Preliminary

# WirelessUSB™ NL Radio Datasheet WUSB_NL V 2.00

Copyright © 2011-2012 Cypress Semiconductor Corporation. All Rights Reserved.

## Resource Usage

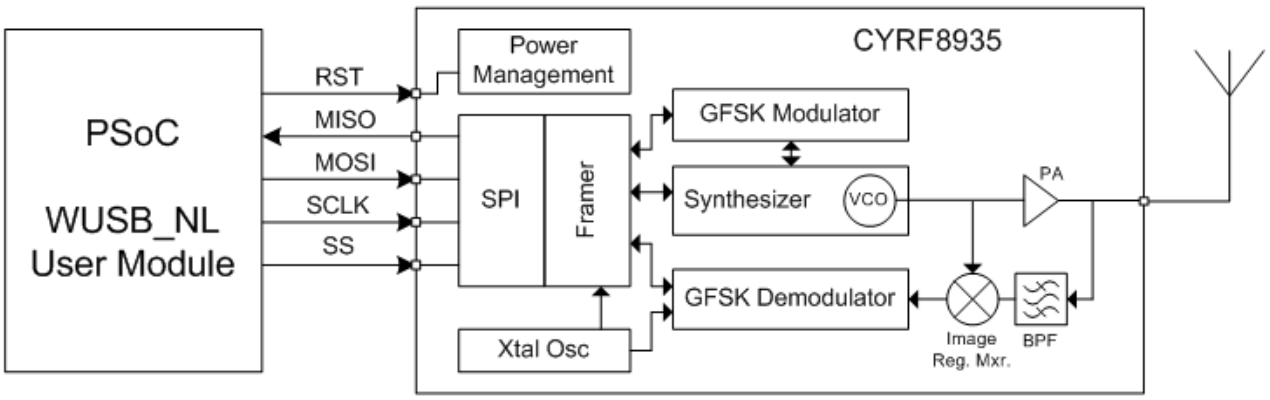
| Resources | PSoC® Blocks | | API Memory (Bytes) | | Pins (per External I/O) |
|---|---|---|---|---|---|
| | I2C/SPI | Timer | Flash | RAM | |
| CY7C60123-PVXC, CY7C60223-QXC, CY7C60223-SXC, CY7C63801-SXC, CY7C63803-SXC, CY7C63813-PXC, CY7C63813-SXC, CY7C63823-XC, CY7C63823-QXC, CY7C63823-SXC, CY7C63833-LTXC, CYRF89x35 | | | | | |
| Radio Status Register Polling | 1 | - | 1300 | 12 | 5 |
| CY7C60413-16LKXC, CY7C60445-32LQXC, CY7C60455-48LTXC, CY7C60456-48LTXC, CY7C64315-16LKXC, CY7C64316-16LKXC, CY7C64343-32LQXC, CY7C64345-32LQXC, CY7C64355-48LTXC, CY7C64356-48LTXC | | | | | |
| Radio Status Register Polling | 1 | - | 1500 | 14 | 5 |

## Features and Overview

The SPI-based WUSB_NL User Module is a firmware interface to the CYRF8935 WirelessUSB NL radio modem hardware.

- Typical transmit power: 0 dBm
- Packet framer with 64-byte first in first out (FIFO) data buffer
- Auto-retry-acknowledge protocol to simplify usage
- Cyclic Redundancy Check (CRC)
- Forward Error Correction (FEC)
- Supports up to 4 Mbps SPI bus data rate
- Digital readout of received signal strength indication (RSSI)

Figure 1. WUSB_NL User Module Block Diagram



## Functional Description

The WUSB_NL User Module can be used as a base for developing proprietary wireless protocols. The WUSB_NL User Module application programming interface (API) provides functions that can be called from both C and assembly to start the radio, send and receive data, change channels, change transmit power, and more.

The WUSB_NL User Module offers hardware SPI interface, initialization, configuration, and communication functions for interfacing with an NL radio.

## DC and AC Electrical Characteristics

- SPI bit rate ranges from 0.5 to 4 Mbps
- Refer to the WirelessUSB NL datasheet for EMC compliance requirements.
- For more information on SPI timing and packet data structure, refer to the WirelessUSB NL datasheet.

## Placement

The WUSB_NL User Module employs the SPI block configured as an SPI Master to communicate with the WUSB_NL radio transceiver.

The user module is restricted to only one instance in a project.

## Parameters and Resources

**PA Setting**

This parameter is used to set up the radio transmission power. The valid values are PA_0_DBM, PA_N3_DBM, PA_N8_DBM, and PA_N12_DBM. The default value is PA_0_DBM.

**Preamble Count**

This parameter sets the count of the preamble bytes. The allowed values are integer numbers between 1 and 4. The default value is 1.

**ACK enable**

This parameter enables or disables the automatic acknowledge radio feature. The valid values are Enable and Disable. The default value is Disable.

**ACK timeout**

This parameter sets the waiting timeout of the automatic acknowledge radio feature. The allowed values are integer numbers between 1 and 255. The default value is 255.

**FEC**

This parameter enables or disables the Forward Error Correction radio feature. The valid values are Enable and Disable. The default value is Disable.

**SS_Port**

This parameter is used to select the GPIO port for Slave Select output. The valid values are all available ports.

