Embedded in Life

An IXYS Company

## ZAURA™ RF Wireless Technology

# ZAURA RF Wireless Library

## Programmer's Reference Manual

RM006003-1011

Copyright © 2011 Zilog®, Inc. All rights reserved.

www.zilog.com

zilog
Embedded in Life
An IXYS Company

**Warning:** DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

## LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

### As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

### Document Disclaimer

©2011 Zilog, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

ZAURA is a trademark of Zilog, Inc. All other product or service names are the property of their respective owners.

# Revision History

Each instance in this document's revision history reflects a change from its previous version. For more details, refer to the corresponding pages linked in the table below.

| Date | Revision Level | Description | Page No. |
|---|---|---|---|
| Oct 2011 | 03 | Corrected ZAURA_RF_ UserParams value in Additional Radio Configuration Variables table. | 20 |
| Oct 2011 | 02 | Updated Library/APIs to accommodate expanding radio frequencies. | All |
| Feb 2011 | 01 | Original issue. | All |

# Table of Contents

Revision History . . . . . iii
List of Figures. . . . . ix
List of Tables . . . . . xi
The ZAURA RF Wireless Library . . . . . .1
ZAURA RF Wireless Module Topology . . . . . .1
Node Addresses . . . . . .2
Radio Frequencies . . . . . .3
ZAURA RF Wireless Cell Network Identifiers . . . . . .4
ZAURA RF Wireless Framing . . . . . .5
Data Transfer Methods . . . . . .5
Frame Formats . . . . . .6
Reliable Data Transfer & Flow Control Protocol . . . . . .11
Service Access Points . . . . . .11
Channel Access Rules . . . . . .12
Radio States . . . . . .14
ZAURA RF Wireless Module Configuration . . . . . .15
Radio Configuration . . . . . .15
Other Radio Configuration Options . . . . . .17
ZAURA RF Shell Configuration Options . . . . . .21
ZAURA RF Wireless API Reference. . . . . .23
ZAURA_RF_Init . . . . . .23
ZAURA_RF_GpioConfig . . . . . .25
ZAURA_RF_GetState . . . . . .27
ZAURA_RF_SetState . . . . . .28
ZAURA_RF_Transmit . . . . . .30
ZAURA_RF_SendData . . . . . .33
ZAURA_RF_SendSeqData . . . . . .34
ZAURA_RF_SendPkt . . . . . .36
ZAURA_RF_Receive . . . . . .38

ZAURA_RF_FreeBuf . . . . . 39
ZAURA_RF_ReadRSSI . . . . . 40
ZAURA_RF_Rssi2Pwr . . . . . 42
ZAURA RF Wireless Radio Configuration API. . . . . 43
ZAURA_RF_GetParams . . . . . 44
ZAURA_RF_SetParams . . . . . 45
ZAURA_RF_GetAddr . . . . . 46
ZAURA_RF_SetAddr . . . . . 47
ZAURA_RF_GetNID . . . . . 48
ZAURA_RF_SetNID . . . . . 49
ZAURA_RF_GetRxFilter . . . . . 50
ZAURA_RF_SetRxFilter . . . . . 51
ZAURA_RF_GetChannel . . . . . 52
ZAURA_RF_SetChannel . . . . . 53
ZAURA_RF_GetTxPwr . . . . . 54
ZAURA_RF_SetTxPwr . . . . . 55
Variable Types and Structures . . . . . 57
Types . . . . . 57
Structures . . . . . 57
ZAURA_RF_PKT_BUF Structure . . . . . 58
ZAURA_RF_NID Structure . . . . . 58
ZAURA_RF_PARAMS Structure . . . . . 59
ZAURA_RF_STATS Structure . . . . . 59
ZAURA RF Wireless Shell API Reference . . . . . 61
Configuring ZAURA RF Shell Commands . . . . . 61
Creating User-Defined ZAURA RF Shell Commands . . . . . 62
Console Global Variables . . . . . 63
ZAURA RF Wireless Shell API Descriptions . . . . . 64
ZAURA_RF_ProcessCommandLine() . . . . . 65
ZAURA_RF_ShellAddCmd . . . . . 66
ZAURA_RF_ShellAtox . . . . . 69

ZAURA_RF_ShellStricmp . . . . . .70
ZAURA_RF_ShellHexDump . . . . . .71
ZAURA_RF_ShellControl . . . . . .72
ZAURA_RF_ShellPrintf . . . . . .73
Timer API Functions . . . . . .77
ZAURA_RF_TickDelay . . . . . .77
ZAURA_RF_ms_TO_TICKS . . . . . .78
ZAURA_RF_ReadTimer . . . . . .79
ZAURA_RF_GetTicks . . . . . .80
Appendix A. Project Information . . . . . .81
Memory Map . . . . . .81
Z8F2480 MCU Peripherals . . . . . .81
Using ZDS II to Create an Application . . . . . .83
Create an Application . . . . . .84
Customer Support . . . . . .87

# List of Figures

Figure 1. Fields within the DA Frame . . . . . .7
Figure 2. Fields within the DA_SA Frame . . . . . .9
Figure 3. Fields within the DA_SA_CTRL Frame . . . . . .10

# List of Tables

Table 1. Center Frequencies in the 863–870 MHz Band . . . . . .3
Table 2. Center Frequencies in the 902–928 MHz Band . . . . . .3
Table 3. Service Access Point Labels and Functions . . . . . .11
Table 4. ZAURA_RF_PARAMS Structure Settings . . . . . .15
Table 5. Additional Radio Configuration Variables . . . . . .17
Table 6. UART Configuration Variables . . . . . .21
Table 7. Z8F2480 MCU Configuration Variables . . . . . .22
Table 8. Types of Variables . . . . . .57
Table 9. ZAURA_RF_PKT_BUF Structure Types . . . . . .58
Table 10. ZAURA_RF_NID Structure Types . . . . . .58
Table 11. ZAURA_RF_PARAMS Structure Type . . . . . .59
Table 12. ZAURA_RF_STATS Structure Types . . . . . .59
Table 13. Console Global Variables . . . . . .63
Table 14. Library Shell Commands . . . . . .66
Table 15. Z8F2480 MCU GPIOs . . . . . .82

# The ZAURA RF Wireless Library

This ZAURA RF Wireless Library Programmer's Reference Manual describes the architecture and application programming interface that allows software developers to integrate ZAURA RF Wireless modules into their products.

Zilog's ZAURA RF Wireless Modules can be designed into any system to enable wireless control. The ZAURA RF Wireless Library is used to establish an ad-hoc peer-to-peer network over radio frequencies in the Industrial, Scientific and Medical (ISM) band with raw data rates of 50 Kbits/sec. Nodes with compatible configurations within radio range of each other can communicate directly using either point-to-point or point-to-multipoint data transfers. The ZAURA RF Library allows developers to configure, transmit and receive packets on the networks.

## ZAURA RF Wireless Module Topology

The ZAURA RF Wireless Module uses an ad-hoc topology. Nodes with compatible configurations within radio range of each other communicate directly using either point-to-point (unicast) or point-to-multipoint (broadcast) data transfer. There is no central coordinating unit in the ZAURA RF Wireless Network through which nodes must communicate. This network does not provide routing of any kind between nodes that are not within radio range of each other. There is no concept of master or slave in the ZAURA RF Wireless Network; all nodes operate as equal peers. The modules' configuration parameters determine whether nodes will be able communicate with one another when they are within range.

The ZAURA RF Wireless Network configuration parameters consist of the RF Channel, Network ID and Frame Format. There are two versions of the ZAURA RF Wireless Module, one using the 868 MHz Frequency band and the other using the 915 MHz frequency band. The ZAURA RF channel determines which subset of frequencies within the band are used

for communications. The Network ID is user-configurable value 1-4 bytes long that logically subdivides a channel into separate groups or cells. ZAURA RF Modules using the same channel but different Network ID's will not be able to communicate with each other.

The ZAURA RF Wireless Network provides both unreliable and reliable data transfer services, depending on the value of the Frame Format configuration parameter. There are three different frame formats. The first, a DA format, is an unreliable data transfer service with CRC checking and up to 58 bytes of application data sent per frame. The second frame format, DA_SA, is the same as the DA frame format except that it adds the source address to the headers. The third frame format, DA_SA_CTRL, provides a reliable data transfer service. Of the three frame formats, the DA_SA_CTRL format incurs the most overhead and is designed for instances in which reliable data transfer is more important than throughput.

## Node Addresses

Each node within the ZAURA RF Wireless Network should have a unique 8-bit address that allows all other nodes to identify that node. This is a mandatory requirement if the ZAURA RF Wireless Network is configured to use a frame format that includes the source address field. However, if all nodes are configured to use frames that only carry a destination address and all transmissions are broadcast, then there is no requirement for unique node addresses.

A node can be assigned an address in the range `0x01` to `0xFE` (i.e., 254 unique node addresses). Node address `0xFF` is used as the broadcast address; i.e., when a frame is transmitted to the `0xFF` address, it will be received by all nodes within radio range of the transmitter. Node address `0x00` is reserved; it must not be used as a node address and should not be transmitted in either the destination or source address fields of a packet.

The manner in which addresses are assigned to nodes is up to the implementer. The ZAURA RF Wireless Library contains a user-configurable node address that is stored in Flash.

# Radio Frequencies

The ZAURA RF Wireless Module uses radio frequencies in the 863–870 MHz or 902–928 MHz bands. There are separate ZAURA RF Wireless Module libraries for each band.

The 863–870 MHz band is divided into 4 channels with 600 kHz channel spacing and with the center frequencies indicated in Table 1.

**Table 1. Center Frequencies in the 863–870 MHz Band**

| Channel | Center Frequency (MHz) |
|---|---|
| 0 | 865.6 |
| 1 | 866.2 |
| 2 | 866.8 |
| 3 | 867.4 |

The 902–928 MHz band is divided into 25 channels with 1 MHz channel spacing and with the following center frequencies indicated in Table 2.

**Table 2. Center Frequencies in the 902–928 MHz Band**

| Channel | Center Frequency (MHz) | Channel | Center Frequency (MHz) |
|---|---|---|---|
| 0 | 903 | 13 | 916 |
| 1 | 904 | 14 | 917 |
| 2 | 905 | 15 | 918 |
| 3 | 906 | 16 | 919 |
| 4 | 907 | 17 | 920 |

**Table 2. Center Frequencies in the 902–928 MHz Band (Continued)**

| Channel | Center Frequency (MHz) | Channel | Center Frequency (MHz) |
|---|---|---|---|
| 5 | 908 | 18 | 921 |
| 6 | 909 | 19 | 922 |
| 7 | 910 | 20 | 923 |
| 8 | 911 | 21 | 924 |
| 9 | 912 | 22 | 925 |
| 10 | 913 | 23 | 926 |
| 11 | 914 | 24 | 927 |
| 12 | 915 | | |

All data communication occurs at 50 kbps with a Frequency Deviation (FDEV) of 100 kHz and a Modulation Index (MI) of 4, regardless of the frequency band used.

The ZAURA RF Wireless Network does not implement frequency hopping. After a node has been configured to operate on a particular channel, it will continue to use that channel until the node is configured to use a different channel.

# ZAURA RF Wireless Cell Network Identifiers

A ZAURA RF Wireless cell is composed of up to 254 peer nodes. Each ZAURA RF Wireless cell operates independently of all other cells; each cell uses a unique Network Identifier (NID). The size of the NID is a user-configurable parameter that must be between 1 and 4 bytes in length. Because it may at times be difficult for the radio hardware to distinguish between a valid Network ID and noise (or the absence of any FSK signal), network identifiers should not contain long sequences of repeated binary digits. For example, the 16-bit Network ID `0x0001` is a poor choice; whereas `0xA5B9` provides much better immunity to noise.

ZAURA RF Wireless cells operating within radio range of one another can cause interference. Therefore, overlapping cells must have different NID values. If possible, overlapping cells should use different RF channels. ZAURA RF Wireless addresses are not globally unique and can be reused in adjacent cells.

# ZAURA RF Wireless Framing

This section describes the format of frames the ZAURA RF Wireless nodes use to communicate. In this document a frame is used to describe the collection of bits transmitted and received by the radio and a packet is used to describe the portion of the frame that resides in host memory.

## Data Transfer Methods

Each ZAURA RF Wireless Module accommodates two methods of data transfer: *reliable* and *unreliable*. Each is described in this section.

### Unreliable Data Transfer

The unreliable data transfer mechanism employed by the ZAURA RF Wireless Library allows for either broadcast (point-to-multipoint) or unicast (point-to-point) communication. After an unreliable data frame has been successfully transmitted, the sender does not receive any indication that the frame was received by the intended recipient(s). This type of data transfer is supported by all frame formats.

