AN906

RF4CE ZigBee Remote Control Firmware Guide

This document is intended for firmware engineers working with Silicon Labs RF4CE ZRC (ZigBee Remote Control) Low Cost Remote and Full Featured Remote reference designs. It explains the main firmware structures and how to customize them.

New in This Revision

Initial release

Contents

1 Introduction 3
2 Remote Control Reference Design Overview 3
3 Controller Implementation 4
3.1 Devices 4
3.1.1 Pairing with Multiple Targets 5
3.1.2 Using the TV Device for IR Control 5
3.2 Infrared Driver 5
3.3 Accelerometer Motion Detection 5
3.4 Voice Compression and Streaming 5
3.4.1 Hardware Audio Codec 6
3.4.2 Software Audio Codec 7
3.5 Key Matrix Driver 8
3.5.1 Key Detection 9
3.5.2 Pin Sharing 9
3.6 Identification Message 9
3.7 Action Mapping 9
3.8 Storage 9
3.9 Power Consumption 10
4 Target Implementation 10
5 Target-to-Host Protocol 11
5.1 Protocol Layers 11
5.2 Protocol Operation 12
5.3 Presentation Layer 12
5.3.1 Message Format 12
5.3.2 Framing 12

5.3.3 Checksum 12
5.3.4 Escaping 12
5.3.5 Example 13
5.4 Application Layer 13
5.4.1 Message Format 13
5.5 Application Layer Messages Defined 13
5.5.1 Message Payload 13
5.5.2 Message Id 18

1 Introduction

This application note relates to the Low Cost Remote reference design (RD43; Figure 1A) included in the EM34x Development Kit (EM34X-DEV), and the Full Featured Remote reference design (RD17; Figure 1B) included in the EM34x Voice Remote Control Development Kit (EM34X-VRDK) and the EM34x Voice Remote Control Evaluation Kit (EM34X-VREVK).

A) B)

Figure 1. A) Low Cost Remote Control; B) Full Featured Remote Control

The following documentation is recommended for firmware engineers starting with Silicon Labs ZigBee Remote Control applications:

UG103.10	*Application Development Fundamentals: RF4CE*
QSG100	*EM34x Development Kit Quick-Start Guide*
UG113	*EM34x Development Kit User Guide*
QSG103	*EM341 Voice Remote Development Kit Quick Start Guide*
QSG104	*EM341 Voice Remote Evaluation Kit Quick Start Guide*

2 Remote Control Reference Design Overview

The Silicon Labs ZRC reference designs implement the following topology (Figure 2):

- Controller: This is the hand-held wireless remote control device
- Target: This device communicates wirelessly with the Controller over the RF4CE network
- Host: This device communicates with the Target over a wired serial connection using the target-to-host protocol described in this document.

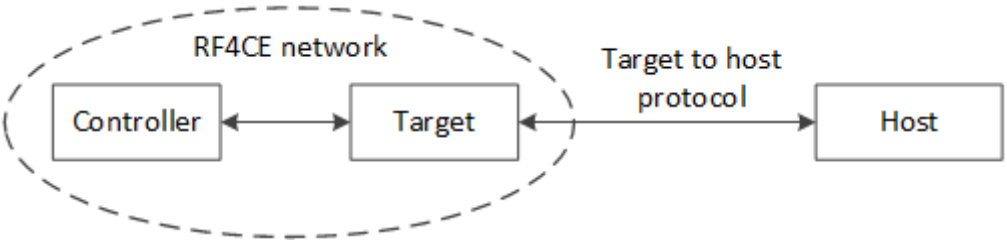

Figure 2. RF4CE Remote Control System Topology

The Host is usually the main processor in a set-top box or other RF4CE-compliant equipment. In this setting the Target device is the embedded processor inside the set-top box. For the Silicon Labs remote control reference designs the Host device is a PC. The Target device is a USB dongle or development kit that communicates with the Host via a serial connection or USB. As the target-to-host protocol is implemented in the Target firmware using a UART or USB CDC connection, the target firmware can easily be reused in designs where the Target is embedded in a set-top box.