**SS_Pin**

This parameter is used to select the GPIO pin in the selected SS_Port for Slave Select output. The valid values are all available pins.

**RST_Port**

This parameter is used to select the GPIO port for Reset output. The valid values are all available ports.

**RST_Pin**

This parameter is used to select the GPIO pin in the selected RST_Port for Reset output. The valid values are all available pins.

**Clock Divider**

This parameter is used to set up the SPI Clock Divider.

The allowed values for CY7C601xx/2xx and CY7C638xx are integer numbers between 6 and 12. The default value is 6.

The allowed values for CY7C604xx and CY7C643xx are SysClk/2, SysClk/4, SysClk/8, SysClk/16, and SysClk/32. The default value is SysClk/8.

## Application Programming Interface

The Application Programming Interface (API) functions are provided as part of the user module to allow you to deal with the user module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns WUSB_NL_1 to the first instance of this user module in a given project. This name can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable, and constant symbol. In the following descriptions the instance name has been shortened to UMName for simplicity.

**Note** ** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must also ensure their code observes the policy. Though some user module API functions may leave A and X unchanged, there is no guarantee they may do so in the future.

### Data Types

The following user module-specific data types are used by some global variables and the arguments and return values of some API functions:

| User Module Data Type | C Compiler Compliant Data Type |
|---|---|
| WUSB_NL_STATE | unsigned char |
| WUSB_NL_XACT_CONFIG | unsigned int |
| WUSB_NL_TX_CONFIG | unsigned int |
| WUSB_NL_RX_STATUS | unsigned char |
| WUSB_NL_LENGTH | unsigned char |
| WUSB_NL_REG_ADDR | unsigned char |

| User Module Data Type | C Compiler Compliant Data Type |
|---|---|
| WUSB_NL_BUFFER_PTR | void pointer |
| WUSB_NL_CONST_PTR | const char pointer |
| WUSB_NL_RSSI | unsigned char |
| WUSB_NL_FRAME_CONFIG | unsigned char |

## Global Variables

The WUSB_NL User Module API functions use a global variable. You must not alter this variable manually, but you can monitor its value for debugging purposes. This global variable can be described as:

**WUSB_NL_State** - This is a global variable of the WUSB_NL_STATE type that contains the most recent return value from WUSB_NL_bGetReceiveState or WUSB_NL_bGetTransmitState. This value can be used instead of the return values. This is especially convenient when using WUSB_NL_Interrupt or WUSB_NL_Poll to perform radio state management. You can use WUSB_NL_State instead of the return values from WUSB_NL_bGetReceiveState or WUSB_NL_bGetTransmitState calls; for example to save storage to contain a return value or to simplify the code architecture.

The values for the WUSB_NL_State variable are defined in the following table:

| Value Name | Description |
|---|---|
| WUSB_NL_IDLE | When the radio is not performing the transmit or receive operations, WUSB_NL_State is WUSB_NL_IDLE. |
| WUSB_NL_RX | When the radio is performing a receive operation, the WUSB_NL_RX bit will be set. This bit will be set between calls to WUSB_NL_StartReceive and WUSB_NL_EndReceive. |
| WUSB_NL_TX | When the radio is performing a transmit operation, the WUSB_NL_TX bit will be set. This bit will be set between calls to WUSB_NL_StartTransmit and WUSB_NL_EndTransmit. |
| WUSB_NL_DATA | During a receive operation, this bit indicates that some data has been received. This bit will not be set during a transmit operation. |
| WUSB_NL_COMPLETE | This bit when set indicates that the current transmit or receive operation is complete. |
| WUSB_NL_ERROR | This bit when set indicates that the current transmit or receive operation has ended in an error. This bit will be set only when the WUSB_NL_COMPLETE bit is set. |

## API Functions

## WUSB_NL_Init

**Description:**

This function resets the radio and initializes its configuration to its default value. Using this API function, XactConfig and TxConfig radio settings can be configured by the application during radio bootup.

**C Prototype:**

```
void   WUSB_NL_Init(WUSB_NL_XACT_CONFIG wXactConfig, WUSB_NL_TX_CONFIG wTxConfig)
```

**Assembly:**

```
mov   A, >wTxConfig
push  A
mov   A, <wTxConfig
push  A
mov   A, >wXactConfig
push  A
mov   A, <wXactConfig
push  A
lcall WUSB_NL_Init
```

**Parameters:**

wXactConfig - Parameter to pass to WUSB_NL_SetXactConfig(). The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name for wXactConfig | Value |
|---|---|
| WUSB_NL_ACK_EN | 0x0800 |
| WUSB_NL_ACK_TO_4X | 0x6B |
| WUSB_NL_ACK_TO_8X | 0x9C |
| WUSB_NL_ACK_TO_12X | 0xCD |
| WUSB_NL_ACK_TO_15X | 0xFF |

wTxConfig - Parameter to pass to WUSB_NL_SetTxConfig(). The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name for wTxConfig | Value |
|---|---|
| WUSB_NL_PA_1_DBM | 0x1820 |
| WUSB_NL_PA_0_DBM | 0x1920 |
| WUSB_NL_PA_N3_DBM | 0x1A20 |
| WUSB_NL_PA_N8_DBM | 0x1C20 |
| WUSB_NL_PA_N12_DBM | 0x1E20 |

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetChannel

**Description:**

This function sets the radio channel to a specified frequency. The frequency has the value of (2402 + channel) MHz . This function takes an 8-bit argument that is passed to it by 'BYTE bChannel'. Radio-SetChannel is limited to a maximum of RADIO_MAX_CHAN, which is 78.

