Application Note

Getting Started with ZAURA RF Control

AN033601-0811

Abstract

This application note describes how Zilog's ZAURA RF Wireless modules and software libraries can be used to quickly add RF control to an embedded application. This document offers three methods for sending control information using these ZAURA RF libraries.

These three methods differ in the amount of new software that must be written for the ZAURA RF Wireless modules. The first two methods are only applicable if the embedded application includes the ZAURA RF Wireless Shell Library (which is included in the ZAURA RF Wireless software package); however, they require the least amount of custom software and understanding of these libraries. The third method does not require the use of the Shell, yet requires more custom software and a better understanding of the ZAURA RF libraries.

While it is also possible to use the ZAURA RF Wireless modules without using any code from the ZAURA RF libraries, this approach requires a very intimate knowledge of the underlying RF controller, and is beyond the scope of this document.

Note: The source code file associated with this application note, AN0336-SC01.zip, is available for download on zilog.com. This source code has been tested with version 5.0.0 of ZDSII for Z8 Encore! XP-powered MCUs. Subsequent releases of ZDSII may require you to modify the code supplied with this application note.

Hardware Overview: ZAURA RF Wireless Module and Validation Board

This application requires the use of the ZAURA RF Wireless Module and ZAURA Validation Board. The Module is a cost-effective, high-performance radio system for the wireless transmission of digital information. The module's small form factor allows easy integration into any number of monitoring and control applications. Either of the 868MHz or 915MHz ZAURA RF Wireless modules can be used with this application note.

For illustrative purposes, the Module is mounted to the ZAURA Validation Board to allow easy access to the GPIO port pins and peripheral devices on the Module's Z8F2480 MCU. This application note and its accompanying software utilizes the SW1 and SW2 pushbuttons, LED1 (Port E5) and UART0 on the Validation Board.

Software Overview: ZAURA RF Libraries

This application note leverages the software in the ZAURA RF and Shell libraries to demonstrate techniques to add basic wireless communications to an embedded application. The RF Library (`ZAURA_RF_866p5_MHz.lib` or `ZAURA_RF_915_MHz.lib`, depending on which Module is used) contains routines to configure the local Module's radio and communicate with remote modules. The RF library also implements a Reliable Data Transfer and Flow Control protocol (RDTFC) that can be used to reduce (but not eliminate) data loss inherent in RF systems.

The ZAURA RF Shell Library implements a command line interpreter that can be used to control and communicate with local and remote ZAURA RF Wireless modules through an RS-232 (UART) interface. The ZAURA Validation Board provides access to UART0 on the Z8F2480 MCU through a dedicated USB to RS-232 controller. This access allows a terminal program running on a PC to be used to control the ZAURA RF Wireless modules. In an actual embedded application, RS-232 communication could be established between the ZAURA RF Wireless Module and a special-purpose embedded processor requiring RF communication with remote nodes.

Although it is not mandatory to use the ZAURA RF Shell Library to enable wireless communications with remote modules, two of the three sample programs described in this application note use the Shell Library. All of the sample programs described in this application note use the ZAURA RF Wireless Library.

Theory of Operation

The ZAURA RF Wireless Library uses an ad-hoc wireless 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 methods. There is no central coordinating unit through which nodes must communicate and the RF Library does not provide routing of any kind between nodes that are not within radio range of each other.

Two nodes have compatible configurations if they use the same RF channel, network ID and frame format. The RF channel specifies the centre frequency the ZAURA RF Wireless radio uses to send or receive data. The network ID allows ZAURA RF Wireless nodes that use the same RF channel to be subdivided into logical groups or cells such that if two nodes are configured to use the same RF channel but have different network IDs, they will be unable to communicate with each other. The ZAURA RF Wireless Library supports three different frame formats that allow for increasingly more complex data transfer. Nodes that must communicate with one another should be configured to use the same frame format.

All frame formats use an 8-bit destination address that identifies the intended recipient(s) of the frame. Accordingly, each node in the RF cell can be assigned a unique 8-bit address in the range of 1 to 254 (`0x01` to `0xFE`). Node address 255 (`0xFF`) is the broadcast address. Frames transmitted to the broadcast address can be received by any node within the same cell (common RF channel and network ID) within radio range of the transmitter.

The ZAURA RF Wireless Library implements both reliable and unreliable data transfer mechanisms (based on the configured frame format). Data that is transmitted unreliably is sent on a best-effort basis. The data may reach the intended recipient(s) or may get lost (e.g. out of range) or corrupted (RF interference) during transmission. When data is sent reliably, the RF Library will attempt to determine if the intended target actually received the transmission and will automatically retransmit the data a configurable number of times until the target indicates that the data was successfully received, or until reaching the retry limit. In general, it will take longer to send the same amount of data reliably versus unreliably due to the extra overhead incurred.