### Reliable Data Transfer

Reliable data transfer is only possible when all nodes within the ZAURA RF Wireless cell are configured to use the DA_SA_CTRL frame format. In addition, reliable data transfer is only supported for point-to-point communication; i.e., if a broadcast frame is transmitted using the DA_SA_CTRL frame format, the data transfer will be unreliable.

This transfer mechanism requires the recipient of an SDATA frame to provide feedback if the frame is actually received. This feedback arrives in the form of an Acknowledgement (ACK) frame. Reception of an ACK informs the transmitter that the recipient received the SDATA frame as well as whether the SDATA frame was accepted or rejected (invalid sequence number or no buffer space); it can optionally request the transmitter to delay the transmission of the next directed frame (reliable or unreliable) targeting the requestor.

If a transmitter does not receive an ACK within a certain period of time after transmitting an SDATA frame, it will automatically retransmit the frame a user-configurable number of times before it aborts the transmission. If an ACK has not been received after exhausting all retransmission attempts, the transmitting application should not assume that the target failed to receive the SDATA frame. In this instance, the user application must perform some form of polling to determine if the target actually received the frame.

When a wireless node retransmits a frame, it is possible for the target to receive multiple copies of the same frame. Therefore, ZAURA RF Wireless SDATA frames carry a sequence number and a retransmission bit that recipients use to distinguish between new and duplicate frames.

## Frame Formats

The ZAURA RF Wireless Network can be configured to use one of three different frame formats: DA, DA_SA and DA_SA_CTRL. These frame formats differ in the length of the header included in every frame. Extra header bytes reduce the effective data rate of the RF channel but allow for more complex data transfer modes (e.g., reliable data transfer). A user-configurable frame format allows the ZAURA RF Wireless Network to be deployed in a wide range of applications. For proper communications, all nodes within a ZAURA RF Wireless cell must be configured to use the same frame format.

## DA Frame Format

The simplest of the ZAURA RF Wireless frame formats is DA. Fields within the DA frame are depicted in Figure 1.

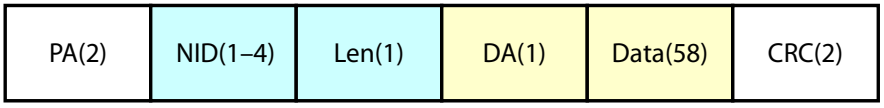


**Figure 1. Fields within the DA Frame**

Fields shown in white are configured by the ZAURA RF Wireless Library and cannot be modified by the user. The preamble (PA) is a fixed pattern of repeating 1010 bits (`0xAA`). The radio hardware uses the preamble for synchronizing the receiver's clock to the transmitter's clock. The ZAURA RF Library is configured to transmit two preamble bytes, but the radio will typically emit an extra PA as it prepares for transmission.

The CRC is calculated over the Len, DA and Data fields. The ZAURA RF Wireless Module configures the radio hardware to automatically generate the CRC on transmission and validate the CRC upon packet reception. The ZAURA Module only processes frames with a valid CRC; the radio hardware automatically rejects frames with an invalid CRC.

The ZAURA Module configures the radio to only accept packets that match the user-configurable Network ID (NID). The NID can be between 1 and 4 bytes. The longer the NID, the less susceptible the receiver will be to noise at a cost of a lower effective data rate. Zilog recommends using a NID of at least 2 bytes; this NID should not contain long sequences of repeated binary 1 or 0 digits.

The length field (Len) contains the number of bytes in the Data field. Note that the Len fields of the RF frames include the number of bytes between the Len field and the CRC. The ZAURA RF Wireless Library automatically adjusts the length field of received and transmitted packets

so that the application only has to process the length of the application data.

The DA field contains the 8-bit target address of this frame. Addresses `0x01` to `0xFE` identify unique nodes within the ZAURA RF Wireless cell. Address `0x00` is reserved and must not be used as a node address or sent in the DA or SA address of any frame. Address `0xFF` is used to broadcast data to all nodes within the same ZAURA RF Wireless cell that are in radio range of the transmitter. The ZAURA RF Wireless Module will only receive a frame targeting the broadcast address or its unique 8-bit address. Promiscuous packet reception is not supported.

The Data field contains up to 58 bytes of application data. The ZAURA RF Wireless Module does not examine or interpret the Data field during packet transmission or reception.

**Limitations and Utility of the DA Format.** The DA frame format includes the least amount of RF header bytes and will be able to achieve the highest effective data rate of any of the ZAURA RF Wireless frame formats. This frame format is most useful in applications where data flow is either unidirectional or all broadcast. In these applications, knowing the identity of the node that transmitted the frame is not required.

This data transfer mode is also useful in customer applications that use a proprietary frame format and protocol. The application is free to define additional header bytes within the ZAURA RF data fields.

Reliable ZAURA RF data services cannot be used with the DA frame format. Successful transmission of a frame using the DA frame format does not imply that the target actually received the frame; it only implies that no errors were encountered during transmission.

## DA_SA Frame Format

The only difference between the DA_SA frame format and the DA format is the addition of a Source Address (SA), as shown in Figure 2.

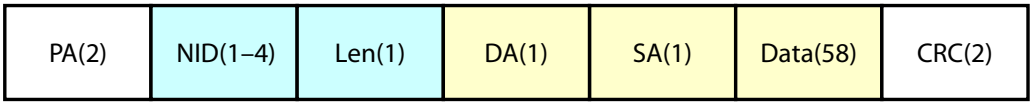


**Figure 2. Fields within the DA_SA Frame**

The ZAURA RF Wireless Library adds the SA field to all transmitted frames and makes the SA available to the application on all received packets.

**Limitations and Utility of the DA_SA Format.** The DA_SA format allows the recipient of a frame to know the identity of the node that transmitted the frame. This frame format is useful in applications that require unreliable bidirectional communications or point-to-point communication between unique nodes.

As with the DA format, the DA_SA format also allows the use of broadcast traffic (point-multipoint) and cannot be used with the ZAURA RF reliable data transfer protocol.

## DA_SA_CTRL Frame Format

The DA_SA_CTRL frame format uses a 3-byte RF header consisting of the destination address (DA), source address (SA) and the Control (CTRL) field. The DA and SA fields are identical to the corresponding fields in the DA_SA frame format, and the control field is used to identify the type of frame being transmitted as well as the target Service Access Point (SAP).

The ZAURA RF Library uses the following frame types:

**DATA.** Used to transmit application data on a best-effort basis (unreliable data transmission)

**SDATA.** Used to transmit application data using the ZAURA RF Reliable Data Transfer and Flow Control (RDTFC) protocol (reliable data transmission).

**ACK.** Used by the RDTFC to acknowledge receipt of an SDATA frame

The SAP field can be used to segregate application data into different logical streams. See the Service Access Points section on page 11 for more information.

The DA_SA_CTRL format is shown in Figure 3.

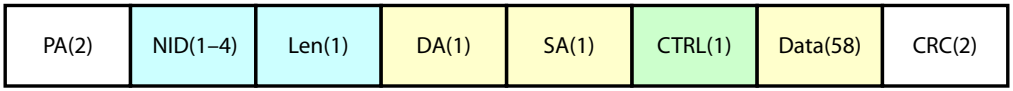


**Figure 3. Fields within the DA_SA_CTRL Frame**

This DA_SA_CTRL frame format must be used to enable ZAURA RF reliable data services. The ZAURA RF Wireless Library is configured to use this frame format by default.

**Limitations and Utility of the DA_SA_CTRL Format.** The DA_SA_CTRL format allows the transmitter to determine if the intended recipient of a packet actually received the data that was sent. This frame format also allows a simple point-to-point flow control mechanism to be implemented through the use of ACK frames.

When this frame format is used, the effective application data rate will typically be lower than that of the other frame formats because of a longer header as well as software overhead incurred with the operation of the reliable data transfer protocol.

This frame format is most useful in applications where it is more important to know whether a frame was successfully received than to transmit data as fast as possible.

## Reliable Data Transfer & Flow Control Protocol

When the ZAURA RF Wireless Library is configured to use the DA_SA_CTRL frame format, the application can choose to send unicast frames unreliably or reliably via the Reliable Data Transfer & Flow Control (RDTFC) protocol. Broadcast frames cannot be transmitted using RDTFC and will be sent unreliably even if the application requests reliable delivery.

The wireless reliable data transfer protocol cannot guarantee that a transmitted packet will be received by the target (e.g., the target is out of radio range). In some cases it is not even possible for the transmitter to determine that a target successfully received a frame (e.g., ACK lost). Even with these limitations, the RDTFC protocol ensures that the receiver will not accept duplicate frames. If the RDTFC indicates that a reliable data transfer succeeded, it guarantees the target actually received the frame.

## Service Access Points

When the ZAURA RF Library is configured to use the DA_SA_CTRL frame format, the least significant four bits of the control field of DATA and SDATA frames identify the target Service Access Point (SAP). A Service Access Point is a process that expects data in a certain format and performs a specific function on that data. Of the 16 possible SAP values, 4 are reserved for the use of the ZAURA RF Library and the remaining 12 can be arbitrarily defined by the user's application.

**Table 3. Service Access Point Labels and Functions**

| SAP | Label | Function |
|---|---|---|
| 0–11 | SAP_APP_0 to SAP_APP_11 | Defined by the application. |
| 12 | SAP_RESERVED_12 | Reserved. |
| 13 | SAP_RESERVED_13 | Reserved. |

**Table 3. Service Access Point Labels and Functions (Continued)**

| SAP | Label | Function |
|---|---|---|
| 14 | SAP_UART_0 | UART_0 Output. |
| 15 | SAP_CMD_INTRPTR | Command Interpreter Input. |

The application can use the SAP field as a simple means to segregate data streams without having to use any byte(s) from the payload. It is up to the application to implement SAP handling for receiving and transmitting SAP_APP packets. The use of the SAP field is optional; therefore, this field is safe to ignore for receiving and transmitting data.

The ZAURA RF Wireless Library defines two SAP values that applications may target when sending data: SAP_UART_0 and SAP_CMD_INTRPTR. Data sent to SAP_UART_0 will be bridged to UART_0 on the target device(s). The data field received is not interpreted by the receiving ZAURA RF Wireless Module prior to transmission on UART_0.

Data sent to the SAP_CMD_INTRPTR must be ASCII text that contains a valid console command. The string is not required to be NULL terminated. Upon receipt of the ASCII string, the remote ZAURA RF Wireless Module will process the command and send any output to `ZAURA_RF_RemoteConsoleSap` (a user-configurable parameter defaulting to SAP_UART_0) of the node that issued the remote console command.

# Channel Access Rules

The ZAURA RF Wireless Library can optionally be configured to perform a carrier sense prior to transmitting a DATA or SDATA frame to help prevent (but not eliminate) the possibility that one node initiates a transmit operation while another node is already transmitting. If more than one

node is transmitting at any given moment, it is unlikely that a receiver in range of the transmitters will be able to successfully receive any frames.

To perform a carrier sense before transmission, the ZAURA_RF_CsAttempts configuration variable must be set to a nonzero value. In this instance, the radio is placed into receive mode long enough to capture the current Receive Signal Strength Indicator (RSSI). The higher the RSSI value, the more RF energy the radio is detecting, and the more likely another station is already transmitting. If ZAURA_RF_CsAttempts is set to zero, RSSI is not sampled prior to transmission (i.e., no carrier sense is performed).

If the sampled RSSI is below ZAURA_RF_RssiThresh, the node will initiate transmission of the DATA or SDATA frame. If the RSSI value is greater than or equal to ZAURA_RF_RssiThresh, the transmitting station determines that the channel is in use. In this instance, the node will increment a ChBusy counter (initialized to 0 each time the ZAURA_RF_Transmit API is called) and will wait for a pseudo-random back-off period. After this pseudorandom delay, the node will again sample RSSI if ChBusy is less than ZARA_RF_CsAttempts. This process continues until the frame is transmitted (RSSI < ZAURA_RF_RssiThresh) or the transmission is aborted (ChBusy reaches the value of ZARA_RF_CsAttempts) and a status of ZAURA_RF_TX_CHANNEL_BUSY is returned to the caller.

The back-off period is chosen at random on an interval of 164 to 419 ticks of the external 32.768kHz oscillator corresponding to a delay of approximately 5.0ms to 12.8ms.

Prior to sending an ACK frame, the ZAURA RF node will sample RSSI if ZAURA_RF_CsAttempts is nonzero. However, unlike sending a DATA or SDATA frame, there is no random back-off, and a ChBusy counter is not used. Instead, the node attempting to send an ACK will initiate transmission as soon as RSSI < ZAURA_RF_RssiThresh. If the channel remains busy for a period of approximately 8ms, then the node will abort transmission of the ACK. If ZAURA_RF_CsAttempts is zero, then the

ACK will be transmitted without performing a carrier sense to determine if the channel is busy.