**C Prototype:**

```
void  WUSB_NL_SetChannel(BYTE bChannel)
```

**Assembly:**

```
mov   A, bChannel
lcall WUSB_NL_SetChannel
```

**Parameters:**

bChannel - 8-bit value that is passed to the function. The allowed value range is from 0 to 77 inclusive.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bGetChannel

**Description:**

This function gets the 8-bit value from the channel.

**C Prototype:**

```
BYTE  WUSB_NL_bGetChannel(void)
```

**Assembly:**

```
lcall WUSB_NL_bGetChannel
```

**Parameters:**

None

**Return Value:**

This function returns the 8-bit value of the channel number. The frequency must be interpreted as (2402 + channel) MHz).

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetTxConfig

**Description:**

This function sets the radio power amplifier gain. The transmitter configuration is also set when ther-adio is initialized by calling the WUSB_NL_Init. Both the functions write to the same register in the radio. This example sets a transmit configuration with a transmit power of -12 dBm:

```
WUSB_NL_SetTxConfig(PA_N12_DBM)
```

**C Prototype:**

```
void  WUSB_NL_SetTxConfig(WUSB_NL_TX_CONFIG wTxConfig)
```

**Assembly:**

```
mov   X, >wTxConfig
mov   A, <wTxConfig
lcall WUSB_NL_SetTxConfig
```

**Parameters:**

wTxConfig - Radio power amplifier gain setting. Possible values of wTxConfig are mentioned in the table in the WUSB_NL_Init API section.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_wGetTxConfig

**Description:**

This function returns the radio power amplifier gain.

**C Prototype:**

```
WUSB_NL_TX_CONFIG WUSB_NL_wGetTxConfig(void)
```

**Assembly:**

```
lcall WUSB_NL_wGetTxConfig
```

**Parameters:**

None

**Return Value:**

Radio power amplifier gain setting: Possible values are listed in the WUSB_NL_Init API.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetXactConfig

**Description:**

This function sets the transaction configuration composed of two parts: auto-ack enable and Rx ack timeout.

**C Prototype:**

```
void  WUSB_NL_SetXactConfig(WUSB_NL_XACT_CONFIG Config)
```

**Assembly:**

```
mov   X, >wXactConfig
mov   A, <wXactConfig
lcall WUSB_NL_SetXactConfig
```

**Parameters:**

Config - Transaction configuration is composed of two parts:

a) Enable auto-ack; possible value is ACK_EN

b) Rx ack timeout; possible values are:

ACK_TO_4X

ACK_TO_8X

ACK_TO_12X

ACK_TO_15X

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_wGetXactConfig

**Description:**

This function returns the transaction configuration composed of two parts, auto-ack enable and Rx ack timeout.

**C Prototype:**

```
WUSB_NL_XACT_CONFIG WUSB_NL_wGetXactConfig(void)
```

**Assembly:**

```
lcall WUSB_NL_wGetXactConfig
```

**Parameters:**

None

**Return Value:**

This function returns the transaction configuration composed of two parts:

a) Enable auto-ack; possible value is ACK_EN

b) Rx ack timeout; possible values are:

ACK-TO_4X

ACK_TO_8X

ACK_TO_12X

ACK_TO_15X

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_ForceState

**Description:**

This function immediately changes the radio state. If the Radio is in Transmit or Receive modes, then WUSB_NL_bAbort MUST be called before calling WUSB_NL_ForceState.

**C Prototype:**

```
void  WUSB_NL_ForceState(WUSB_NL_XACT_CONFIG wEndStateBitsOnly)
```

**Assembly:**

```
mov   X, >wEndStateBitsOnly
mov   A, <wEndStateBitsOnly
lcall WUSB_NL_ForceState
```

**Parameters:**

wEndStateBitsOnly - Immediate new state for the radio. The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Value | Description |
|---|---|---|
| WUSB_NL_END_STATE_SLEEP | 0 | Put the radio into sleep mode |
| WUSB_NL_END_STATE_IDLE | 1 | Wake up the radio from sleep mode |

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetFrameConfig

**Description:**

This function sets the frame configuration composed of two parts: packet length enable and syncword length. Syncword length can be 64, 48, 32, or 16 bits. Enabling packet length causes the radio to treat the first byte of the payload as the length.

**C Prototype:**

```
void  WUSB_NL_SetFrameConfig(WUSB_NL_FRAME_CONFIG bConfig)
```

**Assembly:**

```
mov   A, bConfig
lcall WUSB_NL_SetFrameConfig
```

**Parameters:**

bConfig - Frame configuration is composed of:

a) Syncword length; possible values are:

SYNC_WRD_64_BITS

SYNC_WRD_48_BITS

SYNC_WRD_32_BITS

SYNC_WRD_16_BITS

WUSB_NL_SOP_LEN

b) Packet length enable; possible value is LEN_EN.

| Symbolic Name | Value |
|---|---|
| WUSB_NL_SYNC_WRD_16_BITS | 0x00 |
| WUSB_NL_SYNC_WRD_32_BITS | 0x08 |
| WUSB_NL_SYNC_WRD_48_BITS | 0x10 |
| WUSB_NL_SYNC_WRD_64_BITS | 0x18 |
| WUSB_NL_LEN_EN | 0x20 |
| WUSB_NL_SOP_LEN | 0x18 |

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bGetFrameConfig

**Description:**

This function returns the packet format of the NL radio.