In this application note, one Module acts as a control unit that senses an external event (e.g., the push of a button) and commands multiple remote slave units to initiate an action (e.g., setting GPIO port pins to a certain state). The implementations described in this application note typically use unreliable data transfer (broadcast target address) and will automatically switch to using reliable data transfer with directed addressing based on the value of the global variable `ZAURA_RF_Dest`.

For additional information about the ZAURA RF Wireless Library, please refer to the ZAURA RF Wireless Library Reference Manual (RM0060).

Software Implementation

The following sections describe three different implementations that can be used to transmit commands from a control unit to multiple remote slave units using the ZAURA RF Wireless Library. Each section begins with a description of the implementation, and is followed by a description of the relevant project files, source code and configuration of the Library. Each section concludes with a suggested procedure for running the project.

Implementation #1: Using the Remote Port Shell Commands

In this approach, the ZAURA RF Wireless Library is used to manipulate GPIO port pins on remote slave units in response to events on a control unit. This approach is most useful if there is a secondary processor in the control unit that runs application code independently of the ZAURA RF Wireless Module. In this scenario, the secondary processor communicates with the RF Module using an RS-232 interface. Because the ZAURA RF Shell Library already includes commands that can be used to manipulate GPIO pins on remote nodes, this approach requires less code to be written for the ZAURA RF Wireless Module than the other approaches. In fact, this approach can be demonstrated using the ZAURA RF Wireless Demo project without writing a single line of code.

Because it is beyond the scope of this document to create software for a secondary processor, use of the Shell for RF control is demonstrated using a PC running a terminal emulation program such as Tera Term or HyperTerminal. This terminal emulator allows a user to type shell commands in the terminal program which the PC (secondary processor) will send over a USB interface to the RF Module's UART interface.

Observe the following procedure to demonstrate how the shell can be used to control GPIO pins on remote nodes:

1. Slide switch SW3 on the ZAURA Validation Board to the USB PWR ON position.

2. Insert two AA batteries into the holders on the bottom of the ZAURA Validation Board. Next, slide switch SW3 to the BAT PWR ON.
3. Launch ZDSII and navigate to the `..\ZAURA_RF_Wireless_vX.YZa\Demo` folder (in which *X.YZa* is the version of the ZAURA RF Wireless software installed; for example: *v1.01a*).
4. Connect a Zilog debug tool, such as Zilog's USB Smart Cable, to the PC. Connect the ribbon cable between the debug tool and the ZAURA Validation Board, noting the orientation of pin 1.
5. Open the `ZAURA_RF_Demo.zdsproj` project file and select the build configuration appropriate for the RF modules being used; this configuration will be either the `866p5_MHz` or `915_MHz` build option.
6. Next, select the **Build** → **Rebuild All** menu option.
7. Download the project using the **Tools** → **Flash Loader** option. After the firmware has been flashed, disconnect the debug tool ribbon cable and slide switch SW3 to the USB PWR ON position, then slide it back to the BAT PWR ON position to reboot the unit and ensure that it will start running the new firmware.
8. Return to Step 4 for each ZAURA RF Wireless Module that must be reprogrammed before continuing.

Note: By default, the `ZAURA_RF_Demo` firmware is programmed into the ZAURA RF Wireless modules at the time of manufacture. Therefore, unless the firmware has been modified, these steps may not be necessary.

9. Select one of the ZAURA RF Wireless modules to act as the control unit and remove the two AA batteries from this unit. Connect a USB Mini-B to A cable between the control unit and a PC running a terminal program such as HyperTerminal.

Note: When the USB cable is first connected to the PC, the PC may need to download software from the Internet to enable the USB-to-serial interface connection on the ZAURA Validation Board.

10. Configure the terminal program for 57600bps, 8 data bits, no parity, one stop bit and optionally enable hardware flow control.

Note: By default, the ZAURA RF Shell does not echo typed characters onto the console. Consequently, to see typed characters, it will be necessary to either configure the PC terminal program to enable local echo or issue the `uecho on` console command (or set the

ZAURA configuration variable, `ZAURA_RF_UartEcho`, to TRUE in `..\Conf\ZAURA_RF_Conf.c`).

11. On the control unit's console, enter the command `remote port e & df`. This command should cause LED D1 (PE5) to illuminate on all slave nodes within radio range of the control unit.
12. On the control unit's console, enter the command `remote port e | 20`. This command should cause LED D1 (PE5) to extinguish on all slave nodes within radio range of the control unit.
13. On the control unit's console, enter the command `remote port e ^ 20`. This command should toggle the state of LED D1 (PE5) on all slave nodes within radio range of the control unit.