## Radio States

The ZAURA RF Module's radio features five different operating states: Sleep, Standby, Frequency Synthesis, Receive and Transmit. Because the ZAURA RF Wireless Network is half duplex, it is only possible to send data while the radio is in the Transmit state, and it is only possible to receive data while the radio is in the Receive state. Transmit and Receive states are the two active modes of operation for the radio. The remaining states are power-saving modes that can be used to reduce the current consumption of the radio when it is not required for active operation.

The radio consumes the least amount of current in the Sleep state (approximately 80µA) and the most amount of current in the Transmit state (up to 35mA). The ZAURA RF Library includes API functions to query and set the state of the radio. The Library automatically transitions the radio to the Transmit state from whatever state it is currently in when the ZAURA_RF_Transmit API is called.

The amount of time it takes to initiate a transmit operation depends on the current state of the radio. If the radio is in a very low power mode (Sleep state), it will take the ZAURA RF Library longer to initiate the transmit operation than if the radio is in a medium power state (Frequency Synthesis or Receive). Similarly, it will take longer to transition the radio to the Receive state from the Sleep state than it will from Standby state.

When the ZAURA RF Wireless Library completes a transmit operation it automatically transitions the radio to a state corresponding to the value of the ZAURA_RF_IdleState configuration variable. By default, this variable is set to ZAURA_RF_RECEIVE. However, the application can modify the value of the ZAURA_RF_IdleState configuration variable to use a different state such as ZAURA_RF_SLEEP or ZAURA_RF_STANDBY to reduce the radio's current consumption.

**Caution:** Zilog does not recommend setting ZAURA_RF_IdleState to ZAURA_RF_TRANSMIT. If the radio should remain in the ZAURA_RF_TRANSMIT state, it will emit a continuous stream of extra preambles until the radio is placed in another state. While the radio is transmitting preambles, other ZAURA RF Wireless nodes within range will sense a busy channel and will not be able to transmit.

# ZAURA RF Wireless Module Configuration

Configuration of the ZAURA RF Wireless Library is determined by the value of a set of global variables contained in the `ZAURA_RF_Conf.c` file, which must be linked to every ZAURA RF project. Customers can optionally modify the values assigned to these configuration variables to modify the default behavior of the ZAURA RF Library.

## Radio Configuration

`ZAURA_RF_Params` contains the ZAURA_RF_PARAMS structure and settings shown in Table 4.

**Table 4. ZAURA_RF_PARAMS Structure Settings**

| Structure Member | Default | Range | Meaning | Reference |
|---|---|---|---|---|
| **Network ID** | | | Contains the network identifier. All nodes that will communicate together must have the same network ID. | See ZAURA RF Wireless Cell Network Identifiers. |
| Nid.Len (Length of the ID) | 0x02 | 1–4 | | |
| Nid.Value.Data8[] | 0x11,0x22,0x00, 0x00 | 4 bytes | | |
| Addr | 0x1B | 0x01–0xFE | The address of this node. | See Node Addresses. |

**Table 4. ZAURA_RF_PARAMS Structure Settings (Continued)**

| Structure Member | Default | Range | Meaning | Reference |
|---|---|---|---|---|
| Ch | 0x01 | 868 MHz band (0–3)<br>915 MHz band (0–24) | The channel to use within the RF Frequency spectrum:<br>• The 868 MHz band has 4 channels<br>• The 915 MHz band has 25 channels | See Radio Frequencies. |
| Tx Pwr | 0x00 | 0–7 | Set the transmit power (+13 dBm to –8 dBm in increments of –3 dBm). | |
| RxFilter | 0x03 | 0x01, 0x03 | Specifies whether the broadcast packet should be passed to the application, such that:<br>• If 0x01, only accept packets directed to this node's 8-bit address.<br>• If 0x03, accept packets directed to this node's 8-bit address or to the broadcast address. | |

## Other Radio Configuration Options

Additional RF configuration variables and their default addresses are described in Table 5.

**Table 5. Additional Radio Configuration Variables**

| Variable | Default | Range | Description | Reference |
|---|---|---|---|---|
| ZAURA_RF_ Dest | RF_BC_ADDR | 0x01 to 0xFF | When issuing remote console commands or in Data Mode, the ZAURA_RF_Dest address specifies the target address of the command/data. This value is placed in the DA field of the corresponding DATA or SDATA frame. | n/a |
| ZAURA_RF_ Format | XFER_DA_SA_ CTRL | XFER_DA, XFER_DA_SA, or XFER_DA_SA_ CTRL | Defines the frame format used to communicate with ZAURA RF peers. All nodes within the same cell must be configured to use the same frame format. | See ZAURA RF Wireless Framing. |
| ZAURA_RF_ RssiThresh | 0x60 | 0x00 to 0xFF | When ZAURA_RF_CsAttempts is nonzero, ZAURA_RF_RssiThresh determines the lowest RSSI value that will cause the transmitter to defer transmitting for a random period of time between approximately 5.0ms and 12.8ms. | See Channel Access Rules. |

**Table 5. Additional Radio Configuration Variables (Continued)**

| Variable | Default | Range | Description | Reference |
|---|---|---|---|---|
| ZAURA_RF_CsAttempts | 3 | 0x00 to 0xFF | Determines if the ZAURA RF node listens for an idle channel before transmitting (Carrier Sense). If the Channel is busy (RSSI >= ZAURA_RF_RssiThresh), the transmitter will wait a random back-off period (between 0 and approximately 5.0ms and 12.8ms) and repeat the CS until succeeding or reaching ZAURA_RF_CsAttempts. If ZAURA_RF_CsAttempts is 0, the node transmits without listening for an idle channel. | See Channel Access Rules. |
| RfIdleState | RF_RECEIVE | RF_SLEEP, RF_STANDBY or RF_RECEIVE | Determines the radio state to use after completing a transmit operation. Not recommended to use a value of ZAURA_RF_TRANSMIT as the ZAURA_RF_IdleState. | See ZAURA RF Wireless Shell API Reference. |

**Table 5. Additional Radio Configuration Variables (Continued)**

| Variable | Default | Range | Description | Reference |
|---|---|---|---|---|
| ZAURA_RF_TxDelay | ZAURA_RF_ms_TO_TICKS(0) | 0x0000 to 0xFFFF | Use a non-zero value to force a transmitter to delay its next transmission by the specified number of 32.768kHz timer ticks. The ZAURA_RF_ms_TO_TICKS macro can be used to convert an integer number of milliseconds into the corresponding number of timer ticks. | See ZAURA RF Wireless Shell API Reference. |
| ZAURA_RF_MaxRetry | 3 | 0x00 to 0xFF | When sending SDATA frames the transmitter expects to receive an ACK frame from the target. If an ACK is not received within approximately 10ms the transmitter will resend the packet up to ZAURA_RF_MaxRetry times before aborting transmission of the SDATA Frame. | See Reliable Data Transfer. |

**Table 5. Additional Radio Configuration Variables (Continued)**

| Variable | Default | Range | Description | Reference |
|---|---|---|---|---|
| ZAURA_RF_ PauseUnits | 4 | 0 to 16 | Upon receipt of an SDATA frame, the recipient sends an ACK frame. If the receiver has fewer than 4 receive buffers remaining the ACK can optionally request the transmitter to delay its next data frame directed to the sender of the ACK between 0 and 16 Pause delay units. Each Pause delay unit corresponds to approximately 25 ms. | See Reliable Data Transfer. |
| ZAURA_RF_ UserParams | | | The ZAURA_RF_UserParams buffer is an arbitrary block of 128 bytes used to store application level information to nonvolatile storage. The ZAURA_RF_Demo program uses this variable to store the default wake-up message. | |

# ZAURA RF Shell Configuration Options

The five UART configuration variables are described in Table 6.

**Table 6. UART Configuration Variables**

| Variable | Default | Range | Meaning |
|---|---|---|---|
| ZAURA_RF_ DataEscChar | + | Any 8-bit value | While in Data Mode, any character received on UART0 is transmitted to SAP_APP_0 of ZAURA_RF_Dest. UART0 returns to Command Mode after two consecutive ZAURA_RF_DataEscChar bytes have been received. |
| ZAURA_RF_ UartEcho | FALSE | TRUE, FALSE | If TRUE any character received on UART 0 is transmitted (echoed) on UART 0. UART Echoing is typically enabled to allow visual confirmation of characters typed into a terminal emulator if the emulator is not configured for Local Echo. |
| ZAURA_RF_ UseUartFC | TRUE | TRUE, FALSE | When set to TRUE data transfer over UART0 implements RTS/CTS flow control. |
| ZAURA_RF_Remote ConsoleSap | SAP_UART_0 | SAP_APP_0 to SAP_APP_11 or SAP_UART_0 | Identifies the SAP to which remote console output will be sent. |
| ZAURA_RF_ UartBaudDiv[4] | {12, 6, 3, 3} | An array of 4 x 16-bit values | The ZAURA RF Wireless Library operates at 4 pre-defined frequencies of the F2480 IPO (11.06MHz, 5.53MHz, 2.76MHz and 1.38MHz). This table defines the UART baud rate divisor to use for each of the IPO settings. |

Table 7 lists two configuration options specific to the Z8F2480 MCU.

**Table 7. Z8F2480 MCU Configuration Variables**

| Variable | Default | Range | Meaning |
|---|---|---|---|
| ZAURA_RF_ OscCtrl1 | 0x82 | 0x80 = 11.06 MHz<br>0x81 = 5.53 MHz<br>0x82 = 2.76 MHz<br>0x83 = 1.38 MHz | This variable is used to determine system clock frequency (via IPO). |
| ZAURA_RF_ EnableWDT | TRUE | TRUE, FALSE | Determines whether the ZAURA RF Wireless Library enables the Z8F2480 Watchdog Timer (WDT). If this variable is set to TRUE, then after system initialization, the Z8F2480 WDT will be enabled with a default time out of approximately 4.5 seconds. In this instance, it is necessary for the application to periodically issue the eZ8 CPU's "WDT" instruction to prevent a system reset due to a WDT time out. |

# ZAURA RF Wireless API Reference

This section describes the ZAURA RF Wireless Library APIs.

## ZAURA_RF_Init

**Description** The ZAURA_RF_Init API is used to initialize the ZARA RF Wireless Library and prepare the radio for use. If the Shell Library (ZAURA_RF_Shell.lib) is linked to the project it will also be initialized by the call to ZAURA_RF_Init. This function must be called before any other ZAURA RF Library or Shell Library API function is called. The first time an application calls this function ZAURA_RF_SUCCESS is returned. Subsequent calls will return ZAURA_RF_FAILURE and will not re-initialize the library.

During the initialization process the ZAURA RF Library configures the Z8F2480 GPIO pins, initializes the Module's network and user parameters and configures the initial state of the radio.

The ZAURA RF Module is intended to be integrated with customer defined hardware platforms that may require some of the Z8F2480 GPIO pins to be configured in an application-specific manner. Therefore the ZAURA RF Library calls the user-defined ZAURA_RF_GpioConfig API to establish a platform-specific GPIO configuration. After calling ZAURA_RF_GpioConfig the Library configures the set of GPIO pins required for exclusive use of the Module and (if appropriate) the ZAURA RF Shell.

The Module.s network configuration is determined by the values assigned to the ZAURA_RF_Params structure or values stored in Flash (see ZAURA_RF_SetParams). To determine which set of parameters are used the ZAURA RF Library first checks if the parameters stored in Flash are valid (see ZAURA RF Configuration). If they are valid, then the library will modify the ZAURA_RF_Params structure to use the values from Flash. If the values in Flash are invalid, the ZAURA RF Library will use the values in the ZAURA_RF_Params structure at the time the ZAURA_RF_Init API was called (defaults specified in the `ZAURA_RF_Conf.c` file).

After the Module's network parameters have been configured, the ZAURA_RF_Init API will initialize the value of the ZAURA_RF_UserParams structure with values read from Flash (always assumed to be valid). Applications may modify the

ZAURA_RF_UserParams at run time as appropriate and should call ZAURA_RF_SetParams before the system is turned off to ensure the modified parameters are written back to Flash and available for use on subsequent power on reset events.

Upon return from this function the radio will be placed into the ZAURA_RF_IdleState which defaults to ZAURA_RF_RECEIVE.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_Init( void );

**Parameters** None.

**Returns** ZAURA_RF_SUCCESS, ZAURA_RF_FAILURE

**Example**

```
/* Initialize ZAURA RF Wireless Library */
ZAURA_RF_STATUS Status;

Status = ZAURA_RF_Init();
```

# ZAURA_RF_GpioConfig

**Description** The ZAURA RF Wireless Library calls this user-defined function during system initialization to configure the Z8F2480 MCU's GPIO port pins (as appropriate) for the particular hardware platform being used.