**C Prototype:**

```
WUSB_NL_FRAME_CONFIG WUSB_NL_bGetFrameConfig(void)
```

**Assembly:**

```
lcall WUSB_NL_bGetFrameConfig
```

**Parameters:**

None

**Return Value:**

This function returns the current framing configuration.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetPreambleCount

**Description:**

This function sets the preamble length in bytes. Maximum length of the preamble is 8.

**C Prototype:**

```
void  WUSB_NL_SetPreambleCount(BYTE bCount)
```

**Assembly:**

```
mov   A, bCount
lcall WUSB_NL_SetPreambleCount
```

**Parameters:**

bCount - 8-bit Preamble length that is passed to the function. The symbolic names given in C and assembly, and their associated preamble lengths, are listed in the following table:

| Symbolic Name | Value |
|---|---|
| WUSB_NL_PREAMBLE_LEN_1_BYTE | 0x01 |
| WUSB_NL_PREAMBLE_LEN_2_BYTE | 0x02 |
| WUSB_NL_PREAMBLE_LEN_3_BYTE | 0x03 |
| WUSB_NL_PREAMBLE_LEN_4_BYTE | 0x04 |
| WUSB_NL_PREAMBLE_LEN_5_BYTE | 0x05 |
| WUSB_NL_PREAMBLE_LEN_6_BYTE | 0x06 |
| WUSB_NL_PREAMBLE_LEN_7_BYTE | 0x07 |
| WUSB_NL_PREAMBLE_LEN_8_BYTE | 0x08 |

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bGetPreambleCount

**Description:**

This function gets the preamble length in bytes. The maximum length of the preamble is 8.

**C Prototype:**

```
BYTE WUSB_NL_bGetPreambleCount(void)
```

**Assembly:**

```
lcall WUSB_NL_bGetPreambleCount
```

**Parameters:**

None

**Return Value:**

This function returns the preamble repetition count.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetCrcSeed

**Description:**

This function sets the CRC seed value for both transmit and receive. As the NL supports only an 8-bit CRC, the MSB of the CRC seed is ignored.

**C Prototype:**

```
void  WUSB_NL_SetCrcSeed(WORD wCrcSeed)
```

**Assembly:**

```
mov   X, >wCrcSeed
mov   A, <wCrcSeed
lcall WUSB_NL_SetCrcSeed
```

**Parameters:**

wCrcSeed - the CRC seed value.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_wGetCrcSeed

**Description:**

This function returns the current CRC seed value. As the NL supports only an 8-bit CRC, the MSB of CRC Seed can be ignored.

**C Prototype:**

```
WORD WUSB_NL_wGetCrcSeed(void)
```

**Assembly:**

```
lcall WUSB_NL_wGetCrcSeed
```

**Parameters:**

None

**Return Value:**

This function returns the CRC seed value for both transmit and receive. As the NL supports only an 8-bit CRC, the MSB of the CRC seed can be ignored.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bGetRssi

**Description:**

This function returns the receiver signal strength indicator.

**C Prototype:**

```
WUSB_NL_RSSI WUSB_NL_bGetRssi(void)
```

**Assembly:**

```
lcall WUSB_NL_bGetRssi
```

**Parameters:**

None

**Return Value:**

This function returns the Receive Signal Strength Indicator value. When read completes, the return value can be the signal strength of the received packet or of the noise. If the MSB of the return value is 0, then the signal strength is for the received packet; if not, it is for noise. While in the receive state, but before a packet is received, this can be called continuously to check the carrier strength at the current frequency. It is necessary that WUSB_NL_bGetRssi is called at least 100 us after the radio enters Receive mode.

The symbolic names of bitfield masks given in C and asembly, their associated values, and descriptions are listed in the following table.

| Symbolic Name | Value | Description |
|---|---|---|
| WUSB_NL_NOISE_RSSI | 0x80 | Flag which indicates if the RSSI value is noise or signal |
| WUSB_NL_RSSI_LVL_MSK | 0x3F | Mask for value of RSSI level |

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_StartTransmit

**Description:**

This function starts the non-blocking transmission of a packet. The location of the packet buffer to transmit must have previously been set with a call to WUSB_NL_SetPtr.

After starting the transmission of a packet with this call, the state of the transmit operation should be checked by calling the WUSB_NL_bGetTransmitState. When WUSB_NL_bGetTransmitState indicates that the transmission has completed, WUSB_NL_EndTransmit must be called.

After calling WUSB_NL_StartTransmit, NO CALLS can be made to the configuration access routines until the transmit operation is terminated with a call to WUSB_NL_EndTransmit or WUSB_NL_Abort. Until one of those calls is made to end the transmit operation the only other call supported is WUSB_NL_bGetTransmitState.