The Silicon Labs RF4CE ZRC Full Featured Controller and Low Cost Controller support the following features:

- Key matrix scanning
- ZRC 2.0 action commands sent over RF4CE when buttons are pushed
- Infrared LED for controlling legacy equipment
- Power button LED (dual color on Low Cost Controller and single color on Full Featured Controller)
- Over-the-air action mapping to set actions for keys to user-defined ZRC2.0 action commands and/or IR codes
- Over-the-air identify message to locate Controller via blinking LED

In addition, the Full Featured Controller includes the following features:

- Buzzer to provide audio feedback
- Backlight LEDs for key matrix
- Accelerometer to turn on Backlight when controller is moved
- Streaming voice audio to Target (and Host) when Voice button is pushed
- Over-the-air identify message to locate Controller via buzzer sound

3 Controller Implementation

Firmware examples for the EM341 RF SoC in the Low Cost and Full Featured Controllers are included as RF4CE sample applications in the EmberZNet stack release.

3.1 Devices

Both of the remote controls support for two devices, called TV and STB. The devices are selected using the TV and the STB buttons on the controllers. The controller firmware is configured to support pairing with up to two targets that can be mapped to the STB or TV devices. The targets do not necessarily have to be a television and a set-top box, even though the text on the buttons and the device names indicate this.

To pair a controller and target, first press the pairing button on the target (or host sending the Bind Request as described in section 5.5.1.8) to allow a controller to pair with it. Then, while holding down the PAIRING/SET key on controller, press one of the device keys (TV or STB). The power button LED will be lit to indicate that a pairing is ongoing. When the target accepts the pairing, the LED will be turned off. If the controller does not find a target to pair with it will stop trying after about 45 seconds and turn off the LED.

3.1.1 Pairing with Multiple Targets

When the controller is paired with two targets, the user first selects the target to communicate with by pressing the associated device button (TV or STB). When the user then presses a key mapped to a specific ZRC 2.0 action command, this command is sent to the target paired as the selected device.

The first target that is paired to a controller is paired as both the TV and STB device. When the second target is paired, it is paired as the device corresponding to the button used in the pairing procedure.

3.1.2 Using the TV Device for IR Control

A common configuration is only to have one RF4CE-enabled device (e.g. set-top box) and a TV that only supports legacy IR signals. In such a configuration, the set-top box can be used to program the action mappings on the controller to allow it to send IR signals to the TV. In the reference firmware, IR signals are only supported for the TV device. Both the TV and the STB device must be paired with the set-top box target to allow the set-top-box to update IR action mappings on the controller.

3.2 Infrared Driver

The infrared driver included in the reference firmware supports two IR database formats, the ZRC2.0 SIRD (Standard Infrared Database) format and the UIRD (Universal Infrared Descriptor) format (both regular and encrypted). The IR codes can be transferred to the controller as action mapping updates matching the mappable actions that the controller supports. As SIRD codes are native to the ZRC2.0, this format is expected for IR descriptors unless a vendor ID is provided. For the UIRD codes the following dummy vendor IDs are used:

- 0x01: Regular UIRD
- 0x02: Encrypted UIRD

Note that the 32-bit encryption key must be obtained from the UIRD database provider and applied as a hash-define called UIRD_DECRYPTION_KEY. The encryption UIRD encryption algorithm is not available as source code, only as a pre-compiled library.

3.3 Accelerometer Motion Detection

The Full Featured Controller includes a 3-axis accelerometer that is used to detect motion in any direction. The accelerometer is programmed over I2C to sample the 3-axis acceleration data every 50 ms and set the interrupt output to the EM341 if the acceleration exceeds the configured threshold. After receiving an accelerometer interrupt output, the EM341 will turn on the eight key pad backlight LEDs and keep them on until the controller has been still for at least 5 seconds.

3.4 Voice Compression and Streaming

The Full Featured Controller includes a digital PDM (pulse density modulation) microphone to record voice audio through an audio inlet at the top of the remote, between the POWER and SET buttons. The voice system is designed for speech recognition to enable voice control of the host from the controller.