Because the ZAURA RF Wireless Module is a self-contained unit, the Z8F2480 MCU's pins, as used by the Module, must never be used or reconfigured by the application. Refer to the the Z8F2480 MCU Peripherals section on page 81 to determine which GPIO port pins are available to the application.

The default implementation of the ZAURA_RF_GpioConfig routine is contained in the `GpioConfig.c` file, which is located in the `.\Conf` directory of the installation folder.

**Syntax** void ZAURA_RF_GpioConfig ( void );

**Parameters** None.

**Returns** None.

**See Also** ZAURA_RF_Init

**Example**

```
void
ZAURA_RF_GpioConfig
(
   void
)
{
   /*
    * The Shell Library will reconfigure UART 0 RxD and TxD
    * during initialization. If the Shell library is not
    * used, PA4 and PA5 may be used by the application.
    */
   PAOUT = BIT5;
   PADD  = BIT4;

   PBOUT = 0;
   PBDD  = 0;
```

```
    /*
     * The ZAURA RF Library will configures these pins
     */
    //PCOUT = 0;
    //PCDD  = 0;

    PDAF  = 0;
    PDOUT = 0;
    PDDD  = 0;

    /*
     * Shell library reconfigures Uart0 RTS and CTS pins
     * during initialization. If the Shell library is not
used
     * PE3 and PE4 may be used by the application.
     */
    PEOUT = 0;
    PEDD  = BIT3;
}
```

## *ZAURA_RF_GetState*

**Description** This function retrieves the current radio state.

**Syntax** ZAURA_RF_STATE ZAURA_RF_GetState( void );

**Parameters** None.

**Returns** Current RF state.

**See Also** ZAURA_RF_SetState

**Example**

```
ZAURA_RF_STATE   State;
State = ZAURA_RF_GetState();
```

# ZAURA_RF_SetState

**Description** When the application is not required to transfer data for extended periods of time, the radio should be placed into ZAURA_RF_SLEEP state to reduce current consumption. However, it can take over 6ms to wake up the radio from sleep state. If the radio will only be idle for shorter periods of time, use the ZAURA_RF_STANDBY state. It will typically take less than 2ms to initiate a transmission or receive operation from standby state.

Because the ZAURA RF network is half duplex, it is only possible to receive data while the radio is in the ZAURA_RF_RECEIVE state. This state can be entered via an explicit call to this function or by configuring ZAURA_RF_IdleState to ZAURA_RF_RECEIVE. When ZAURA_RF_IdleState is set to ZAURA_RF_RECEIVE, the ZAURA RF Wireless Library will automatically configure the radio for reception after each transmit operation completes.

Under normal circumstances, the application should never have to explicitly set the radio to the ZAURA_RF_FREQ_SYNTH or ZAURA_RF_TRANSMIT states. The ZAURA RF Library will automatically step the radio through these states during the process of data transfer. Furthermore, if the radio is allowed to remain in the ZAURA_RF_TRANSMIT state after completing the transmission of a data frame, it will emit a continuous stream of extra preambles until the radio is placed into another state. While the radio is transmitting preambles, other ZAURA RF nodes within range will sense a busy channel and will not be able to transmit. Therefore, Zilog recommends never setting ZAURA_RF_IdleState to ZAURA_RF_TRANSMIT.

This function returns ZAURA_RF_SUCCESS when called with a valid state parameter; otherwise, ZAURA_RF_INVALID_PARAM is returned.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_SetState( ZAURA_RF_STATE State );

**Parameters** state: specifies new radio state. Permissible states are:

ZAURA_RF_SLEEP
ZAURA_RF_STANDBY
ZAURA_RF_FREQ_SYNTH
ZAURA_RF_RECEIVE
ZAURA_RF_TRANSMIT

**Returns** ZAURA_RF_SUCCESS
ZAURA_RF_INVALID_PARAM

**See Also** ZAURA_RF_GetState

**Example**

```
ZAURA_RF_STATUS   Status;
Status = ZAURA_RF_SetState( ZAURA_RF STANDBY );
```

# ZAURA_RF_Transmit

**Description** The ZAURA_RF_Transmit API sends data to a specific node (0x00 < Dest < 0xFF) or broadcasts the data to all nodes within range of the transmitter (Dest = ZAURA_RF_BC_ADDR). The data will be transferred reliably or unreliably, depending on the values of the Dest and Ctrl parameters. Data transfer will be reliable using the RDTFC if the Ctrl parameter specifies the use of an SDATA frame; the Dest parameter is not ZAURA_RF_BC_ADDR (0xFF) and the ZAURA_RF_Format configuration variable is set to XFER_DA_SA_CTRL. If any of these conditions is not satisfied, the data will be transferred unreliably. The Ctrl field is meaningless if the ZAURA_RF_Format configuration variable is set to either XFER_DA or XFER_DA_SA.

A return code of ZAURA_RF_SUCCESS means different things depending on whether RDTFC was used or whether the data was sent unreliably with a DATA frame. If the data was transported via a DATA frame, then ZAURA_RF_SUCCESS indicates that the transmitter did not encounter any errors while sending the frame. If the data was sent using an SDTA frame, ZAURA_RF_SUCCESS indicates that the peer node acknowledged and accepted the frame.

A return code of TX_CHANNEL_BUSY indicates that the data was not transmitted because the channel was in use. This status can only be returned if the ZAURA_RF_CsAttempts configuration variable is non-zero, because a zero value disables carrier sensing before transmission.

A ZAURA_RF_TX_NOT_ACK status means that the SDATA frame was successfully transmitted but no acknowledgement was received after ZAURA_RF_MaxRetry +1 attempts to transmit the SDATA frame. As a result, either the target did not receive the SDATA frame or the transmitter did not receive the ACK.

A ZAURA_RF_TX_NAK status indicates that the SDATA frame was successfully transmitted and an ACK was received. However, the ACK indicates that the target rejected the data due to a sequence error. This situation can occur if the first transmission of the SDATA frame is received and accepted by the target but the target's ACK is lost. In this instance, the transmitter will attempt to retransmit the SDATA frame and the target will reject the data as a duplicate. The ZAURA_RF_TX_NAK status could also indicate that the recipient was unable to accept the otherwise-valid SDTA frame because no buffer space was available to hold the data.

All SDATA transmit requests are synchronous, meaning control is not returned to the calling application until the frame has been transmitted and an ACK has been received, or until all retransmission attempts have failed.

All DATA transmit requests are asynchronous, meaning the ZAURA_RF_Transmit API will return control to the caller possibly before the frame has been completely sent. Therefore, the application should not explicitly change the radio state until the transmit operation completes (see ZAURA_RF_GetState on page 27). There is no need to wait for the previous DATA frame to complete transmission before calling the ZAURA_RF_Transmit API again.

**Note:** The data to be transmitted is buffered by the ZAURA RF Wireless Library before it returns control from the ZAURA_RF_Transmit API. Therefore, the application may modify the data buffer prior to completion of the actual transmission.

The ZAURA_RF.h header file also defines several macros that call the ZAURA_RF_Transmit API. These macros include:

| | |
|---|---|
| ZAURA_RF_SendData | Used to send DATA packets to SAP_APP_0. |
| ZAURA_RF_SendSeqData | Used to send SDATA packets to SAP_APP_0. |
| ZAURA_RF_SendPkt | Uses a packet buffer structure to call ZAURA_RF_Transmit. |

### Syntax

INT8 ZAURA_RF_Transmit( ZAURA_RF_ADDR Dest, UINT8 Ctrl, HANDLE hData, UINT8 Len );

### Parameters

| | |
|---|---|
| Dest | Specifies the node(s) targeted by the transmit operation. |
| Ctrl | Specifies frame type (DATA or SDATA) and target SAP. |

| | |
|---|---|
| hData | Arbitrary pointer to the data to be transmitted. |
| Len | The number of bytes of data referenced by hData (<= ZAURA_RF_MAX_DATA_LEN). |

**Returns**

ZAURA_RF_SUCCESS
ZAURA_RF_INVALID_PARAM
ZAURA_RF_TX_NOT_ACK (only applicable to SDATA frames)
ZAURA_RF_TX_NAK (only applicable to SDATA frames)
ZAURA_RF_TX_CHANNEL_BUSY

**See Also**

ZAURA_RF_SendData, ZAURA_RF_SendSeqData, ZAURA_RF_SendPkt, ZAURA_RF_Receive, ZAURA_RF_PKT_BUF Structure

**Example**

```
ZAURA_RF_STATUS   Status;

// Send a broadcast (unreliable data transfer) to
SAP_APP_0 on all remote nodes
Status = ZAURA_RF_Transmit( ZAURA_RF_BC_ADDR, DATA |
SAP_APP_0, "Hello" , 5 );

// Send a message to SAP_APP_5 on node 0x23 using the
ZAURA RDTFC protocol
Status = ZAURA_RF_Transmit( 0x23, SDATA | SAP_APP_5,
"Hello" , 5 );
```

# ZAURA_RF_SendData

**Description** The ZAURA_RF_SendData macro sends data (using unreliable data transfer) to SAP_APP_0 on remote node(s).

**Syntax** ZAURA_RF_STATUS
ZAURA_RF_SendData(ZAURA_ZAURA_RF_ADDR Dest, HANDLE hData, UINT8 Len)

**Parameters**

| | |
|---|---|
| Dest | ZAURA_ZAURA_RF_ADDR value for the destination address. Addresses `0x01` through `0xFE` are specific nodes (point-point), `0xFF` is a broadcast address (point-multipoint). |
| hData | An arbitrary pointer to the application data. |
| Len | The number of bytes referenced by hData. |

**Returns** ZAURA_RF_SUCCESS is returned if the transmissions did not encounter any errors. This only means it was transmitted, not that it was received.

TX_CHANNEL_BUSY is returned if the channel was in use. (See Channel Access Rules for more information).

**See Also** ZAURA_RF_Transmit, ZAURA_RF_SendSeqData, ZAURA_RF_SendPkt, ZAURA_RF_Receive, ZAURA_RF_PKT_BUF Structure

**Example 1**

```
ZAURA_RF_SendData( ZAURA_RF_BC_ADDR, "Hello world", 11 );
```

**Example 2**

```
UINT16 Data = 0x1234;
ZAURA_RF_SendData( ZAURA_RF_Dest, &Data, 2 );
```

# ZAURA_RF_SendSeqData

**Description** The ZAURA_RF_SendSeqData macro sends data (using RDTFC) to SAP_APP_0 on a remote node.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_SendSeqData(
ZAURA_RF_ADDR Dest, HANDLE hData, UINT8 Len)

**Parameters**

| | |
|---|---|
| Dest | ZAURA_RF_ADDR value for the destination address. Addresses `0x01` through `0xFE` are specific nodes (point-to-point). |
| hData | A void pointer to the application byte data. |
| Len | UINT8 value specifies the length of the valid bytes referenced by hData. |

## Returns

ZAURA_RF_SUCCESS is returned if the peer node acknowledged and accepted the frame.

TX_CHANNEL_BUSY is returned if the channel was in use (see Channel Access Rules for more information).

ZAURA_RF_TX_NOT_ACK is returned if no acknowledgement was received. This return result could either be that the packet did not arrive at its destination or the ACK was not received.

RF_TX_NAK is returned if the node received the frame but rejected the frame because no buffer space was available to hold the data.

**See Also** ZAURA_RF_Transmit, ZAURA_RF_SendData, ZAURA_RF_SendPkt, ZAURA_RF_Receive, ZAURA_RF_PKT_BUF Structure

**Example**

```
/* Send reliable data to node */
ZAURA_RF_SendSeqData( 0x23, "Hello world", 11 );
```

# ZAURA_RF_SendPkt

**Description** The RfSendPkt macro sends data to remote node(s) using a packet buffer structure. This packet buffer structure can be obtained from the ZAURA_RF_Receive API or defined within application memory space. The calling application retains control of the packet buffer after this function returns. The data will be transferred reliably or unreliably, depending on the values of the Dst and Ctrl members of the packet buffer structure (see ZAURA_RF_Transmit on page 30 for more information).