**C Prototype:**

```
void  WUSB_NL_StartTransmit(BYTE bRetryCount, WUSB_NL_LENGTH bLength)
```

**Assembly:**

```
mov   A, bRetryCount
mov   X, bLength
lcall WUSB_NL_StartTransmit
```

**Parameters:**

bRetryCount - number of times the packet should be retried it the transmit fails.

bLength - length of the packet in bytes.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bGetTransmitState

**Description:**

This call must be made after starting a transmit operation with the WUSB_NL_StartTransmit function. The value returned is of the type WUSB_NL_STATE and can take a value of WUSB_NL_COMPLETE, WUSB_NL_TX, or WUSB_NL_ERROR.

**C Prototype:**

```
WUSB_NL_STATE WUSB_NL_bGetTransmitState(void)
```

**Assembly:**

```
lcall WUSB_NL_bGetTransmitState
```

**Parameters:**

None

**Return Value:**

Returns the state of the current transmit operation.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_EndTransmit

**Description:**

This function completes a transmit operation.

**C Prototype:**

```
void  WUSB_NL_EndTransmit(void)
```

**Assembly:**

```
lcall WUSB_NL_EndTransmit
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bAbort

**Description:**

This function aborts a transmit or receive operation and transitions the radio to the IDLE state. This API always returns a zero.

**C Prototype:**

```
WUSB_NL_LENGTH WUSB_NL_bAbort(void)
```

**Assembly:**

```
lcall WUSB_NL_bAbort
```

**Parameters:**

None

**Return Value:**

Zero.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bBlockingTransmit

**Description:**

This function transmits a packet, and blocks execution until it completes. The function attempts to transmit a packet. The address of the packet buffer should have previously been set with a call to WUSB_NL_SetPtr.

**C Prototype:**

```
WUSB_NL_STATE WUSB_NL_bBlockingTransmit(BYTE bRetryCount, WUSB_NL_LENGTH bLength)
```

**Assembly:**

```
mov   A, bRetryCount
mov   X, bLength
lcall WUSB_NL_bBlockingTransmit
```

**Parameters:**

bRetryCount - number of times the packet should be retried if the transmit fails.

bLength - length, in bytes, of the packet.

**Return Value:**

8-bit value of the WUSB_NL_STATE type which indicates the radio state.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_StartReceive

**Description:**

This function starts the reception of a packet. The location and length of the packet buffer to receive the data into must have previously been set with calls to WUSB_NL_SetPtr and WUSB_NL_SetLength.

After starting the reception of a packet with this call, the state of the receive operation should be checked by calling WUSB_NL_bGetReceiveState. When WUSB_NL_bGetReceiveState indicates that the transmission is complete, WUSB_NL_EndReceive must be called.

After calling WUSB_NL_StartReceive NO CALLS can be made to the configuration access routines until the receive operation is terminated with a call to WUSB_NL_EndReceive or WUSB_NL_Abort.

Until one of those calls is made to end the receive operation, the only other calls supported are WUSB_NL_bGetReceiveState and WUSB_NL_bGetRssi.

**C Prototype:**

```
void  WUSB_NL_StartReceive(void)
```

**Assembly:**

```
lcall WUSB_NL_StartReceive
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_bGetReceiveState

**Description:**

This call should be made after starting a receive operation with the WUSB_NL_StartReceive function.

This function checks for errors during reception or completion of reception and returns either WUSB_NL_ERROR or WUSB_NL_COMPLETE. If reception completes with no errors, received data is available in the buffer pointed to by the most recent call to WUSB_NL_SetPtr.

**C Prototype:**

```
WUSB_NL_STATE WUSB_NL_bGetReceiveState(void)
```

**Assembly:**

```
lcall WUSB_NL_bGetReceiveState
```

**Parameters:**

None

**Return Value:**

Returns the state of the current receive operation.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_EndReceive

**Description:**

This function completes a receive operation.

**C Prototype:**

```
WUSB_NL_LENGTH WUSB_NL_bEndReceive(void)
```

**Assembly:**

```
lcall WUSB_NL_bEndReceive
```

**Parameters:**

None

**Return Value:**

Returns the length of the packet that was received. If the packet payload truncation occurs (due to inadequate buffer length) it does not change this return value.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetPtr

**Description:**

This function sets the buffer pointer address for WUSB_NL_StartTransmit, WUSB_NL_bBlockingTransmit, and WUSB_NL_StartReceive functions.

**C Prototype:**

```
void  WUSB_NL_SetPtr(WUSB_NL_BUFFER_PTR bRamPtr)
```

**Assembly:**

```
mov   A, bRamPtr
lcall WUSB_NL_SetPtr
```

**Parameters:**

bRamPtr - pointer to RAM buffer for future transmit and receive operations.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_SetLength

**Description:**

This function sets the buffer length pointed to by most recent call to RadioSetPtr.

**C Prototype:**

```
void  WUSB_NL_SetLength(WUSB_NL_LENGTH bLength)
```

**Assembly:**

```
mov   A, bLength
lcall WUSB_NL_SetLength
```

**Parameters:**

bLength - length of the buffer pointed to by the most recent call to WUSB_NL_SetPtr.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_WriteRegister

**Description:**

This function writes the value to the register.