How It Works

This implementation uses two ZAURA RF Shell Library commands to manipulate GPIO ports on remote slave units; these Shell commands are `port` and `remote`. The `port` command, which is described in the ZAURA RF Module Shell User Manual (UM0235), can be used to manipulate the Z8F2480 MCU's PxOUT Register, which allows control of individual GPIO pins. The following port pins are available for general-purpose use with the ZAURA Validation Board:

Table 1.

Port	Available Pins
A	0, 1, 6, 7
B	0–5
D	3–7
E	2

On the ZAURA Validation Board, PE5 controls LED D1. When PE5 is set to 0, LED D1 illuminates, and when PE5 is 1, LED D1 extinguishes. Because pin 5 corresponds to the hexadecimal value `0x20`, the commands m the test procedure perform bit-wise operations on the PEOUT Register and the value `0x20`. For example, `port e | 20` is equivalent to the *C* statement `PEOUT |= 0x20;`.

The second Shell command used is the `remote` command. This command sends the text following the `remote` keyword to the node address of the ZAURA RF Wireless Module(s) targeted by the value of the global variable `ZAURA_RF_Dest`. The `dst` shell command can be used to query or modify the value of the `ZAURA_RF_Dest` global variable. By default, this value will be `FF`, the broadcast address. As a result, when the `remote` keyword prefixes one of the shell commands, it will be broadcast to all ZAURA RF Wireless modules within radio range of the control unit.

Limitations

By sending a different `port` command, it is relatively easy for the secondary processor to manipulate GPIO pins on remote RF modules. However, for control applications that require more complex manipulation of the GPIO port pins, this approach is very limiting. For example, if multiple pins on different GPIO ports must be manipulated, it would be necessary to send multiple `port` commands. In addition, there is often a time delay between subsequent `port` commands which may produce undesirable effects.

Another undesirable effect of using this approach is that the default implementation of the `port` command sends status information back to the console of the control node; this status information must consequently be processed by the secondary processor. If there are only a few remote slave nodes, this situation may be acceptable. However, as the number of remote slave nodes increases, the volume of response data returning to the control unit's console may become unwieldy. Furthermore, each of the slave nodes will attempt to send their responses at approximately the same time, an issue which can cause collisions and result in lost data. Therefore, this approach is most useful when there is no output from the shell command executed on the remote slaves.

Yet another limitation of this approach is that all ZAURA RF Wireless modules are using the same 8-bit address (default value `0x1B`). As a result, it is not possible for the control unit to query the status of individual slave units. Even worse, because all units share a common RF address, it is not possible to send data reliably. For example, if the control unit was to send an SDATA packet (reliable data transfer) directed to node `0x1B`, multiple slave nodes would attempt to ACK/NAK the packet, thereby rendering the RDTFC protocol useless.

Implementation #2: Custom Shell Command

The second implementation strategy attempts to overcome some of the limitations of the first approach. The ZAURA RF Shell will still be used to issue commands to remote nodes, but this time, custom shell commands will be created that allow for more complex manipulation of the GPIO port pins, eliminate console output on the new commands and allow for the unique addressing of RF modules for the purpose of polling status. In addition, the pushbuttons on the control unit's Validation Board will be used as triggers for issuing remote commands.

Before running the project, the significant code blocks from the source file `..\AN0336\p1_main.c` are analyzed.

Custom Shell Commands

The P1 project adds three new shell commands to the ZAURA RF Shell: `on`, `off` and `query`. The `on` command is used to illuminate LED D1 on the Validation Board(s) targeted by the value of the `ZAURA_RF_Dest` configuration variable. Similarly, the `off` command is used to extinguish LED D1 on the Validation Board(s) targeted by the value of the `ZAURA_RF_Dest` configuration variable. The `query` command directs the node(s) targeted by the `ZAURA_RF_Dest` configuration variable to report the state of LED D1.

The code that implements the `on` and `off` shell commands is trivial, as is shown in the routine that follows. Even though the ZAURA RF Wireless Library `port` command can

be used to manipulate LED D1 (PE5), there are two advantages to using a custom shell command. First, for more complicated GPIO port pin manipulations, using the `port` shell command is inefficient at best and possibly ineffective. Second, the `port` command generates console output that must be processed by the secondary processor of the control node. With the new `on` and `off` shell commands, there is no output from the remote slave units that must be processed by the secondary processor on the control unit.

```
void
OnCmd
(
 void
)
{
 LED_D1_ON;
}


void
OffCmd
(
 void
)
{
 LED_D1_OFF;
}


void
QueryCmd
(
 void
)
{
 DEBUGPRINTF( "Node %02x LED_D1 ", ZAURA_RF_Params.Addr );
 if( PEOUT & LED_D1_PIN )
 {
   DEBUGPRINTF( "Off\r\n" );
 }
  else
  {
   DEBUGPRINTF( "On\r\n" );
  }
}
```