**Syntax**

```
ZAURA_RF_STATUS ZAURA_RF_SendPkt(
ZAURA_RF_PKT_BUF * pBuf)
```

**Parameters**

| | |
|---|---|
| pBuf | A pointer to a packet buffer structure that contains the information to be transmitted. |

**Returns** The return value depends on the type of packet that was sent.

For unreliable DATA transmissions (including broadcast):

- ZAURA_RF_SUCCESS is returned if the transmissions did not encounter any errors. This only means it was transmitted, not that it was received
- TX_CHANNEL_BUSY is returned if the channel was in use. (See Channel Access Rules for more information)

For Reliable SDATA transmissions:

- ZAURA_RF_SUCCESS is returned if the peer node acknowledged and accepted the frame
- TX_CHANNEL_BUSY is returned if the channel was in use (See Channel Access Rules for more information)
- ZAURA_RF_TX_NOT_ACK is returned if no acknowledgement was received. This could be either the packet did not arrive at destination or the ACK was note received
- RF_TX_NAK is returned if the node received the frame but rejected the frame because no buffer space was available to hold the data

**See Also** ZAURA_RF_Transmit, ZAURA_RF_SendSeqData, ZAURA_RF_SendData, ZAURA_RF_Receive, ZAURA_RF_PKT_BUF Structure

**Example**

```
pPkt = RfReceive();
if( pPkt != NULLPTR )
{
   /*
    * Forward this packet to SAP_APP_1 on ZAURA_RF_Dest
using RDTFC
    */
   pPkt->Dst = ZAURA_RF_Dest;
   pPkt->Ctrl = SDATA | SAP_APP_1;
   ZAURA_RF_SendPkt( pPkt );


   ZAURA_RF_FreeBuf( pBuf );
}
```

## *ZAURA_RF_Receive*

**Description** When the ZAURA RF Module receives a frame, the ZAURA RF Wireless Library attempts to allocate a packet buffer structure to hold the packet. The allocated packet buffer is then added to a queue of received packets waiting for application processing. To retrieve the next packet from the queue, an application must call the ZAURA_RF_Receive API. If the receive packet queue is empty, this function returns NULLPTR; otherwise it returns a pointer to a packet buffer structure containing the data to be processed. After the application has finished processing the received packet, it must call ZAURA_RF_FreeBuf to pass the packet buffer pointer obtained from this API as an argument.

**Syntax** ZAURA_RF_PKT_BUF * ZAURA_RF_Receive( void )

**Parameters** This function does not contain any parameters.

**Returns** A pointer to a ZAURA_RF_PKT_BUF structure. If there are no receive packets available, a NULL pointer will be returned.

**See Also** ZAURA_RF_FreeBuf, ZAURA_RF_PKT_BUF Structure

**Example**

```
ZAURA_RF_PKT_BUF * pPkt;

pPkt= ZAURA_RF_Receive();
if( pPkt )
{
   /*
    * Process the packet here
    */
   ZAURA_RF_FreeBuf();
}
```

## *ZAURA_RF_FreeBuf*

**Description** The ZAURA_RF_FreeBuf API is called to return control of a packet buffer to the ZAURA RF Wireless Library after the application has finished processing the packet. An application acquires control of a packet buffer structure when the ZAURA_RF_Receive API returns a nonzero value. If an application fails to call ZAURA_RF_FreeBuf after processing a received packet, the ZAURA RF Library will eventually run out of packet buffers, thereby preventing subsequent data transfers.

**Syntax** void ZAURA_RF_FreeBuf(ZAURA_RF_PKT_BUF * pPkt)

**Parameters**

| | |
|---|---|
| pPkt | A pointer to the packet buffer to be released back into the packet buffer pool. |

**Returns** None.

**See Also** ZAURA_RF_Receive, ZAURA_RF_PKT_BUF Structure

**Example** See ZAURA_RF_Receive example.

## ZAURA_RF_ReadRSSI

**Description** This function stores the current Receive Signal Strength Indicator (RSSI) in the ZAURA_RF_RSSI global variable and returns the value of the global variable to the caller. The RSSI value returned is an 8-bit quantity that can range from `0x00` to `0xFF`. Typically, the RSSI value varies over a narrower range.

**Note:** RSSI can only be sampled if the radio is in the ZAURA_RF_RECEIVE state. If the radio is not in the ZAURA_RF_RECEIVE state when this function is called, a value of `0xFF` is returned.

The RSSI can be used as a relative indicator of the amount of RF energy the receiver is detecting. A higher value indicates the presence of more energy (i.e., close to the transmitter) and a lower value indicates only the presence of noise (no signal being transmitted). The RSSI value returned can be passed to the ZAURA_RF_Rssi2Pwr function to estimate the signal strength in dBm.

Because the ZAURA RF Wireless Library can optionally be configured to perform carrier sensing (via RSSI sampling) prior to transmitting data frames, there is typically no need for the application to call this function before calling the ZAURA_RF_Transmit routine. This function is most useful while scanning ZAURA RF channels to determine which channels are in use.

**Syntax** UINT8 ZAURA_RF_ReadRSSI( void );

**Parameters** None.

**Returns** The current RSSI value.

**See Also** ZAURA_RF_Rssi2Pwr, ZAURA_RF_GetState, ZAURA_RF_SetState

**Example**

```
UINT8   CurRssi;
```

```
CurRssi = ZAURA_RF_ReadRSSI();
```

## *ZAURA_RF_Rssi2Pwr*

**Description** This function returns a signed 8-bit value representing the (approximate) power level (in dBm) of the RSSI parameter.

**Syntax** INT8 ZAURA_RF_Rssi2Pwr(UINT8 RSSIvalue)

**Parameters**

| | |
|---|---|
| RSSIvalue | RSSI 8-bit value, typically returned from ZAURA_RF_ReadRSSI() function. |

**Returns** Returns a signed 8-bit value representing the approximate power level in dBm of the received signal strength.

**See Also** ZAURA_RF_ReadRSSI, ZAURA_RF_GetState, ZAURA_RF_SetState

**Example**

```
INT8     Pwr;
UINT8   CurRssi;

CurRssi = ZAURA_RF_ReadRSSI();
Pwr      = ZAURA_RF_Rssi2Pwr( CurRssi );
```

# ZAURA RF Wireless Radio Configuration API

This section of the API allows developers to configure the radio parameters programmatically at run time to override the compile time configuration defined in the `ZAURA_RF_Conf.c` file. Changes made to the RF parameters at run time as a result of calling APIs in this section are not saved to nonvolatile storage (Flash) unless the application calls the ZAURA_RF_SetParams API (see the ZAURA RF Wireless Module Configuration section on page 15).

# ZAURA_RF_GetParams

**Description** This function reads the RF and User parameters stored in Flash.

If the Flash-based RF parameters are valid, the RAM-based ZAURA_RF_Params configuration variable is updated with the values read from Flash. If the Flash-based RF parameters are invalid, the RAM-based ZAURA_RF_Params configuration variable is not modified. Prior to returning to the caller, this function will configure the RF Module with values stored in the RAM-based ZAURA_RF_Params configuration variable.

The User parameters read from Flash are always assumed to be valid and will overwrite the contents of the RAM-based ZAURA_RF_UserParams configuration variable.

The ZAURA RF Wireless Library calls this function during initialization to determine whether the RF parameters from Flash or the default values specified in the ZAURA RF configuration file will be used to initialize the radio.

**Syntax** void ZAURA_RF_GetParams( void );

**Parameters** None.

**Returns** None.

**See Also** ZAURA_RF_SetParams, ZAURA_RF_PARAMS Structure, ZAURA RF Wireless Module Configuration

**Example**

```
ZAURA_RF_GetParams();
/*
 * If the RF parameters stored in Flash were valid, the
 * ZAURA_RF_Params global variable now contains the values
 * stored in Flash and the radio has been reprogrammed to
 * use the RF parameters stored in Flash.
 *
 * In addition the RAM-based ZAURA_RF_UsserParams global
 * variable will be overwritten with the values from
 * Flash.
 **/
```

# ZAURA_RF_SetParams

**Description** This routine writes the contents of the ZAURA_RF_Params and ZAURA_RF_UserParams global variables to Flash. The ZAURA RF Wireless Library also writes private configuration information to Flash when this API is called.

Whenever any of the RF parameters is modified by a shell command, the ZAURA RF Library calls this function to update the corresponding parameter in Flash. If the application uses a Library API function to modify the RF parameters, the application should call this routine to ensure that the modified parameter(s) are written to Flash and will be used to configure the radio on the next power-on reset event.

**Syntax** void ZAURA_RF_SetParams( void );

**Parameters** None.

**Returns** None.

**See Also** ZAURA_RF_GetParams, ZAURA_RF_PARAMS Structure, ZAURA_RF_UserParams description in Table 5.

**Example**

```
/*
 * Change this node's RF address to 0x23 and store the new
 * RF configuration in Flash.
 */
ZAURA_RF_GetParams();
ZAURA_RF_Params.Addr = 0x23;
ZAURA_RF_SetParams();
```

## *ZAURA_RF_GetAddr*

**Description** This function returns the ZAURA RF node address currently programmed into the radio. The default node address is specified in the Addr member of the ZAURA_RF_Params structure. The address can be modified using the ZAURA_RF_SetAddr function or the `addr` shell command.

**Syntax** ZAURA_RF_ADDR ZAURA_RF_GetAddr( void );

**Parameters** None.

**Returns** The 8-bit node address.

**See Also** ZAURA_RF_SetAddr, ZAURA_RF_GetParams, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_ADDR   Addr;
Addr = ZAURA_RF_GetAddr();
```

# ZAURA_RF_SetAddr

**Description** This function is used to modify the ZAURA RF node address used by the radio. If the node address is in the range of `0x01` to `0xFE`, the radio is reconfigured to use the new address, the ZAURA_RF_Params structure is updated and ZAURA_RF_SUCCESS is returned. Otherwise, ZAURA_RF_INVALID_PARAM is returned and the node address is not modified.

The ZAURA RF Wireless Library does not verify the uniqueness of the address. The application designer is responsible for assigning unique addresses to all nodes within the ZAURA RF cell.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_SetAddr( ZAURA_RF_ADDR Addr );

**Parameters** Addr: The 8-bit node address.

**Returns** ZAURA_RF_SUCCESS
ZAURA_RF_INVALID_PARAM

**See Also** ZAURA_RF_GetAddr, ZAURA_RF_SetParams, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_STATUS   Status;
Status = ZAURA_RF_SetAddr( 0x23 );
```

# *ZAURA_RF_GetNID*

**Description** This function retrieves the ZAURA RF Network ID (NID) currently programmed into the radio. All nodes within the same ZAURA RF cell must use the same NID (and channel) to communicate with each another. The default NID is specified in the Nid member of the ZAURA_RF_Params structure. The Nid can be modified using the ZAURA_RF_SetNID function or the `nid` console command.

**Syntax** void ZAURA_RF_GetNID( RF_NID * pNid );

**Parameters**

| | |
|---|---|
| pNid | A pointer to a Network ID structure initialized with the current NID. |

**Returns** None.

**See Also** ZAURA_RF_SetNID, ZAURA_RF_GetParams, ZAURA_RF_NID Structure, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_NID   CurNid;
ZAURA_RF_GetNid( &CurNid );
```

# ZAURA_RF_SetNID

**Description** This function is used to modify the ZAURA RF Network ID (NID) used by the radio. All nodes within the same ZAURA RF cell must use the same NID (and channel) to communication with one another. The NID is an arbitrary value between 1 and 4 bytes in length. If the NID is of the proper length, the radio is reconfigured to used the new NID, the ZAURA_RF_Params structure is updated and ZAURA_RF_SUCCESS is returned. Otherwise, ZAURA_RF_INVALID_PARAM is returned and the current NID is not modified.

To improve the radio's immunity to noise, the caller should avoid using NID values that contain long sequences of repeated binary 1 or 0 digits. For example, the NID value `0x00000001` should be avoided because a single pulse of RF energy near the channel centre frequency could be misinterpreted as a valid NID.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_SetNID( RF_NID * pNid );

**Parameters**

| | |
|---|---|
| pNid | Points to a new NID. |

**Returns** ZAURA_RF_SUCCESS, ZAURA_RF_INVALID_PARAM

**See Also** ZAURA_RF_GetNID, ZAURA_RF_SetParams, ZAURA_RF_NID Structure, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_NID   MyNid;
ZAURA_RF_STATUS;

MyNid.Len = 2;
MyNid.Value.Data8[0] = 0x11;
MyNid.Value.Data8[1] = 0x22;

Status = ZAURA_RF_SetNid( &MyNid );
```

## ZAURA_RF_GetRxFilter

**Description** This function retrieves the receiver filter setting currently programmed into the radio. The receive filter is used to decide whether an inbound frame should be accepted or silently discarded at the hardware level. The default receive filter setting is specified in the RxFilter member of the ZAURA_RF_Params structure. The filter setting can be modified using the ZAURA_RF_SetRxFilter function or the `filter` shell command.