**C Prototype:**

```
void  WUSB_NL_WriteRegister(WUSB_NL_REG_ADDR bRegAddr, WORD wValue)
```

**Assembly:**

```
mov   A, >wValue
push  A
mov   A, <wValue
push  A
mov   A, bRegAddr
push  A
lcall WUSB_NL_WriteRegister
```

**Parameters:**

bRegAddr - register address. The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Value |
|---|---|
| WUSB_NL_ANALOG_BLK_CONFIG1_ADR | 0 |
| WUSB_NL_ANALOG_BLK_CONFIG2_ADR | 1 |
| WUSB_NL_RX_CTRL_ADR | 2 |
| WUSB_NL_TX_RX_STS_ADR | 3 |
| WUSB_NL_BPF_CTRL1_ADR | 4 |
| WUSB_NL_BPF_CTRL2_ADR | 5 |
| WUSB_NL_RSSI_VAL_ADR | 6 |
| WUSB_NL_RF_SYNTH_TX_RX_CTRL_ADR | 7 |
| WUSB_NL_TX_DAC_CTRL_ADR | 8 |
| WUSB_NL_PA_CTRL_ADR | 9 |
| WUSB_NL_AMS_TST_CTRL1_ADR | 10 |
| WUSB_NL_AMS_TST_CTRL2_ADR | 11 |
| WUSB_NL_AMS_TST_CTRL3_ADR | 12 |
| WUSB_NL_RX_IF_CTRL_ADR | 13 |

| Symbolic Name | Value |
|---|---|
| WUSB_NL_PM_CTRL_ADR | 14 |
| WUSB_NL_RSRVD_CTRL1_ADR | 15 |
| WUSB_NL_RSRVD_CTRL2_ADR | 16 |
| WUSB_NL_RSRVD_CTRL3_ADR | 17 |
| WUSB_NL_RSRVD_CTRL4_ADR | 18 |
| WUSB_NL_RSRVD_CTRL5_ADR | 19 |
| WUSB_NL_RSRVD_CTRL6_ADR | 20 |
| WUSB_NL_RSRVD_CTRL7_ADR | 21 |
| WUSB_NL_PWRMGMT_ADR | 22 |
| WUSB_NL_TX_RX_VCO_CAL_CTRL_ADR | 23 |
| WUSB_NL_FRACT_N_VCO_CAL_CTRL1_ADR | 24 |
| WUSB_NL_FRACT_N_VCO_CAL_CTRL2_ADR | 25 |
| WUSB_NL_VCO_SET_CTRL_ADR | 26 |
| WUSB_NL_VCO_N_CTYST_TRIM_ADR | 27 |
| WUSB_NL_FREQ_VAL_ADR | 28 |
| WUSB_NL_MAN_REV_CODE_ADR | 29 |
| WUSB_NL_MAN_ID_LSB_ADR | 30 |
| WUSB_NL_MAN_ID_MSB_ADR | 31 |
| WUSB_NL_CONFIG_ADR | 32 |
| WUSB_NL_DELAY_REG0_ADR | 33 |
| WUSB_NL_DELAY_REG1_ADR | 34 |
| WUSB_NL_MISC1_ADR | 35 |
| WUSB_NL_SYNC_WORD1_ADR | 36 |
| WUSB_NL_SYNC_WORD2_ADR | 37 |
| WUSB_NL_SYNC_WORD3_ADR | 38 |
| WUSB_NL_SYNC_WORD4_ADR | 39 |
| WUSB_NL_THRESHOLD_REG_ADR | 40 |
| WUSB_NL_MISC2_ADR | 41 |
| WUSB_NL_MISC3_ADR | 42 |
| WUSB_NL_RSRVD_CTRL8_ADR | 43 |
| WUSB_NL_RSRVD_CTRL9_ADR | 44 |
| WUSB_NL_RSRVD_CTRL10_ADR | 45 |

| Symbolic Name | Value |
|---|---|
| WUSB_NL_RSRVD_CTRL11_ADR | 46 |
| WUSB_NL_RSRVD_CTRL12_ADR | 47 |
| WUSB_NL_MAIN_STS_REG_ADR | 48 |
| WUSB_NL_RSRVD_CTRL13_ADR | 49 |
| WUSB_NL_TX_RX_FIFO_REG_ADR | 50 |
| WUSB_NL_RSRVD_CTRL14_ADR | 51 |
| WUSB_NL_FIFO_RD_PTR_REG_ADR | 52 |

wValue - value to be written to the register.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_wReadRegister

**Description:**

This function returns the value of the register.

**C Prototype:**

```
WORD WUSB_NL_wReadRegister(WUSB_NL_REG_ADDR bRegAddr)
```

**Assembly:**

```
mov   A, bRegAddr
lcall WUSB_NL_wReadRegister
```

**Parameters:**

bRegAddr - register address. For the list of the register address symbolic names given in C and assembly, and their associated values, refer to the appropriate table given for the WUSB_NL_wReadRegister API function.

**Return Value:**

Returns the value of the register.

**Side Effects:**

See Note ** at the beginning of the API section.

## WUSB_NL_FECEnable

**Description:**

This function enables or disables the Forward Error Correction radio feature.

**C Prototype:**

```
void  WUSB_NL_FECEnable(BOOL bEnable)
```

**Assembly:**

```
mov   A, bEnable
lcall WUSB_NL_FECEnable
```

**Parameters:**

bEnableflag – if 1, FEC23 is enabled else disabled.