By design, the new `query` console command generates output that the secondary processor on the control unit must process. The response will be similar to the following console output:

```
Node 02 LED D1 Off
```

```
Node 03 LED D1 On
```

The code fragment that adds these three new commands to the ZAURA RF Shell is in the `InitShell` routine:

```
/*
 * Application-specific Shell commands
 */
 ZAURA_RF_ShellAddCmd( "on", OnCmd );
 ZAURA_RF_ShellAddCmd( "off", OffCmd );
 ZAURA_RF_ShellAddCmd( "query", QueryCmd );
```

Pushbuttons Issue On/Off Commands

In the P1 project, the SW1 pushbutton is used to send the `on` shell command to the node(s) targeted by `ZAURA_RF_Dest` and SW2 is used to send the `off` command. Therefore, the new project has two methods of commanding remote nodes to turn on/off LED D1. The secondary processor can use its RS-232 interface to transmit the ASCII strings `remote on` or `remote off` to modify the state of LED D1 on `ZAURA_RF_Dest`; or the user can press the corresponding button on the ZAURA Validation Board.

The implementation of the SW1 interrupt service routine (ISR) is shown below.

```
void interrupt
OnButtonIsr
(
 void
)
{
 if( SW1_DOWN )
  {
   /*
    * Disable subsequent button interrupts until this
    * interrupt has been fully processed.
    */
    IRQ1ENL &= ~(SW1_PIN | SW2_PIN);

   /*
    * Since this implementation transmits from within the ISR
    * it is necessary to reenable interrupts to allow proper
    * operation of the ZAURA_RF_Transmit API.
    */
   asm( "EI" );

   /*
    * Issue the "on" command to the Shell on ZAURA_RF_Dest
    */
   ZAURA_RF_Transmit( ZAURA_RF_Dest, SDATA | SAP_CMD_INTRPTR, "on", 2 );
   while( ZAURA_RF_GetState() == ZAURA_RF_TRANSMIT );
```

```
    OnCmd();

  /*
   * Reenable pushbutton interrupts
   */
   asm( "DI" );
   IRQ1ENL |= (SW1_PIN | SW2_PIN);
  }
}
```

The first thing the ISR does is verify that button SW1 is in the *pushed* state; it next disables subsequent interrupts from either SW1 or SW2 to ensure that any bounce on the switch inputs does not result in multiple commands being issued to `ZAURA_RF_Dest`.

Next, the ISR reenables interrupts on the Z8F2480 processor. This step is necessary because the button ISR transmits from within the ISR and the ZAURA RF Wireless Library requires RF and timer interrupts to process a transmit request.

The `ZAURA_RF_Transmit` API is used to send ASCII text of the command to be executed (`on`) to the node(s) targeted by the `ZAURA_RF_Dest` global variable. The second parameter on the transmit call specifies the target Service Access Point (SAP) on the remote node(s). Because the `on` command must be processed by the ZAURA RF Shell command interpreter on the remote node(s), a SAP address of `SAP_CMD_INTRPTR` is specified. The second parameter also specifies whether unreliable or reliable data services are being requested.

The ZAURA RF Wireless library will transmit DATA frames unreliably and will transmit SDATA frames reliably using the ZAURA RDTFC protocol. However, the ZAURA RDTFC protocol does not support reliable delivery of broadcast data. Therefore, if `ZAURA_RF_Dest` equals `ZAURA_RF_BC_ADDR (0xFF)`, then the RF library will automatically use a DATA frame without the RDTFC protocol to send the requested data. By coding the `ZAURA_RF_Transmit` call to request reliable data transfer (via the SDATA parameter), this same code fragment can be used to transmit data reliably or unreliably without having to explicitly check the value of `ZAURA_RF_Dest`.

It is beyond the scope of this application note to determine which value of `ZAURA_RF_Dest` to use. By default, `ZAURA_RF_Dest` is assigned the value `ZAURA_RF_BC_ADDR` in `..\Conf\ZAURA_RF_Conf.c`, but this value can be changed at compile time or run time. At run time, the secondary processor can issue the `dst xx` shell command to set the value of `ZAURA_RF_Dest` to `xx`. Code that runs natively on the Z8F2480 processor and links to the ZAURA RF Wireless Library can simply modify the value of the `ZAURA_RF_Dest` global variable.

The final function performed in the ISR is to reenable button interrupts and return to the interrupted task. The code in `OffButtonIsr` is similar, but the transmit command is slightly different, as shown below.

```
/*
 * Issue the "off" command to the Shell on ZAURA_RF_Dest
 */
 ZAURA_RF_Transmit( ZAURA_RF_Dest, SDATA | SAP_CMD_INTRPTR, "off", 3 );
```

Serializing the ZAURA RF Address

In the previous implementation, the ZDS Flash Loader tool was used to program the controller node and multiple slave nodes with the ZAURA Demo project. Because the RF configuration file was not modified between downloads, the same set of RF parameters was assigned to all nodes.