See ZAURA_RF_SetRxFilter on page 51 to review the definitions of the filter flags.

**Syntax** UINT8 ZAURA_RF_GetRxFilter( void );

**Parameters** None.

**Returns** Current Filter Setting (0 to 3)

**See Also** ZAURA_RF_SetRxFilter, ZAURA_RF_GetParams, ZAURA_RF_PARAMS Structure

**Example**

```
UINT8   RxFilter;
RxFilter = ZAURA_RF_GetRxFilter();
```

# ZAURA_RF_SetRxFilter

**Description** This function modifies the filter setting that the radio uses when deciding to accept inbound frames. Only the bottom 2 bits of the Filter argument are processed to ensure that the new filter setting is always between 0 and 3. After calling this function, the new filter setting is programmed into the radio and the ZAURA_RF_Params structure is updated with the new filter setting.

When using the ZAURA RF RDTFC protocol, the Filter parameter should be set to either 1 or 3. In particular, the use of promiscuous mode (Filter=0) is not supported because it will defeat the RDTFC protocol. Nodes using a filter setting less than 3 will not be able to receive broadcasts.

**Syntax** void ZAURA_RF_SetRxFilter( UINT8 Filter );

**Parameters** Filter. The filter settings are:

0 = Any address (promiscuous).

1 = Node Address.

2 = Node Address + 0x00 reserved address.

3 = Node Address + 0x00 reserved address + 0xFF broadcast address (ZAURA_RF_BC_ADDR).

**Returns** ZAURA_RF_SUCCESS, ZAURA_RF_INVALID_PARAM

**See Also** ZAURA_RF_GetRxFilter, ZAURA_RF_SetParams, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_SetFilter( 3 );
```

## ZAURA_RF_GetChannel

**Description** Returns the 0-based channel number that the radio is currently using for communication. The default RF channel is specified in the Ch member of the ZAURA_RF_Params structure. The RF channel can be modified using ZAURA_RF_SetChannel function or the `ch` shell command.

**Syntax** UINT8 ZAURA_RF_GetChannel( void );

**Parameters** None.

**Returns** Current channel setting (0 to the maximum number of channels – 1)

**See Also** ZAURA_RF_SetChannel, ZAURA_RF_GetParams, ZAURA_RF_PARAMS Structure

**Example**

```
UINT8   CurCh;
CurCh = ZAURA_RF_GetChannel();
```

# ZAURA_RF_SetChannel

**Description** The Channel parameter must be between 0 and the maximum number of channels minus 1. The value of MaxChannels depends on which version of the ZAURA RF Wireless Library is linked to the project. The ZAURA RF_866p5_MHz library uses 4 channels (i.e., the channel parameter must be between 0 and 3). The ZAURA RF_915_MHz library uses 25 channels (i.e., the channel parameter must be between 0 and 24). For specific channel frequencies, see the Radio Frequencies section on page 3.

If the channel parameter is valid, the radio configuration is modified, the new channel number is stored in the `Ch` member of the ZAURA_RF_Params structure and ZAURA_RF_SUCCESS is returned. If the Channel number is invalid, ZAURA_RF_INVALID_PARAM is returned and the current channel is not modified.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_SetChannel( UINT8 Channel );

**Parameters** Channel: The new channel on which to communicate.

**Returns** ZAURA_RF_SUCCESS, ZAURA_RF_INVALID_PARAM

**See Also** ZAURA_RF_GetChannel, ZAURA_RF_SetParams, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_STATUS   Status;
Status = ZAURA_RF_SetChannel( 1 );
```

## ZAURA_RF_GetTxPwr

**Description** This function returns a 3-bit value indicating the current transmit power level setting of the radio. The default transmit power level is specified in the TxPwr member of the ZAURA_RF_Params structure. The transmit power level can be modified using ZAURA_RF_SetTxPwr function or the `pwr` shell command.

**Syntax** UINT8 ZAURA_RF_GetTxPwr( void );

**Parameters** None.

**Returns** Power level setting (between 0 and 7) used during transmission.

**See Also** ZAURA_RF_SetTxPwr, ZAURA_RF_GetParams, ZAURA_RF_PARAMS Structure

**Example**

```
UINT8   CurPwr;
CurPwr = ZAURA_RF_GetTxPwr();
```

# ZAURA_RF_SetTxPwr

**Description** This function modifies the power level at which the radio transmits. If the Pwr parameter is between 0 and 7, the radio configuration is modified, the new power level is stored in the ZAURA_RF_Params structure and ZAURA_RF_SUCCESS is returned. If the power level is invalid, ZAURA_RF_INVALID_PARAM is returned and the current radio configuration is not modified.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_SetTxPwr( UINT8 Pwr );

**Parameters** Pwr: Specifies the new transmit power level setting. Valid settings are:

| | |
|---|---|
| 0 | +13dBm |
| 1 | +10dBm |
| 2 | +7dBm |
| 3 | +4dBm |
| 4 | +1dBm |
| 5 | –2dBm |
| 6 | –5dBm |
| 7 | –8dBm |

**Returns** ZAURA_RF_SUCCESS, ZAURA_RF_INVALID_PARAM

**See Also** ZAURA_RF_GetTxPwr, ZAURA_RF_SetParams, ZAURA_RF_PARAMS Structure

**Example**

```
ZAURA_RF_STATUS   Status;
Status = ZAURA_RF_SetTxPwr( 0 );
```

# Variable Types and Structures

This section describes the types of RF Wireless Library configuration variables featured in the ZAURA RF Wireless modules, as well as their data containment structures. For more information about the variables themselves, see the ZAURA RF Wireless Module Configuration section on page 15.

## Types

Table 8 describes the types of variables employed by the ZAURA RF Wireless Configuration API.

**Table 8. Types of Variables**

| Name | Description |
|---|---|
| UINT8 | Unsigned 8-bit integer |
| INT8 | Signed 8-bit integer |
| UINT16 | Unsigned 16-bit integer |
| UINT32 | Unsigned 32-bit integer |
| HANDLE | void pointer (void *) |
| ZAURA_RF_ADDR | UINT8 |
| ZAURA_RF_STATUS | Signed 8-bit integer |
| ZAURA_RF_STATE | Enumeration of state types |

## Structures

The ZAURA RF Wireless Library employs four types of data structures: ZAURA_RF_PKT_BUF, ZAURA_RF_NID, ZAURA_RF_PARAMS and ZAURA_RF_STATS. Each of these structures is described in this section.

## ZAURA_RF_PKT_BUF Structure

The ZAURA_RF_PKT_BUF structure is used to hold a packet for processing, receive or transmit. It contains the structure types indicated in Table 9.

**Table 9. ZAURA_RF_PKT_BUF Structure Types**

| Member | Type | Description |
|---|---|---|
| Len | UINT8 | Length of the application data packet. |
| Dst | ZAURA_RF_ADDR | Destination. |
| Src | ZAURA_RF_ADDR | Source (valid on DA_SA, DA_SA_CTRL frames). |
| Ctrl | UINT8 | Control byte (valid on DA_SA_CTRL frames). |
| Data[] | UINT8 | Array of application-specific data bytes in the packet. |

## ZAURA_RF_NID Structure

The ZAURA_RF_NID structure is used to hold the Network ID information. It contains the structure types indicated in Table 10.

**Table 10. ZAURA_RF_NID Structure Types**

| Member | Type | Description |
|---|---|---|
| Len | UINT8 | The number of bytes that are valid in Data. |
| Value | Union | The Network ID byte(s). |

In the following code snippet, the first column represents each Union type; the second column represents its value.

```
{
 UINT8      Data8[4]
```

```
 UINT16     Data16[2]
 UINT32     Data32
}
```

## ZAURA_RF_PARAMS Structure

The ZAURA_RF_PARAMS structure is used to hold the configuration parameters for the radio. It contains the structure types indicated in Table 11.

**Table 11. ZAURA_RF_PARAMS Structure Type**

| Member | Type | Description |
|---|---|---|
| Nid | ZAURA_RF_NID | Network ID information. |
| Addr | ZAURA_RF_ADDR | Local address of this node. |
| Ch | UINT8 | Current Channel. |
| TxPwr | UINT8 | Transmit Power Level. |
| RxFilter | UINT8 | Receive filter. |

## ZAURA_RF_STATS Structure

The ZAURA_RF_STATS structure is used to hold the statistics for the radio. These stats can be accessed through the global variable `ZAURA_RF_Stats`. The ZAURA_RF_STATS structure contains the structure types indicated in Table 12.

**Table 12. ZAURA_RF_STATS Structure Types**

| Member | Type | Description |
|---|---|---|
| RxPkts; | UINT16 | Total number of packets received. |
| RxBytes; | UINT32 | Total bytes received. |
| RxNoBuf; | UINT16 | Number of buffers. |

**Table 12. ZAURA_RF_STATS Structure Types (Continued)**

| Member | Type | Description |
|---|---|---|
| RxAck; | UINT16 | Total received Acknowledgments. |
| RxNak; | UINT16 | Total received Negative ACKs. |
| RxPause; | UINT16 | Total number of pauses requests received. |
| RxRetry; | UINT16 | Total number of Retries received. |
| TxPkts; | UINT16 | Total packets transmitted. |
| TxBytes; | UINT16 | Total bytes transmitted. |
| TxUR; | UINT16 | Transmit Underruns issued. |
| TxBusy; | UINT16 | Number of failed transmit attempts due to a busy RF channel. |
| TxAck; | UINT16 | Total Acknowledgments issued. |
| TxNak; | UINT16 | Total Negative Acknowledgements issued. |
| TxRetry; | UINT16 | Total number of retries issued. |
| TxPause; | UINT16 | Total number of pause requests issued. |

# ZAURA RF Wireless Shell API Reference

The ZAURA RF Wireless Library includes a configurable interactive command interpreter (shell), using UART 0 as the console. The Shell Library provides a base set of commands and allows you to add optional commands from the Shell Library or add user-defined shell commands. The Shell also provides the ability to set up remote console support (similar to TelNet), allowing you to control a remote ZAURA RF node. Console commands received by the local ZAURA RF Shell are sent wirelessly to remote node(s) for processing. Console output from the operation of the shell command on the remote node(s) is transmitted back to the originating ZAURA RF node and displayed on the local console.

The Shell is an optional component in ZAURA RF Wireless projects. If your application requires the use of the Shell, the project must link to the `ZAURA_RF_Shell.lib` library. If your project does not require the use of the Shell, your application must not call any shell function. In this instance, your project must link to the `No_Shell.lib` library.

## Configuring ZAURA RF Shell Commands

The ZAURA RF Module Shell User Manual (UM0235) describes the default set of shell commands and all predefined shell commands that can optionally be added to ZAURA RF applications. To add any nondefault command to the ZAURA RF Shell, use the ZAURA_RF_ShellAddCmd API to pass the ASCII name assigned to the command as well as a function pointer to the routine that implements the command. The ZAURA_RF_ShellAddCmd API is used regardless of whether the command being added to the Shell is an optional ZAURA RF shell command or a user-defined shell command.

# Creating User-Defined ZAURA RF Shell Commands

The ZAURA RF Shell Library provides a set of utility functions and global variables that applications can use to build custom shell commands. To implement a user-defined shell command, it is necessary to understand how the ZAURA RF Shell Library processes ASCII text from the console.

As a user enters characters on the console, the ZAURA RF Shell Library buffers these characters and increments a global counter to track the number of characters entered (ZAURA_RF_UartAvail). After the user presses the Enter key, the Shell Library detects a carriage return in the console input stream and sets the ZAURA_RF_UartEOL flag to TRUE to indicate that a command line has been received.

Applications that use the ZAURA RF Shell must monitor the ZAURA_RF_UartEOL flag and call ZAURA_RF_ShellProcessCmdLine at noninterrupt time to allow the Shell Library to interpret the command just entered.

**Note:** After the ZAURA_RF_UartEOL flag is set to TRUE, the Shell Library stops buffering console input. Therefore, applications should call ZAURA_RF_ShellProcessCmdLine soon after this flag is set to avoid lost console input. Alternatively, the console program can be configured to insert delays between command lines.

The ZAURA_RF_ShellProcessCmdLine function parses the console input into space-delimited tokens that are referenced by the pZAURA_RF_Tokens array. The first token (at index 0 in pZAURA_RF_Tokens array) is always the name of the console command, and the remaining tokens (if any) point to user-supplied parameters.

**Example.** If the user enters `test 1 2 3` on the console, the ZAURA RF Shell will recognize 4 tokens. The first token contains the value `test` and

the remaining three tokens reference the parameters `1`, `2` and `3`, respectively. If a parameter must contain spaces, enclose the parameter within spaces. The command line `test "this is one token" 2 3` also contains 4 tokens, with the first parameter being `this is one token`.

