existing Ethernet interface. The example code fragments were extracted from main.cpp of the nburn\examples\Wifi_infrastructure example installed with the NNDK tools. The iprintf( ) status messages are sent out the stdio port, which is the debug serial port by default. The Wifi SSID is configured by the IPSetup utility.

The example code is similar to other Ethernet examples. The major different is that to add Wifi connectivity we add the function: AddWiFiInterfacewName( ). The rest of the Wifi related code get a DHCP address and display status information on the SSID and WEP key (if used).

The example has the following flow:

- Initialize the TCP/IP stack, enable code updates and start the web server.
- Configure the Ethernet interface and display settings.
- Specify the Wifi callback progress function. This is not required to run Wifi, but it is used in the example program to display the activity of the Wifi interface initialization.
- Add the Wifi interface.
- Configure the Wifi interface and display settings.
- Continually loop wait for any input from the debug serial port for status.
- The primary interface is the web server, which provides status information on both interfaces.

```
/*-----------------------------------------------------------------
 * UserMain
 * ---------------------------------------------------------------*/
void UserMain( void *pd )
{
   WifiInitComplete = FALSE;  // flag used by web pages for status
   InitializeStack();               // Init TCP/IP Stack
   OSChangePrio( MAIN_PRIO ); // Set UserMain task priority to default
   EnableAutoUpdate();              // Enable network updates
   StartHTTP();                           // Start web server
   EnableTaskMonitor();             // Enable remote task monitoring

   // Will get a DHCP address if static IP is not specified
   ProcessEthernetDhcp();

   // Specify the callback function for wireless initialization progress.
   SetupWifiProgressCallback( displayProgress );

   /* We will now add the wireless network interface. This is
    * similar to the Ethernet interface, and once complete, we
    * will have both an Ethernet and Wireless interface available.
    * The first parameter in AddWifiInterfacewName() is the SSID
    * and the second is the WEP key. A NULL value means they use
    * the systems setting value, which can be configured by
    * IPSetup.
    */
   char ConfigSSID[40];
   GetWiFiConfigurationSSID( ConfigSSID, 40 );
   char ConfigWepKey[40];
   GetWiFiConfigurationWEPKey( ConfigWepKey, 40 );
   iprintf("Attempting to connect to network: [%s], WEP: [%s]\r\n",
            ConfigSSID, ConfigWepKey );

   // NULL parameters use the IPSetup values for SSID & WEP
   WifiInterface = AddWiFiInterfacewName( NULL, NULL );



   if ( WifiInterface <= 0 ) // Valid interface numbers are > 0
   {
      iprintf( "Wifi: NIC did not initialize, check module and
               connections\r\n" );
   }
   else if ( WiFiConnected() == TRUE )  // connected to a Wifi network
   {
      // Will get a DHCP address if static IP is not specified
      ProcessWifiDhcp();

      // Flag used by web page code to display Wifi settings
      WifiInitComplete = TRUE;

      // Returns SSID if connected, otherwise NULL
      char buf[40];
      if ( GetCurrentSSID( buf, 40 ) > 0 )
         iprintf("Connected to network with SSID: [%s]\r\n", buf );
   }
   else
   {
      iprintf( "Not connected\r\n" );
   }

   DisplayMenu(); // Display serial port menu options

   // Process serial commands
   while ( 1 )
   {
      char buffer[255];
      buffer[0] = '\0';

      while ( buffer[0] == 0 )
         gets( buffer );

      ProcessCommand( buffer );
   }
}
```

7. Important Implementation Notes

You must call the InitializeStack() routine before calling any Wifi functions.

The software driver is designed and implemented by default to support the hardware and pinout described in the previous sections. Any hardware changes may require corresponding changes to software interface function call parameters, module libraries, and system library.

The AddWiFiInterface or AddWifiInterfacewName routines must be called before any other Wifi functions to initialize the Wifi hardware, load the firmware, start the driver, and add the interface to the I/O system. These functions return the interface number for the Wifi module, which can then be used for subsequent Wifi module functions.

The listen() routine listens for socket requests on all active interfaces simultaneously. Replies are sent the same interface from which they were received. So a web page request received on the Wifi interface, will be replied to on the Wifi Interface.

The default network interface is the Ethernet interface. To make an outgoing connection to the Wifi interface use the connectvia() instead of the connect() function.