The digital PDM microphone converts the audio input to a 1-bit digital output in PDM format. In PDM format, the density of digital ones increases or decreases with increased or decreased audio pressure into the microphone. Since PDM is a digital format, the microphone can be placed a reasonable distance from the hardware codec without impacting the audio performance, which might occur with an analog microphone.

The Full Featured Controller is implemented in two versions. The first uses a hardware codec for processing the microphone audio before it is received by the EM341. The second does all of the audio processing internally to the EM341. The software audio codec has the lowest BOM cost, but uses more EM341 CPU cycles than the hardware audio codec. The EM34X-VRDK includes the hardware codec solution. For information on how to obtain the software codec, see www.silabs.com/zrctraining.

3.4.1 Hardware Audio Codec

The PDM 1-bit data stream is converted to a 16-bit PCM (pulse code modulation) by passing the PDM data through a decimation filter. To remove the microphone's DC component, the codec's 1st order IIR filter is also configured as a high-pass filter. To compensate the audio frequency response for artifacts induced by implemented plastics and audio waveguide, the codec's multiple IIR bi-quads can be configured as an equalizer. In the Full Featured Controller firmware, the codec's IIR bi-quads are not enabled.

Figure 3 shows an overview of the voice front-end.

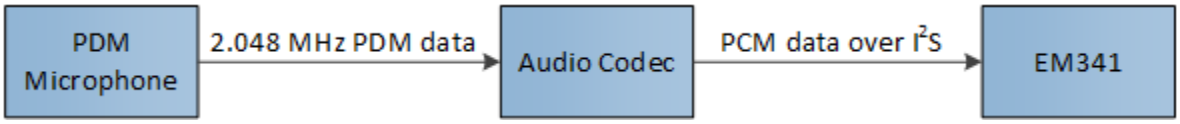

Figure 3. Controller Hard Codec Voice Front-End

When the key matrix driver detects that the VOICE button is pressed, the audio streaming is set up by the following steps:

1. Key matrix shared GPIOs are reconfigured for the hardware audio codec.
2. Hardware audio codec is powered up by asserting a GPIO.
3. Timer 2 (TIM2) is configured to drive the master clock to the hardware audio codec.
4. Serial Controller 1 (SC1) is configured as an SPI slave to accept I2S mono audio.
5. Serial Controller 0 (SC0), already in I2C mode, is used to configure the hardware codec.
   - Set up codec's PLL and clock tree
   - Enable digital PDM microphone operation
   - Set up decimation and digital filters
   - Set digital gain
   - Enable 16-bit (PCM) I2S mono voice audio data output @ 16 kSample/s.
6. The filtered PDM to PCM I2S captured data is compressed and streamed to the paired target by the controller.

The controller will continue streaming voice data to the target that is paired with the current device selection (TV or STB) as long as the VOICE button is held down. The target device receives the voice data and relays this to the host. When the VOICE button is released:

1. Timer 2 (TIM2) is reset.
2. Key matrix shared GPIOs are reconfigured for the key matrix.
3. Hardware audio codec is powered down by de-asserting a GPIO.

ZigBee (and RF4CE) has a max bitrate of 250 kbps. However, due to overhead and real-world RF interference, the highest practical sustained throughput with the EmberZNet stack is about 80 kbps. The voice audio is recorded as 16-bit samples @ 16 kSamples/s, which gives a bit rate of 256 kbps. To fit the audio within the allowed bit rate over ZigBee, a 4:1 MS ADPCM compression is applied, resulting in 64 kbps compressed voice data. The 64 kbps compressed voice data is then sent over RF4CE using a vendor-specific profile. This profile allows a voice payload of 103 bytes to be sent with each message. The payload consists of the following elements:

Bytes	Contents
0	Message sequence number (increased for each message up to 127 before wrapping to 0)
7:1 (7 bytes)	MS ADPCM compressor state variable
102:8 (95 bytes)	Compressed audio

The compressor state variable contains 2 uncompressed 16-bit audio samples. Together with the 95 bytes of compressed audio (2 samples per byte), a total of 192 PCM samples are transmitted with each message. To keep up with the continuous 16 kSamples/s audio stream, a compressed audio message is sent every 12 ms from the controller to the target. For details on the microphone driver implementation, see the comments at the beginning of the driver source code.