To allow communication between the controller and slave unit(s), it is mandatory that all units are configured to use the same RF channel and network ID. However, it is undesirable that all nodes are configured to use the same nodes address. Ideally, each node should be assigned a unique 8-bit node address in the range of 1 to 254. Exactly how the node address (and for that matter, the network ID and RF channel) is assigned is implementation-dependent. These parameters can all be modified at run time using API calls in the ZAURA RF Wireless Library or by issuing shell commands via a secondary processor. In this implementation, the P1 project makes use of the ZDSII Flash Loader's Serialization utility to assign a unique address to each node without having to rebuild the project.

Before describing how serialization of the node address can be accomplished in the P1 project, it is first necessary to understand how the ZAURA RF Wireless library obtains its RF parameters. The RF library has two methods of obtaining its RF parameters; the first method uses values stored in the last page of Flash memory. The second method uses the compile-time values assigned to the global variable `ZAURA_RF_Params` (see the `ZAURA_RF_Conf.c` file). During the call to `ZAURA_RF_Init`, the RF Library reads the set of parameters stored in Flash starting at address `0x5F00`. If all of the RF parameters are valid (`NID`, `Addr`, `Channel`, `Tx Power` and `Rx Filter`), then these values will be used to override the values assigned to the `ZAURA_RF_Params` structure at compile time. If any of the parameters read from Flash are invalid, then the RF library will use the values assigned to the `ZAURA_RF_Params` structure at compile time.

The node address RF parameter is stored in Flash memory at address `0x5F05`. Therefore, while downloading the P1 project to Flash, the ZDS Flash Loader's Serialization tool will be used to set a different value into this Flash location for up to 254 units programmed with the same hex file. However, in itself, this operation is not sufficient to assign a unique address to each node, because only one byte's worth of the RF parameters stored in Flash is serialized; the remainder of the parameter values will have the value `0xFF`. Consequently at run time, the RF Library will determine that the RF parameters in Flash are invalid and will use the default values assigned to the `ZAURA_RF_Params` structure at compile time.

To solve this problem, the following code block is added to `main` just after the call to `ZAURA_RF_Init`.

```
#ifdef USE_ADDR_SERIALIZATION
  {
   UINT8 Addr;

   Addr = *(const UINT8 *)0x5F05;
   if( (Addr != ZAURA_RF_BC_ADDR) && (Addr != ZAURA_RF_Params.Addr) )
   {
```

```
    DEBUGPRINTF( "Overriding default node address %02x with
%02x\r\n",
   ZAURA_RF_Params.Addr, Addr );
   ZAURA_RF_SetAddr( Addr );
   ZAURA_RF_SetParams( );
   }
  }
  #endif
```

Note: The code block above is conditionally compiled into the project if the `USE_ADDR_SERIALIZATION` macro has been defined (it is currently set with a default definition that includes conditionally-compiled code blocks)[1]. At the top of the project is the following line of code:

```
#define USE_ADDR_SERIALIZATION
```

By default, this code string is included. The user can comment out the macro definition (i.e., undefines the macro) as follows:

```
//#define USE_ADDR_SERIALIZATION
```

As a result, the code within the conditionally-compiled block will not be included in the project.

Taking the above code block in its entirety, the value at offset 5 in the parameters page (i.e., the node address value stored in Flash at address `0x5F05`) is stored in the local variable `Addr` and validated. If this value is not `0xFF` and does not match the `Addr` member of the `ZAURA_RF_Params` structure, the code block calls the `ZAURA_RF_SetAddr` API to make this the new node address of the unit and then calls `ZAURA_RF_SetParams` to rewrite all run-time parameters back to Flash. Upon subsequent boots, it is expected that the value that is read from Flash will match the `Addr` member of the `ZAURA_RF_Params` structure; therefore, it will not be necessary to repeat this process.

Note: If this unit's node address is ever modified, or if the Flash Parameters block is ever erased, the value written by the ZDSII Serialization Tool will be lost. If the Flash parameters are

1. For a conditionally-compiled code block such as:
   #ifdef XXX
   code
   #endif

   the code within the block will be included in the project if the XXX macro is defined; the code within the block will not be included in the project when the XXX macro is undefined.

erased, then the node will default to the value of the `Addr` member of the `ZAURA_RF_Params` structure.