To ping on the Wifi interface, use PingViaInterface().

When the Wifi interface is added and has an IP address you can update local network ARP tables by calling sendGratuitiousArp()

8. Wifi Library Functions

8.1 AddWiFiInterface

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
int AddWiFiInterface(BOOL adhoc = FALSE )
```

Description:

Initializes the Wifi hardware, loads firmware, initializes driver and attempts to establish the specified connection using the SSID and WEP key in the stored configuration parameters (usually by IPSetup). This is a simple function to use if you are using all the default settings and hardware signals.

Parameters:

Type	Name	Description
BOOL	adhoc	TRUE for adhoc network, FALSE for infrastructure connection

Returns:

Any value greater than 0 --- The interface number
 0 --- Request failed.
-1 --- Wifi module not found.
-2 --- Wifi interface already exists.
-3 --- Unable to create task for Wifi driver.
-4 --- Platform does not support this option.

8.2 AddWiFiInterfacewName

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
int AddWiFiInterfacewName( const char *SSID, const char *WEP_Key,
      BOOL adhoc = FALSE, int irq = 3,
      NetDeviceController controller = ControllerRealtekRtl8711,
      NetDeviceBusType busType = BusTypeSdioBusSpiMode,
      int select = 1 )
```

Description:

Similar to AddWifiInterface( ), but adds parameters for custom hardware or software configurations.

Parameters:

Type	Name	Description
const char*	SSID	Service Set Identity (SSID), NULL uses SSID stored in the configuration record.
const char*	WEP_Key	Wired Equivalency Privacy (WEP) key is the key shared with access point/peer; NULL uses the WEP key stored in the configuration record.
BOOL	adhoc	TRUE for adhoc network, FALSE for infrastructure connection
int	irq	Fixed level interrupt source most likely 1, 3, 5,or7.
NetDeviceController	controller	Controller defined in netDevice.h included in Wifi.h
NetDeviceBusType	busType	Bus used by the controller defined in netDevice.h.
Int	select	Controller selector used by bus type, usually bus chip select.

AddWiFiInterfacewName (continued)

Returns:

Any value greater than 0 --- The interface number
 0 --- Request failed.
-1 --- Wifi module not found.
-2 --- Wifi interface already exists.
-3 --- Unable to create task for Wifi driver.
-4 --- Platform does not support this option.

8.3 WiFiAttachTo

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
BOOL WiFiAttachTo( const char *SSID, const char *WEP_Key,
                   BOOL useDefault = FALSE )
```

Description:

Connect to the specified Wifi network access point.

Parameters:

Type	Name	Description
const char*	SSID	Service Set Identity (SSID), NULL will attempt to attach to the access point/peer with the strongest signal level.
const char*	WEP_KEY	Wired Equivalency Privacy (WEP) key is the key shared with access point/peer; NULL will only allow attachment to an access point/peer with the disabled WEP functionality.
BOOL	useDefault	TRUE use the SSID and WEP key in the stored configuration parameters, FALSE use SSID and WEP_Key.

Returns:

TRUE --- Success
FALSE --- Failure

8.4 WiFiScanForAP

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
ScanResult *WiFiScanForAP( char* sitePtr = NULL )
```

Description:

Scans for all or requested access points.

Parameters:

Type	Name	Description
char*	sitePtr	Service Set Identity (SSID) NULL will scan for all access points within range, else the requested SSID will be actively searched for.

Returns:

Pointer to list of struct ScanResult. End of list has a SSID with a null string (“”).

8.5 WiFiShowScanForAP

Header File:

`#include<Wifi.h> // Found in C:\Nburn\include`

Synopsis:

`void ShowScanForAP( void )`

Description:

Using the iprintf interface, lists the access points from the last call to WiFiScanForAP.