After parsing the command line into tokens, ZAURA_RF_ProcessCmdLine API compares the name command submitted (referenced by pZAURA_RF_Tokens[0]) to the current set of Shell commands. If a match is found, the corresponding function pointer is called. Otherwise, an error message is displayed on the console.

If the entered shell command corresponds to a user-defined function, then the application is responsible for processing the command. Exactly how the application processes this command is application-dependent. The remainder of this section describes the ZAURA RF Shell global variables and API functions available to user-defined shell commands.

## Console Global Variables

Table 13 describes the types of console variables employed by the ZAURA RF Wireless Shell API.

**Table 13. Console Global Variables**

| Variable Name | Type | Description |
|---|---|---|
| ZAURA_RF_NumTokens | UINT8 | Used by shell routines to determine the number of space-separated strings (i.e., tokens) within the console command. For example, the command line `MyCmd has 3 parameters` contains 4 tokens. If a token must include spaces, enclose it in quotation marks. The line `This is a single token` contains just one token. The maximum number of tokens that can be supported is ZAURA_RF_SHELL_MAX_TOKENS (currently defined as 5). |
| pZAURA_RF_Token[MAX_TOKENS] | char * | An array of pointers referencing the NULL-terminated string of characters comprising the token. |

**Table 13. Console Global Variables**

| Variable Name | Type | Description |
|---|---|---|
| ZAURA_RF_UartEOL | UINT8 | This flag is set to TRUE after the Shell Library detects a carriage return received on UART 0. Applications can use this flag to decide when to call the ZAURA_RF_ShellProcessCmdLine API. |
| ZAURA_RF_UartAvail | UINT8 | Yields the number of characters in the console buffer waiting to be processed. |

# ZAURA RF Wireless Shell API Descriptions

The section that follows describes the console utility functions employed by the ZAURA RF Wireless Shell API.

## *ZAURA_RF_ProcessCommandLine()*

**Description** The ZAURA_RF_ProcessCommandLine API handles the processing the console input after a user has pressed the carriage return. An application should call this routine shortly after the ZAURA_RF_UartEOL flag is set to TRUE, which indicates that the console operator has pressed the Enter key. The application is free to defer processing of the command line until more important operations have completed.

**Syntax** void ProcessCommandLine( void )

**Parameters** This function does not have any parameters.

**Returns** None.

**Example**

```
/*
 * Wait for End of Line character before processing
 * console commands.
 */
if( ZAURA_RF_UartEOL )
{
   ZAURA_RF_ShellProcessCmdLine();
}
```

## ZAURA_RF_ShellAddCmd

**Description** The ZAURA_RF_ShellAddCmd function is used to add a nondefault command to the ZAURA RF Shell. The Shell can accommodate a maximum of 30 commands, of which 7 are reserved to hold the default set of shell commands. Consequently, 23 slots are available for adding optional ZAURA RF shell commands or user-defined commands.

**Syntax** ZAURA_RF_STATUS ZAURA_RF_ShellAddCmd( char * pName, SHELL_CMD pFunc)

**Parameters**

| | |
|---|---|
| pName | A pointer to a null-terminated string containing the token for the user to enter into the console to identify this command. |
| pFunc | The name of the function that implements the shell command. The function prototype must be:<br>void pFunc( void ); |

The available Library Shell commands are described in Table 14.

**Table 14. Library Shell Commands**

| Shell Command (pName) | Function Pointer (pFunc) | Use For |
|---|---|---|
| addr | ZAURA_RF_ShellCmdAddr | Get/set the local ZAURA RF Wireless Module station address. |
| ch | ZAURA_RF_ShellCmdChannel | Get/set the current ZAURA RF Wireless Module channel. |
| data | ZAURA_RF_ShellCmdData | Enter Data Mode (exits local console mode). |
| dst | ZAURA_RF_ShellCmdDst | Get/set default remote target address. |
| echo | ZAURA_RF_ShellCmdEcho | Initiates ping-pong test between 2 nodes. |
| filter | ZAURA_RF_ShellCmdFilter | Get/set the ZAURA RF Wireless Module receive filter. |

**Table 14. Library Shell Commands**

| Shell Command (pName) | Function Pointer (pFunc) | Use For |
|---|---|---|
| ipo | ZAURA_RF_ShellCmdIpo | Get/set system Internal Precision Oscillator frequency. |
| nid | ZAURA_RF_ShellCmdNid | Get/set the ZAURA RF Wireless Module Network Identifier. |
| pa | ZAURA_RF_ShellCmdPa | Initiates transmission of preamble pattern. |
| per | ZAURA_RF_ShellCmdPer | Initiate transmission of a burst of 100 packets (use with peer running 'rx' command). |
| port | ZAURA_RF_ShellCmdPort | Modifies a GPIO port output pin. |
| pwr | ZAURA_RF_ShellCmdPwr | Get/set the ZAURA RF Wireless Module transmit power level. |
| reboot | ZAURA_RF_ShellSoftReset | Initiates POR reset. |
| rssi | ZAURA_RF_ShellCmdRssi | Displays RSSI of the current RF channel. |
| rx | ZAURA_RF_ShellCmdRx | Places the radio in receive mode waiting for a burst of 100 packets from remote peer running 'per' command. |
| sleep | ZAURA_RF_ShellCmdSleep | Places the ZAURA RF Wireless Module node in a low power mode until a console character is received or a push button activated. |
| stats | ZAURA_RF_ShellCmdStats | Displays packet level statistics. |
| tx | ZAURA_RF_ShellCmdTx | Transmits one or more data packets. |
| uecho | ZAURA_RF_ShellCmdZAURA_RF_UartEcho | Enable/disable local echoing of console input. |

**Returns** If there was room to add the command, then ZAURA_RF_SUCCESS is returned; otherwise, ZAURA_RF_FAILURE is returned.

**Example**

```
void
```

```
InitShell(void)
{
 ZAURA_RF_ShellAddCmd( "addr",  ZAURA_RF_ShellCmdAddr );
 ZAURA_RF_ShellAddCmd( "ch",    ZAURA_RF_ShellCmdChannel );
 ZAURA_RF_ShellAddCmd( "data",  ZAURA_RF_ShellCmdData );
 ZAURA_RF_ShellAddCmd( "dst",   ZAURA_RF_ShellCmdDst );
 ZAURA_RF_ShellAddCmd( "echo",  ZAURA_RF_ShellCmdEcho );
 ZAURA_RF_ShellAddCmd( "filter",ZAURA_RF_ShellCmdFilter );
 ZAURA_RF_ShellAddCmd( "ipo",   ZAURA_RF_ShellCmdIpo );
 ZAURA_RF_ShellAddCmd( "nid",   ZAURA_RF_ShellCmdNid );
 ZAURA_RF_ShellAddCmd( "pa",    ZAURA_RF_ShellCmdPa );
 ZAURA_RF_ShellAddCmd( "per",   ZAURA_RF_ShellCmdPer );
 ZAURA_RF_ShellAddCmd( "port",  ZAURA_RF_ShellCmdPort );
 ZAURA_RF_ShellAddCmd( "pwr",   ZAURA_RF_ShellCmdPwr );
 ZAURA_RF_ShellAddCmd( "reboot",ZAURA_RF_SoftReset );
 ZAURA_RF_ShellAddCmd( "rssi",  ZAURA_RF_ShellCmdRssi );
 ZAURA_RF_ShellAddCmd( "rx",    ZAURA_RF_ShellCmdRx );
 ZAURA_RF_ShellAddCmd( "sleep", ZAURA_RF_ShellCmdSleep );
 ZAURA_RF_ShellAddCmd( "stats", ZAURA_RF_ShellCmdStats );
 ZAURA_RF_ShellAddCmd( "tx",    ZAURA_RF_ShellCmdTx );
 ZAURA_RF_ShellAddCmd( "uecho", ZAURA_RF_ShellCmdUartEcho );
}
```

## *ZAURA_RF_ShellAtox*

**Description** The ZAURA_RF_ShellAtox function replaces the ASCII string pointed to by pData with a converted hexadecimal value. No error check is performed on the string. It is assumed that the string contains only numbers from 0–9 and A–F (or a–f). If there are other characters, the result is unpredictable.

This API overwrites the pData buffer.

**Syntax** UINT8 ZAURA_RF_ShellAtox(char * pData)

**Parameters**

| | |
|---|---|
| pData | A pointer to the hexadecimal characters to be converted to a hexadecimal value. |

**Returns** Number of characters converted.

**Example**

```
UINT8 Data;
/*
 * Convert ASCII parameter to hexadecimal value.
 */
ZAURA_RF_ShellAtox( pZAURA_RF_Token[1] );
Data = (UINT8)*pZAURA_RF_Token[1];
```

## *ZAURA_RF_ShellStricmp*

**Description** The ZAURA_RF_ShellStricmp function performs a case insensitive comparison on the characters in strings s1 and s2.

**Syntax** INT8 ZAURA_RF_ShellStricmp(char * s1, char * s2)

**Parameters**

| | |
|---|---|
| s1 | A pointer to the first string to compare. |
| s2 | A pointer to the second string to compare. |

**Returns** 0 if identical (regardless of case), a signed integer if they are unequal, a positive number if s1 is more than s2, and a negative number if s2 is more than s1.

**Example**

```
if( ZAURA_RF_ShellStricmp("MyCmd", pZAURA_RF_Token[0]) ==
0 )
{
   // Operator entered a command like "mycmd" or "mYcMD"
}
```

## *ZAURA_RF_ShellHexDump*

**Description** This function displays Len bytes of RAM (in hexadecimal format) starting at the memory location referenced by `hData` on the console.

**Syntax** void ZAURA_RF_ShellHexDump(HANDLE hData, UINT8 Len)

**Parameters**

| | |
|---|---|
| hData | A pointer to the bytes to be displayed (in hexadecimal format). |
| Len | The number of bytes that are valid in hData. |

**Returns** None.

**Example**

```
UINT8 Data[4] = {0x11,0xBB,0xCC,0xDD};

/*
 * Display the Data array on the console in Hex
 */
ZAURA_RF_ShellHexDump( Data, 4 );
```

## *ZAURA_RF_ShellControl*

**Description** The ZAURA_RF_ShellControl function enables and disables the console. To prevent messages from being displayed on the console, you can disable the console by passing a 0 in the parameter. To re-enable pass a non-zero value. Disabling the console will block the operation of the `data` and `remote` console commands.

**Syntax** void ZAURA_RF_ShellControl(UINT8 Enable)

**Parameters**

| | |
|---|---|
| Enable | When this parameter is set to TRUE (nonzero), the console will be enabled. Set to FALSE (0) to disable it. |

**Returns** None.

**Example**

```
/*
 * Disable console output
 */
ZAURA_RF_ShellControl( 0 );
```

# ZAURA_RF_ShellPrintf

**Description** The ZAURA_RF_ShellPrintf prints a formatted string on the console. The format string defines what the output should look like and includes special characters as placeholders for numbers and characters that are passed to the function as arguments. This is a scaled down version of the standard printf().

**Syntax** ZAURA_RF_STATUS ZAURA_RF_ShellPrintf(const char * Format, …)

**Parameters**

| | |
|---|---|
| Format | The format string for the list of arguments. |
| … | A list of 0 or more arguments to be printed on the console, using the format string. |

The format string can contain any of the conversion flags and argument types listed in the following table.

| Conversion Flag | Argument Type | Output |
|---|---|---|
| %c | Unsigned 8-bit integer | ASCII representation of the 1 byte hexadecimal input value. Input should be between `0x20` and `0x7F`. |
| %d | Signed 16-bit integer | –32768 to 32767. |
| %ld | Signed 32-bit integer | –2147483648 to 2147483647. |
| %u | Unsigned 16-bit integer | 0 to 65535. |
| %lu | Unsigned 32-bit integer | 0 to 4,294,967,295. |
| %x | 16-bit hex value | 0 to FFFF (always upper case). |
| %lx | 32-bit hex value | 0 to FFFFFFFF (always upper case). |
| %s | Character string | ASCII representation of the character string. |
| %% | None | Add the '%' sign to the output. |

Additional Conversion Flag formatting information can be added between the '%' symbol and the letter that follows it. This extra formatting can be represented by any of the symbols defined in the following table.

| Conversion Flag | Meaning | Description |
|---|---|---|
| 1 to 9 | Minimum field width (1 to 9 characters) | Conversion is left-padded with spaces until the output reaches the specified minimum field width. Can be combined with '0' conversion flag to pad with leading zeros instead of spaces (for 'd', 'u' and 'x' numeric conversions). Unlike standard printf only a single-digit field width can be specified. |
| * | Minimum field width (1 to 255 characters) | Used to specify minimum field widths larger than 9 characters. Can be combined with '0' conversion flag to pad with leading zeros instead of spaces (for 'd', 'u' and 'x' numeric conversions). The field width is specified in the variable arguments list. |
| 0 | Zero left-pad | Used with either of the previous two flags to specify that left padding for numeric conversions should use leading zeros instead of spaces. Has no effect is used with 'c' or 's' conversion flags. |
| - | Left justify | By default ZAURA_RF_ShellPrintf will right justify the output if a minimum field width is specified and the converted value is narrow than the specified minimum. This flag cause the output to be left justified (right-padded). With left justification, the pad character is always a space (' '), even if the 0 flag is specified. |

| Conversion Flag | Meaning | Description |
|---|---|---|
| + | Show sign | By default ZAURA_RF_ShellPrintf will display a leading '-' when used with signed conversion ('d' or 'ld') and no sign indicator for positive values. This flag causes ZAURA_RF_ShellPrintf to display a leading '+' for positive signed values. |
| # | Hex 0x prefix | Causes con_print to prefix hexadecimal values ('x' and 'lx') with '0x' ignored on non-hexadecimal formats. Unlike printf ZAURA_RF_ShellPrintf will display a leading 0x when used to display the value 0. |

**Returns** ZAURA_RF_SUCCESS if the function was able to complete the request.

ZAURA_RF_FAILURE if the function was unable to write to the console, such as if it was disabled.