Running the Project

Source code for the Custom Shell Command implementation is contained in the AN0336-SC01.zip file. To run the code, observe the following procedure:

1. Place the AN0336-SC01.zip file into the root of your ZAURA RF Wireless installation and extract the contents of this zip file into the AN0336 subfolder. The AN0336 subfolder should be a peer to the Demo folder.
2. Launch ZDSII and navigate to the `..\ZAURA_RF_Wireless_vX.YZa\AN0336` folder (in which *X.YZa* is the version of the ZAURA RF Wireless software installed; for example: *v1.10a*) and open the `p1.zdsproj` file.
3. Select the build configuration appropriate for the RF modules being used. This configuration will be either the `866p5_MHz` or `915_MHz` build option.
4. Next, select the **Build** → **Rebuild All** menu option.
5. Download the project using the **Tools** → **Flash Loader** option.
6. Repeat Step 5 for each RF Module that must be reprogrammed. Note the 1-byte *serial number* programmed into the first slave unit using the Flash Loader.
7. Select one of the ZAURA RF Wireless modules to act as the control unit and remove the two AA batteries from this unit. Connect a USB Mini-B to A cable between the control unit and a PC running a terminal program such as HyperTerminal.

Note: When the USB cable is first connected to the PC, the PC may need to download software from the Internet to enable the USB to serial interface of the ZAURA Validation Board.

8. Configure the terminal program for 57600bps, 8 data bits, no parity, one stop bit and optionally enable hardware flow control.

Note: By default, the ZAURA RF Shell does not echo typed characters onto the console. Consequently, to see typed characters, it will be necessary to either configure the PC terminal program to enable local echo or issue the `uecho on` console command (or set the ZAURA configuration variable, `ZAURA_RF_UartEcho`, to TRUE in `..\Conf\ZAURA_RF_Conf.c`).

9. On the control unit's console, enter the command `remote on`. This command should cause LED D1 (PE5) to illuminate on all slave nodes within radio range of the control unit.

10. On the control unit's console, enter the command `remote off`. This command should cause LED D1 (PE5) to extinguish on all slave nodes within radio range of the control unit.
11. On the control unit's console, enter the command `remote query`. This command should cause all slave units to report the status of LED D1 to the control unit's console.
12. On the control unit, press SW1 (PD2). This command should cause LED D1 to illuminate on all slave units. Similarly, pressing SW2 (PD1) will cause all slave units to extinguish LED D1.
13. On the control unit's console, enter the command `dst xx`, in which `xx` is the *serial number* recorded in Step 7.
14. On the control unit, press SW1 again. This time, LED D1 should only illuminate on the slave unit in which the RF address matches the value used in the previous step. Similarly, pressing SW2 will only extinguish LED2 on the targeted slave unit.

How it Works

This implementation uses the custom shell commands `on`, `off` and `query` to illuminate, extinguish and query the state of LED D1 locally on the control unit. By prefixing these custom shell commands with the `remote` keyword, LED D1 can be manipulated on slave unit(s) targeted by the `ZAURA_RF_Dest` global variable. The value of `ZAURA_RF_Dest` can be changed programmatically or modified using the `dst` console command.

Instead of issuing a console command to control LED D1, this implementation also allows the user to press SW1 and SW2 to turn on/off LED D1 on the slave unit(s) targeted by the `ZAURA_RF_Dest` global variable.

Note: When `ZAURA_RF_Dest` is the broadcast address `0xFF`, data is transmitted unreliably. When `ZAURA_RF_Dest` targets a specific slave (addresses in the range 1 to 254), only that slave unit will react to the command if it is within RF range of the control unit. Slave units are assigned a unique RF address via the ZDSII serialization control function within the Flash Loader utility.

The `on` and `off` commands implemented in the P1 project do not generate any console output; therefore, the control unit's console is not filled with unnecessary status information. This situation is beneficial when a secondary processor within the control unit must process output from UART0.

Limitations

As with the first implementation that used the ZAURA RF Shell Library's `port` console command, it is still necessary for the application to include the ZAURA RF Shell Library in the project. However, when the application is intended to run natively on the Z8F2480 processor without using a secondary processor, including the Shell Library increases the size of the overall project, thereby reducing the amount of Flash available to the applica-

tion. In addition, if the embedded device does not support an external UART interface, the Shell Library is of little utility.

Implementation #3: Application Packet Handling

The final implementation described in this application note requires the use of custom packet handling routines. These custom routines eliminate the requirement for the ZAURA RF Shell Library, which can save more code space than is necessary to implement custom packet-handling routines.

Before running the project, the significant code blocks from the source file `..\AN0336\p2_main.c` are analyzed.