Using the MS ADPCM compressor state contained in each audio message, the target (or Host) decompresses the compressed audio stream to 16-bit PCM audio data @ 16 kSamples/s. The PCM audio data is then sent to the speech recognition engine, implemented in the host or via Cloud. The PCM audio quality far exceeds the performance required by leading speech recognition engines.

3.4.2 Software Audio Codec

The PDM 1-bit data stream is converted to a 16-bit pulse code modulation (PCM) by passing the PDM data through a decimation filter implemented in software. To remove the microphone's dc component, a 1st order IIR filter implements a high-pass filter. To compensate the audio frequency response for artifacts induced by implemented plastics and audio waveguide, four user configurable IIR bi-quads can be configured as an equalizer. In the Full Featured Controller firmware, the software codec's IIR bi-quads are not enabled.

Figure 4 shows an overview of the software audio codec voice front-end.

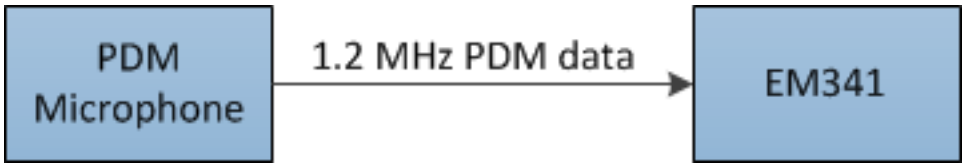

Figure 4. Controller Soft Codec Voice Front-End

When the key matrix driver detects that the VOICE button is pressed, the audio streaming is set up by the following steps:

1. Key matrix shared GPIOs are reconfigured for the software audio codec.
2. Software audio codec is powered up by asserting a GPIO.
3. Serial Controller 1 (SC1) is configured as a SPI master to accept PDM mono audio into DMA buffers.
4. When a DMA buffer is filled, the software codec processes the buffer's 1-bit/1.2 MSps PDM data into 16-bit/16 kSps PCM data by:
   - Applying decimation filter
   - Applying user configurable four IIR equalization biquads
   - Applying high-pass filter to remove dc component
   - Applying digital gain to achieve -12 dBFS at 94 dBSPL
5. The filtered PDM to PCM captured data is compressed and streamed to the paired target by the controller.

The controller will continue streaming voice data to the target that is paired with the current device selection (TV or STB) as long as the VOICE button is held down. The target device receives the voice data and relays this to the host. When the VOICE button is released:

1. Serial Controller 1 (SC1) is reset.
2. Key matrix shared GPIOs are reconfigured for the key matrix.
3. Software audio codec is powered down by de-asserting a GPIO.

ZigBee (and RF4CE) has a max bitrate of 250 kbps. However, due to overhead and real-world RF interference, the highest practical sustained throughput with the EmberZNet stack is about 80 kbps. The voice audio is recorded as 16-bit samples @ 16 kSamples/s, which gives a bit rate of 256 kbps. To fit the audio within the allowed bit rate over

ZigBee, a 4:1 IMA ADPCM compression is applied, resulting in 64 kbps compressed voice data. The 64 kbps compressed voice data is then sent over RF4CE using a vendor-specific profile. This profile allows a voice payload of 104 bytes to be sent with each message. The payload consists of the following elements:

Bytes	Contents
0	Message sequence number (increased for each message up to 127 before wrapping to 0)
3:1 (3 bytes)	IMA ADPCM compressor state variable
103:4 (100 bytes)	Compressed audio

With the 100 bytes of compressed audio (2 samples per byte), a total of 200 PCM samples are transmitted with each message. To keep up with the continuous 16 kSamples/s audio stream, a compressed audio message is sent every 12.5 ms from the controller to the target.