**Return Value:**

None

**Side Effects:**

See Note ** at the beginning of the API section.

## Sample Firmware Source Code

The following sample code demonstrates the radio packets transmission. The sample code examples were developed using the CY3668 WirelessUSB NL Development Kit.

The sample code project continuously loops in the RX state. If the Button is pressed the project transmits the packets and indicates if the packets are transmitted successfully or failed.

The LED1 indicates a transmission error. The LED2 indicates successful transmission.

Table 1. Sample Code Project Settings

| | CY7C601xx/2xx, CY7C638xx | CY7C604xx, CY7C643xx |
|---|---|---|
| Global Resources | CPU Clock: Internal (24 MHz) | IMO Setting: 12MHz |
| | CPU Clock / N: SysClk/2 | CPU_Clock: SysClk/1 |
| WUSB_NL User Module Parameters | Name: WUSB_NL | Name: WUSB_NL |
| | FEC: Enable | FEC: Enable |
| | SS_Port: Port_1 | SS_Port: Port_1 |
| | SS_Pin: Port_1_3 | SS_Pin: Port_1_7 |
| | RST_Port: Port_1 | RST_Port: Port_2 |
| | RST_Pin: Port_1_2[1] | RST_Pin: Port_2_4 |
| | Clock Divider: 12 | Clock Divider: SysClk/8 |

| | CY7C601xx/2xx, CY7C638xx | | CY7C604xx, CY7C643xx | |
|---|---|---|---|---|
| Pins Configuration | Port_0_0[2] | Name: Button | Port_0_0[2] | Name: Button |
| | | Select: Input-CMOS | | Select: StdCPUInput |
| | | Drive: Open Drain, Pullup | | Drive: Pull Up |
| | Port_0_1[3] | Name: LED1 | Port_0_1[3] | Name: LED1 |
| | | Select: Output-8 mA | | Select: StdCPU |
| | | Drive: Push/Pull | | Drive: Strong |
| | Port_0_2[4] | Name: LED2 | Port_0_2[4] | Name: LED2 |
| | | Select: Output-8 mA | | Select: StdCPU |
| | | Drive: Push/Pull | | Drive: Strong |

1. P1_2 should be wired to P2_4 on the board.
2. P0_0 should be wired to S1 on the board.
3. P0_1 should be wired to LED1 on the board.
4. P0_2 should be wired to LED2 on the board.

**Note** Parameters that are not mentioned in the previous table must be left with their default values.

## C Sample Code Example

The WUSB_NL User Module sample code written in C:

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#define LED1_BLINK   {PRT0DR &= ~LED1_MASK; TIME_DELAY; PRT0DR |= LED1_MASK;}
#define LED2_BLINK   {PRT0DR &= ~LED2_MASK; TIME_DELAY; PRT0DR |= LED2_MASK;}
#define TIME_DELAY   {for(i=30000; i>0; i--);}

WUSB_NL_STATE bState;
WORD i;
BYTE arbTxData[] = {1,2,3,4,5,6,7};  // initialize transmission buffer
BYTE arbRxData[7];

void main(void)
{
    PRT0DR |= LED1_MASK | LED2_MASK;  // both LEDs off

    WUSB_NL_Init(WUSB_NL_ACK_EN|WUSB_NL_ACK_TO_15X, WUSB_NL_PA_1_DBM);
    WUSB_NL_SetFrameConfig(WUSB_NL_LEN_EN|WUSB_NL_SOP_LEN);
    WUSB_NL_SetChannel(0x02);  // set radio channel to 2
    WUSB_NL_SetLength(sizeof(arbRxData));

    while(1)
    {
        if(!(PRT0DR & Button_MASK))
        {
            WUSB_NL_SetPtr(arbTxData);  // load the arbTxData array pointer
            bState = WUSB_NL_bBlockingTransmit(2, sizeof(arbTxData));
```

```
            if(bState & WUSB_NL_ERROR) LED1_BLINK;  // indicate failed transmission
          if(bState & WUSB_NL_COMPLETE) LED2_BLINK;  // indicate successful transmission

            PRT0DR |= Button_MASK;

            TIME_DELAY;
        }
        else
        {
            WUSB_NL_SetPtr(arbRxData);  // load the arbRxData array pointer
            WUSB_NL_StartReceive();

            do
            {
                PRT0DR |= Button_MASK;
                if(!(PRT0DR & Button_MASK)) break;  // polling the button

                bState = WUSB_NL_bGetReceiveState();
            }
           while((bState & (WUSB_NL_ERROR | WUSB_NL_COMPLETE)) == 0 );  // check the
receive state