Packet Transmission

In this project, SW1 and SW2 on the control unit are used to manipulate the state of LED D1 on slave unit(s) targeted by the `ZAURA_RF_Dest` global variable. When SW1 is pressed, the `on` command is sent to `ZAURA_RF_Dest` from within the SW1 interrupt service routine.

```
void interrupt
OnButtonIsr
(
  void
)
{
  if( SW1_DOWN )
  {
   /*
    * Disable subsequent button interrupts until this
    * interrupt has been fully processed.
    */
   IRQ1ENL &= ~(SW1_PIN | SW2_PIN);

   /*
    * Since this implementation transmits from within the ISR
    * it is necessary to reenable interrupts to allow proper
    * operation of the ZAURA_RF_Transmit API.
    */
   asm( "EI" );

   /*
    * Issue the "on" command to the AppPktHandler on ZAURA_RF_Dest
    */
   ZAURA_RF_Transmit( ZAURA_RF_Dest, SDATA | SAP_APP_0, "on", 2 );
   while( ZAURA_RF_GetState() == ZAURA_RF_TRANSMIT );

   OnCmd();

   /*
    * Reenable pushbutton interrupts
```

```
    */
   asm( "DI" );
   IRQ1ENL |= (SW1_PIN | SW2_PIN);
  }
}
```

The only difference between this ISR and the one used in the P1 project is that the target SAP address has been changed from `SAP_CMD_INTRPTR` to `SAP_APP_0`. This new address is necessary because the Shell Library and its command interpreter have been removed from the P2 project. If the P2 project was to send a command to `SAP_CMD_INTRPTR`, it would simply be ignored by any remote node(s) that received it. Because the data packet now targets an application-specific SAP, it is necessary for the application to include code to process this packet.

Packet Reception

The P2 project requires additional code to be added to the `AppPktHandler`. Previous implementations simply displayed the contents of received packets; however, because the ZAURA RF Shell Library is not included in the P2 project, it is not possible to display information about the console. Instead, the `AppPktHandler` must explicitly parse and act upon valid packets, as shown in the following routine.

```
void
AppPktHandler
(
  void
)
{
  ZAURA_RF_PKT_BUF* pPkt;

  /*
   * This routine is called when the application is able to check
   * for and process received packets. This routine should not be
   * called from within an interrupt handler.
   */
  pPkt = ZAURA_RF_Receive();
  if( pPkt )
  {
   /*
    * Verify packet destined for SAP_APP_0
    */
   if( (pPkt->Ctrl & ZAURA_RF_CTRL_SAP_MASK) == SAP_APP_0 )
   {
   /*
   * Process commands received from the control unit,
   */
   if( memcmp("on", pPkt->Data, 2) == 0 )
   {
   OnCmd();
   }
```

```
    if( memcmp("off", pPkt->Data, 3) == 0 )
    {
    OffCmd();
    }
    }

    ZAURA_RF_FreeBuf( pPkt );
   }
}
```

Because the P2 project is not required to perform any other processing functions, the main routine calls `AppPktHandler` from within a tight loop. When `AppPktHandler` calls `ZAURA_RF_Receive`, the ZAURA RF Wireless Library will process packets destined for the ZAURA Shell (the target SAP is either `SAP_CMD_INTRPTR` or `SAP_UART_0`). Any packets targeting a different service access point (or when the transfer format is `XFER_DA` or `XFER_DA_SA`) will be furnished to the application for processing upon return from the `ZAURA_RF_Receive` API call.

In the code segment above, the `AppPktHandler` routine first checks that the received packet is targeting `SAP_APP_0`, because the control unit directs its commands to `SAP_APP_0` when either SW1 or SW2 is pressed. Next, the packet handler checks to determine whether the received command is either `on` or `off`, and calls the appropriate routine to manipulate the state of LED D1.

Running the Project

Source code for the Application Packet Handling implementation is contained in the AN0336.zip file. To run the code, observe the following procedure:

1. Place the AN0336.zip files into the root of the ZAURA RF Wireless installation and extract the contents of this zip file into the AN0336 subfolder. This AN0336 subfolder should be a peer to the Demo folder.
2. Launch ZDSII and navigate to the `..\ZAURA_RF_Wireless_vX.YZa\AN0336` folder (in which *X.YZa* is the version of the ZAURA RF Wireless software installed; for example: *v1.10a*) and open the `p2.zdsproj` file.
3. Select the build configuration appropriate for the RF modules being used. This configuration will be either the `866p5_MHz` or `915_MHz` build option.
4. Next, select the **Build** → **Rebuild All** menu option.
5. Download the project using the **Tools** → **Flash Loader** option.
6. Repeat Step 5 for each RF Module that must be reprogrammed. Note the 1-byte *serial number* that is programmed into the first slave unit using the Flash Loader.
7. On the control unit, press SW1 (PD2) to cause LED D1 to illuminate on all slave units. Similarly, pressing SW2 (PD1) will cause all slave units to extinguish LED D1.