Parameters: None

Returns: Nothing

Output example.

```
Wireless Site Survey (3)
------------------------------------------------------------------------
SSID                               BSSID              Channel  Security
wirelessNb                         00:0F:66:2B:FF:54      6        WEP

     RSSI      Noise  Quality  Rates
   -34 dBm   N/A dBm   N/A %     1,2,5.5,11,6,9,12,18,24,36,48,54
------------------------------------------------------------------------
SSID                               BSSID              Channel  Security
aspen                              00:1F:E5:82:40:AB      6        WPA

     RSSI      Noise  Quality  Rates
   -78 dBm   N/A dBm   N/A %     1,2,5.5,11,6,9,12,18,24,36,48,54
------------------------------------------------------------------------
SSID                               BSSID              Channel  Security
cove                               00:1C:11:2B:A8:8D     11        WEP

     RSSI      Noise  Quality  Rates
   -78 dBm   N/A dBm   N/A %     1,2,5.5,11,6,9,12,18,24,36,48,54
------------------------------------------------------------------------
    *** End ***
```

8.6 WiFiConnected

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
BOOL WiFiConnected( void )
```

Description:

Return status of Wifi connection.

Parameters:

None

Returns:

TRUE – Connected.
FALSE – Disconnected.

8.7 GetCurrentSSID

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
int GetCurrentSSID( char *buffer, int maxlen )
```

Description:

Copies the SSID of the current connection to *buffer.

Parameters:

Type	Name	Description
char*	buffer	Service Set Identity (SSID) buffer.
int	maxlen	Maximum length of buffer in bytes, should be at least Wifi_SSID_SIZE_IN_BYTES

Returns:

0 --- Not connected
Any value greater than 0 --- The number of bytes in the SSID

8.8 GetWiFiConfigurationSSID

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

`int` GetWiFiConfigurationSSID**`( char *buffer, int maxlen )`**

Description:

Copy the SSID stored configuration record to *buffer. This values is usually specified by the user in IPSetup.

Parameters:

Type	Name	Description
char*	buffer	Service Set Identity (SSID) buffer.
int	maxlen	Maximum length of buffer in bytes, should be at least Wifi_SSID_SIZE_IN_BYTES

Returns:

0 --- None or invalid record.
Any value greater than 0 --- The number of bytes in the SSID

8.9 GetWiFiConfigurationWEPKey

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

`int` GetWiFiConfigurationWEPKey**`( char *buffer, int maxlen )`**

Description:

Copy the WEP key in the stored configuration record to *buffer. The format is hexadecimal encoded ASCII bytes.

Parameters:

Type	Name	Description
char*	buffer	Wired Equivalency Privacy (WEP) key.
int	maxlen	Maximum length of buffer in bytes, should be at least Wifi_WEP_KEY_SIZE_MAX

Returns:

0 --- None or invalid record.
Any value greater than 0 --- The number of bytes in the WEP key, usually 10 (WEP 64) or 26 (WEP 128).

8.10 SetupWiFiConfig

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
void SetupWiFiConfig( void )
```

Description:

Setup the Wifi configuration over the serial port. Prompting for permanent IP address, SSID and WEP key. Equivalent functionality to SetupDialog from system.h.

Parameters:

None

Returns:

None

8.11 WifiProgressFn

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
typedef void ( WifiProgressFn )( int activity, int progress, char* details )
typedef WifiProgressFn* WifiProgressFnPtr;
```

Description:

User progress callback function for events. Events include initialization, controller code downloading, scanning, connecting and disconnection by the access point. This function can be used to provide detailed status messages that can be useful in debugging, or simply as user informational messages.

Parameters:

Type	Name	Description
int	activity	Activity
int	progress	Progress of activity zero (0) to complete (100)
char*	details	Progress information, NULL for none

Returns:

None

Notes:

Activity	Progress	Description
Wifi_NO_ACTIVITY	0	None
Wifi_CHATTER	Detail string	Detailed status messages
Wifi_INITIALIZING	0 - 100	Setting up interface.
Wifi_DOWNLOADING	0 - 100	Downloading firmware and settings
Wifi_SURVEYING	0 - 100	Surveying wireless sites
Wifi_CONNECTING	0 - 100	Attempting connection to access point
Wifi_DISCONNECTED	100	Disconnect by the access point

8.12 SetupWifiProgressCallback

Header File:

```
#include<Wifi.h> // Found in C:\Nburn\include
```

Synopsis:

```
void SetupWifiProgressCallback( WifiProgressFnPtr wifiProgressFnPtr )
```

Description:

Installs the Wifi progress callback function for events.

Parameters:

Type	Name	Description
WifiProgressFnPtr	wifiProgressFnPtr	Progress routine.

Returns:

None