            WUSB_NL_bEndReceive();

           if(bState & WUSB_NL_ERROR) LED1_BLINK;  // indicate failed reception
           if(bState & WUSB_NL_COMPLETE) LED2_BLINK;  // indicate successful reception
        }
    }
}
```

## Assembly Sample Code Example

The equivalent code written in Assembly:

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

macro LED1_BLINK
   and   reg[PRT0DR], ~LED1_MASK
   TIME_DELAY
   or    reg[PRT0DR], LED1_MASK
endm

macro LED2_BLINK
   and   reg[PRT0DR], ~LED2_MASK
   TIME_DELAY
   or    reg[PRT0DR], LED2_MASK
endm

macro TIME_DELAY
   mov   A, ffh
   mov   X, ffh
```

```
.timeloop:
   nop
   dec   X
   jnc   .timeloop
   dec   A
   jnc   .timeloop
endm

AREA   InterruptRAM(RAM, REL, CON)

arbTxData: BLK 7
arbRxData: BLK 7

AREA   text(ROM, REL, CON)

_main:
; initialize transmission buffer
   mov   [arbTxData], 1
   mov   [arbTxData+1], 2
   mov   [arbTxData+2], 3
   mov   [arbTxData+3], 4
   mov   [arbTxData+4], 5
   mov   [arbTxData+5], 6
   mov   [arbTxData+6], 7
   or    reg[PRT0DR], LED1_MASK | LED2_MASK  ; both LEDs off
   mov   A, >WUSB_NL_PA_1_DBM
   push  A
   mov   A, <WUSB_NL_PA_1_DBM
   push  A
   mov   A, WUSB_NL_ACK_EN
   push  A
   mov   A, WUSB_NL_ACK_TO_15X
   push  A  ; load the Init function parameters into the stack
   lcall WUSB_NL_Init
   add   SP, -4
   mov   A, WUSB_NL_LEN_EN|WUSB_NL_SOP_LEN
   lcall WUSB_NL_SetFrameConfig
   mov   A, 2  ; set radio channel to 2
   lcall WUSB_NL_SetChannel
   mov   A, 7  ; set the arbRxData array lenth
   lcall WUSB_NL_SetLength

.MainLoop:
; load the arbRxData array pointer
   mov   A, >arbRxData
   mov   X, <arbRxData
   lcall WUSB_NL_SetPtr
   lcall WUSB_NL_StartReceive

.RxLoop:
   or    reg[PRT0DR], Button_MASK
   tst   reg[PRT0DR], Button_MASK  ; polling the button
   jz    .ButtonPressed
   lcall WUSB_NL_bGetReceiveState
   and   A, WUSB_NL_ERROR | WUSB_NL_COMPLETE  ; check the receive state
```

```
   jz    .RxLoop
   push  A
   lcall WUSB_NL_bEndReceive
   pop   A
   push  A
   and   A, WUSB_NL_ERROR
   jz    .RxStError
   LED1_BLINK  ; indicate successful reception

.RxStError:
   pop   A
   and   A, WUSB_NL_COMPLETE
   jz    .MainLoop
   LED2_BLINK  ; indicate failed reception
   jmp   .MainLoop

.ButtonPressed:
   lcall WUSB_NL_bEndReceive
   ; load the arbTxData array pointer
   mov   A, >arbTxData
   mov   X, <arbTxData
   lcall WUSB_NL_SetPtr
   mov   X, 7  ; set the arbTxData array lenth
   mov   A, 2  ; set the retry count
   lcall WUSB_NL_bBlockingTransmit
   push  A
   and   A, WUSB_NL_ERROR
   jz    .TxStError
   LED1_BLINK  ; indicate successful transmission

.TxStError:
   pop   A
   and   A, WUSB_NL_COMPLETE
   jz    .TxStCompl
   LED2_BLINK  ; indicate failed transmission

.TxStCompl:
   TIME_DELAY
   jmp   .MainLoop
```

## Configuration Registers

The WUSB_NL User Module uses the SPI digital PSoC block. The block is personalized and parameterized through a set of registers. The set of registers used by the user module with brief descriptions are given in this section. Symbolic names for these registers are defined in the user module instance's C and assembly language interface files (the ".h" and ".inc" files).

### CY7C601xx/2xx, CY7C638xx SPI Block Registers

- Bank 0
  - SPI Configuration Register: SPICR_REG

    This register is the Configuration Register for the SPI block of the WUSB_NL User Module.

- SPI Pin Configuration Register: SPIUM_REG

  This register is the SPI Pin Configuration Register for the SPI block of the WUSB_NL User Module.

- SPI Transmit Data Register: SPI_TX_REG

  This register is the SPI Transmit Data Register for the SPI block of the WUSB_NL User Module.

- SPI Receive Data Register: SPI_RX_REG

  This register is the SPI Receive Data Register for the SPI block of the WUSB_NL User Module.

### *CY7C604xx, CY7C643xx SPI Block Registers*

- Bank 1
  - SPI Configuration Register: SPI_CFG_REG

    This register is used to configure the SPI block for the WUSB_NL User Module.
- Bank 0
  - SPI Control Register: SPI_CR_REG

    This register is the Control Register for the SPI block of the WUSB_NL User Module.
  - SPI Transmit Data Register: SPI_TX_REG

    This register is the SPI Transmit Data Register for the SPI block of the WUSB_NL User Module.
  - SPI Receive Data Register: SPI_RX_REG

    This register is the SPI Receive Data Register for the SPI block of the WUSB_NL User Module.

## *Version History*

| Version | Originator | Description |
|---|---|---|
| 1.00 | DHA | Initial version. |
| 2.00 | DHA | 1. Updated WUSB_NL_ForceState API description in the user module datasheet.<br>2. ACK Timeout parameter min value changed to 1.<br>3. Improved error messaging in case of improper UM confirguration. |

**Note** PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.



Copyright © 2011-2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.