How it Works

When SW1 is pressed, the `on` command is broadcast to all ZAURA RF nodes within radio range of the transmitter. Similarly, when SW2 is pressed, the `off` command is broadcast to all remote nodes. An application-supplied packet handler retrieves the received packet from the ZAURA RF Wireless Library and determines whether it should illuminate LED D1, extinguish LED D1 or take no action.

Limitations

In the P2 project, the `query` operation is not implemented because the control unit only has two buttons and has no simple means to implement a third command. Additionally, without the Shell Library, the only way a specific slave unit can be controlled is by programmatically modifying the value of `ZAURA_RF_Dest` on the control unit.

Additional Enhancements

Because both the control and slave units use identical hardware, both units were programmed with the same firmware image. In an actual implementation, it is more common that different hardware platforms would be used for each type of unit. Typically, this scenario will require that separate project files be used for each different hardware platform; each project will likely require a unique GPIO configuration file (`GpioConfig.c`).

The projects described in this application note may not be suitable for battery-powered operation over an extended length of time, because after a unit is powered up, the radio remains in receive mode and the Z8F2480 processor runs continually. To reduce power consumption, the units could be placed into low-power modes when RF communication is not required (see the description of the `ZAURA_RF_SetState` and `ZAURA_RF_StopMode` APIs).

The projects described in this application note also use a common configuration file to set the default RF parameters (including channel number and network ID); however, this configuration may not be practical when deploying a product. For example, if the embedded device is configured to use Channel 1and is placed in an environment in which other devices are already using Channel 1 for some other purpose, then the ZAURA RF devices could interfere with the non-ZAURA RF devices, resulting in lost data. It this instance, it would be more useful if the ZAURA RF device automatically selected a quiet channel or periodically changed channels to reduce the likelihood of interference.

Hardware Configuration

All projects described in this application note use the ZAURA RF Wireless Module mounted on the ZAURA Validation Board.

Summary

This document describes three methods that can be used to implement the most basic RF control application in which a control unit must manipulate multiple remote slave units.

The first method used the ZAURA RF Wireless Demo project and the Shell Library's `port` command to manipulate a GPIO port pin that turns on/off an LED on the ZAURA Validation Board.

The second method used a project file from the source code that accompanies this application note to perform a similar operation using a custom Shell command. A `query` command was also added to allow the control unit to poll slave units for the state of their LEDs. This project also used the SW1 and SW2 pushbuttons on the ZAURA Validation Board as a means of turning on/off LED D1 on slave validation boards.

Instead of using the ZAURA RF Shell Library, the third implementation used a custom packet handling routine to parse commands received as a result of pressing SW1 and SW2 on the control unit.

Abbreviations, Acronyms and Definitions

Term	Description
ACK	Acknowledgement; an RF control frame sent in response to an SDATA frame indicating receipt of the SDATA frame. Acceptance of the frame is determined by the value of the sequence number field in the ACK frame.
Broadcast	Any frame transmitted to the 0xFF address; a broadcast frame will be received by all ZAURA RF nodes within radio range of the transmitter that are in the same cell. Broadcast frames are always sent unreliably.
Cell	A collection of ZAURA RF nodes with the same network ID and frame format that operate on the same RF channel.
FSK	Frequency Shift Keying; a modulation scheme that transmits information through discrete changes of a carrier signal. The ZAURA RF Wireless Module uses binary FSK, in which a binary 0 is transmitted at a frequency of *carrier – delta* and a binary 1 is transmitted at a frequency of *carrier + delta*.
Pause Delay	ACK packets can optionally request a transmitter to delay transmission of the next directed data frame to the sender of the ACK by up to 400ms, in increments of 25ms.
RDTFC	Reliable Data Transfer & Flow Control; the ZAURA RF Shell Library can detect lost packets and automatically perform retransmission. Receivers are able to detect and discard duplicate packets. Slow receivers can throttle fast transmitters through the use of pause delays.
RF	Radio Frequency; the ZAURA RF Shell Library operates within the 863-870MHZ or 902-928MHz ISM band. There are two versions of the ZAURA RF Shell Library – one for each frequency band.
SDATA	Sequenced data; the ZAURA RF Shell Library implements a simple reliable data transfer and flow control protocol based on the exchange of sequenced data frames and ACK frames. All SDATA and ACK packets include a sequence number.
Unicast	Any frame transmitted to an individual ZAURA RF Shell Library address in the range of 0x01 to 0xFE.

References

The following documents describe functional specifications and/or otherwise support this application note. With the exception of the Semtech Transceiver data sheet, each is available for download from the Zilog website.

ZAURA RF Wireless Modules Reference Design Document (PUG0030)

ZAURA RF Wireless Library Reference Manual (RM0060)

ZAURA RF Module Shell User Manual (UM0235)

Semtech SX1211 Transceiver Datasheet

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.

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

 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.