**Example**

```
/*
 * Displays "Data: 0010"
 */
ZAURA_RF_ShellPrintf( "Data: %04x\r\n", 16 );

/*
 * Displays "Data: 0x10"
 */
ZAURA_RF_ShellPrintf( "Data: %#04x\r\n", 16 );

/*
 * Displays "Data: +00000000016"
 */
ZAURA_RF_ShellPrintf( "Data: %+*0d\r\n", 12, 16 );
```

# Timer API Functions

The ZAURA RF Wireless Library uses TIMER 1 for internal operations; applications must not use TIMER 1. These APIs expose generic timer functions based off of the external 32.768kHz crystal. The tick period is approximately 30.5µs.

## ZAURA_RF_TickDelay

**Description** The ZAURA_RF_TickDelay implements a delay of Delay * 30.5µs. The maximum delay is approximately 1 second (`0x7FFF`). Control is not returned to the caller until the specified delay has expired.

**Syntax** void ZAURA_RF_TickDelay(UINT16 Delay)

**Parameters**

| | |
|---|---|
| Delay | The number of ticks to delay. |

**Returns** None.

**See Also** ZAURA_RF_ms_TO_TICKS

**Example**

```
/*
 * Wait 10 timer ticks (approximately 305µs)
 */
ZAURA_RF_TickDelay( 10 );
```

## ZAURA_RF_ms_TO_TICKS

**Description** The ZAURA_RF_ms_TO_TICKS macro converts milliseconds into ticks.

**Syntax** UINT16 ZAURA_RF_ms_TO_TICKS(UINT16 Delay_ms)

**Parameters**

| | |
|---|---|
| Delay_ms | The duration of time it takes, in milliseconds, to convert to ticks. |

**Returns** The number of ticks that Delay_ms converts to.

**Example**

```
UINT16 Ticks;

/*
 * Convert 30ms to the corresponding number of timer
 * ticks.
 */
Ticks = ZAURA_RF_ms_TO_TICKS( 30 );
```

## *ZAURA_RF_ReadTimer*

**Description** This routine returns a value between `0x0001` and `0x8000`, representing the current tick count of the ZAURA RF timer.

**Syntax** UINT16 ZAURA_RF_ReadTimer( void );

**Parameters** None.

**Returns** A 16-bit tick count of the ZAURA RF timer (frequency approximately 32.768kHz).

**Example**

```
UINT16 TickCount;
TickCount = ZAURA_RF_ReadTimer();
```

## ZAURA_RF_GetTicks

**Description** The ZAURA_RF_GetTicks function is used to measure short intervals (less than 1 second) instead of wasting CPU cycles in a synchronous delay loop. This allows the application to perform simple tasks while waiting for some other operation to complete.

**Syntax** UINT16 ZAURA_RF_GetTicks(UINT16 Start)

**Parameters**

| | |
|---|---|
| Start | A 16-bit timestamp indicating the beginning of an interval. |

**Returns** 16-bit count of the number of timer ticks have elapsed since the interval began. (Max: `0x7FFF`).

**Example**

```
/*
 * Check for console input for up to 700ms.
 */
UINT16 Start = ZAURA_RF_ReadTimer();

while( ZAURA_RF_GetTicks(Start) <
          ZAURA_RF_ms_TO_TICKS(700) )
{
   if( ZAURA_RF_UartEOL )
   {
      ZAURA_RF_ProcessCmdLine();
      break;
   }
}
```

# Appendix A. Project Information

This section of the document discusses the user code space within Flash memory and explains how to create an application with the ZAURA RF Wireless Module using the ZAURA RF Wireless Module and Shell libraries and Zilog Developer Studio.

## Memory Map

The default ZAURA RF Wireless Module demo project uses approximately 18KB of Flash memory. Almost half of the code space is used by the Shell Library (estimated at 8KB). Applications that use the ZAURA RF Wireless Shell Library are unlikely to require all of the extra console commands available in the library.

The ZAURA RF Wireless Library uses the last page of Flash memory, in the address range `0x5E00` to `0x5FFF`, for storing parameters. Application programs that use the ZAURA RF Wireless Module libraries must not use any Flash memory above `0x5DFF`.

The ZAURA RF Library uses all 1KB of available PRAM for packet buffers. Application programs must not attempt to use PRAM for any purpose.

## Z8F2480 MCU Peripherals

The ZAURA RF Wireless Library uses the following Z8F2480 MCU peripherals: UART0, ESPI and TIMER1. These peripherals must not be used by applications linking to the ZAURA RF Wireless Library.

Table 15 shows the Z8F2480 MCU's available GPIO pins and their usage. All other pins are used by the ZAURA RF Wireless Library and cannot be used for any other purpose.

**Table 15. Z8F2480 MCU GPIOs**

| Port Pin | Setting | Usage |
|---|---|---|
| **Port A** | | |
| 0 | Out_0 | Can be used by the application for any purpose. |
| 1 | Out_0 | Can be used by the application for any purpose. |
| 6 | Out_0 | Can be used by the application for any purpose. |
| 7 | Out_0 | Can be used by the application for any purpose. |
| **Port B** | | |
| 0 | Out_0 | Can be used by the application for any purpose. |
| 1 | Out_0 | Can be used by the application for any purpose. |
| 2 | Out_0 | Can be used by the application for any purpose. |
| 3 | Out_0 | Can be used by the application for any purpose. |
| 4 | Out_0 | Can be used by the application for any purpose. |
| 5 | Out_0 | Can be used by the application for any purpose. |
| **Port D** | | |
| 0 | Out_0 | Reset/ GPIO. |
| 1 | In_IRQ | SW2. |
| 2 | In_IRQ | SW1. |
| 3 | Out_0 | Unused/ UART 1 CTS. |
| 4 | Out_0 | Unused/ UART 1 RX. |
| 5 | Out_0 | Unused/ UART 1 Tx. |
| 6 | Out_0 | Unused/ UART 1 RTS. |
| 7 | Out_0 | Can be used by the application for any purpose. |

**Table 15. Z8F2480 MCU GPIOs (Continued)**

| Port Pin | Setting | Usage |
|---|---|---|
| **Port E** | | |
| 0 | Out | ZAURA RF Wireless Library test point. |
| 1 | Out | ZAURA RF Wireless Library test point. |
| 2 | Out_0 | Can be used by the application for any purpose. |
| 3 | In | UART 0 CTS. |
| 4 | Out | UART 0 RTS. |
| 5 | Out | LED 1. |
| 6 | Out | LED 2. |

# Using ZDSII to Create an Application

The ZAURA RF Wireless Module and Shell libraries are designed to be used with ZDSII for Encore! version 5.0.

The library installs the following folders:

| | |
|---|---|
| Conf | Contains the configuration files, including `ZAURA_RF_Conf.c`. Typically, you will want to copy these files to your project folder if you will be making any changes. |
| Demo | This folder contains the demo project that is loaded on the kits at the factory. This also is an example of how to use the libraries. You can use this as a starting point for your application. |
| Docs | This folder contains the demo information and this Programmer's Reference Manual. |
| Inc | This folder contains the header files necessary for using the libraries. |
| Lib | This folder contains the libraries in object form, to be linked with your application. |

For purposes of readability, the base folder will be referenced as `ZAURA_RF_Wireless`, although it may be different on your installation.

## Create an Application

Create a new ZAURA RF Wireless Module application with Zilog Developer Studio by observing the following procedure.

**Note:** The easiest way to create an application is to copy the demo project to a new folder and use that folder as your starting point.

1. Make a new folder in the root of the ZAURA_RF_Wireless installation folder that will be a peer to the Demo folder.
2. Copy the `ZAURA_RF_DEMO.zdsproj` file from the Demo folder into the new folder (rename the project file if appropriate).
3. Launch the ZDSII – Z8 Encore! 5.0.0 program and select the **File → Open Project** menu option. Navigate to the location of the project folder created in Step 1 and double-click the project file created in this folder in Step 2. The default name will be `ZAURA_RF_Demo.zdsproj` unless you used a different project name in Step 2.
4. Remove all four source files from the project by right-clicking each file listed under Standard Project Files and selecting the **Remove Selected File(s)** pop-up menu option.
5. Copy the original `ZAURA_RF_Conf.c`, `GpioConfig.c` and `reset.asm` files from the `ZAURA_RF_Wireless\Conf` folder to the new project folder.
6. Add the three files copied in Step 5 to the project by right-clicking **Standard Project Files**, then selecting the **Add Files To Project...** pop-up menu option. Be sure to navigate to the project folder created in Step 1, then the three files you copied in Step 5.
7. Create a file named `main.c` in the project folder that you created in Step 1.

**Note:** Users that prefer to start from a working project can simply copy `main.c` from the Demo folder to the project folder created in Step 1. In this instance, skip ahead to Step 10.

Users that prefer to create a new `main.c` file can select the **File → New File** menu option and continue with Step 8. In this instance, an empty window will open in ZDSII in which you can start typing code.

8. Copy and paste the following text into the empty window (or enter it manually):

```
void main( void )
{
}
```

9. Select the **File → Save As** menu option. A dialog box will appear, prompting you for a name to give the new file. Enter `main.c` in the **File name:** text field.
10. Add this new `main.c` file to the project using the same process you used in Step 6.
11. Your project is complete! You can now begin building your application.

# Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

# OCEAN CHIPS
## Океан Электроники
## Поставка электронных компонентов

Компания «Океан Электроники» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Поставка оригинальных импортных электронных компонентов напрямую с производств Америки, Европы и Азии, а так же с крупнейших складов мира;
- Широкая линейка поставок активных и пассивных импортных электронных компонентов (более 30 млн. наименований);
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Помощь Конструкторского Отдела и консультации квалифицированных инженеров;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Поставка электронных компонентов под контролем ВП;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- При необходимости вся продукция военного и аэрокосмического назначения проходит испытания и сертификацию в лаборатории (по согласованию с заказчиком);
- Поставка специализированных компонентов военного и аэрокосмического уровня качества (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Actel, Aeroflex, Peregrine, VPT, Syfer, Eurofarad, Texas Instruments, MS Kennedy, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits, General Dynamics и др.);

Компания «Океан Электроники» является официальным дистрибьютором и эксклюзивным представителем в России одного из крупнейших производителей разъемов военного и аэрокосмического назначения «**JONHON**», а так же официальным дистрибьютором и эксклюзивным представителем в России производителя высокотехнологичных и надежных решений для передачи СВЧ сигналов «**FORSTAR**».

AVIC JONHON

**«JONHON»** (основан в 1970 г.)

Разъемы специального, военного и аэрокосмического назначения:

(Применяются в военной, авиационной, аэрокосмической, морской, железнодорожной, горно- и нефтедобывающей отраслях промышленности)

**«FORSTAR»** (основан в 1998 г.)

ВЧ соединители, коаксиальные кабели, кабельные сборки и микроволновые компоненты:

(Применяются в телекоммуникациях гражданского и специального назначения, в средствах связи, РЛС, а так же военной, авиационной и аэрокосмической отраслях промышленности).

FORSTAR®

Телефон: 8 (812) 309-75-97 (многоканальный)
Факс: 8 (812) 320-03-32
Электронная почта: ocean@oceanchips.ru
Web: http://oceanchips.ru/
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, д. 2, корп. 4, лит. А