Using the IMA ADPCM compressor state contained in each audio message, the target (or Host) decompresses the compressed audio stream to 16-bit PCM audio data @ 16 kSamples/s. The PCM audio data is then sent to the speech recognition engine, implemented in the host or via Cloud. The PCM audio quality far exceeds the performance required by leading speech recognition engines.

3.5 Key Matrix Driver

The key matrix driver in the reference firmware is set up to scan a 7x7 key matrix, connected to 14 GPIO pins (7 row pins and 7 column pins). However, it can be configured to fit other keyboard layouts. An example 4x4 key matrix is shown in Figure 5.

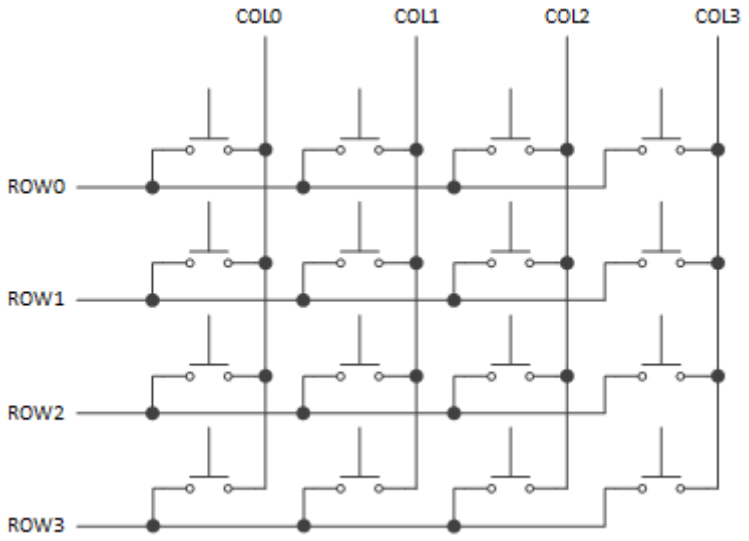

Figure 5. Example 4x4 Key Matrix

In Deep Sleep mode all the columns are configured as inputs with internal pull-down resistors enabled. The columns' GPIO pins are also set up to wake up the device when it detects a change in the logic input value of the pin. The rows are all configured to be high output push-pull. When a key is pressed, the row pin drives the associated column pin high through the closed circuit and the device wakes up because of the input pin change.

3.5.1 Key Detection

When the device wakes up, the driver is then called to figure out which key was pressed. This is done by tri-stating all row pins except for the one that is being driven high. Then the state of each of the columns is recorded, before the process is repeated with the next row being driven high. At the end of this process the driver has recorded all buttons that were pressed during the scan. The application can later read out the latest recorded key states and perform the appropriate actions based on them. When no key presses are detected, the GPIOs are set in their sleep configuration before the device enters Deep Sleep. In this example the rows are used as outputs and driven high, but both the drive direction and active level can be configured with defines.

Note that the Low Cost Controller and Full Featured Controller hardware do not include any reverse flow protection diodes for the switches. As such, ghosting (i.e., an incorrect key detection) can occur if more than two keys are pressed simultaneously. The key matrix driver would, however, be compatible with hardware designs that include diodes for full N-key rollover support.

3.5.2 Pin Sharing

For both of the controller designs, the full 7x7 key matrix uses 14 GPO pins. To maximize functionality with the available GPIO pins, the key matrix pins are shared with other functions. For both controller designs, the debug header shares pins with the key matrix and, on the Full Featured Controller, the audio front-end control also shares pins.

Some of the debug pins are re-used as row and column pins. To support debugging, the key matrix driver checks if the device is in a debug session during the initialization and skips the key scanning for any columns or rows that are used as debug pins. This means that some keys will not work when a debug adapter is attached. When the device is disconnected from the adapter and power-cycled, all pins will function normally again. The masks for the debug pins are set in the board header file.

For the Full Featured Controller, key matrix GPIO pins are also used to support the audio front-end, both for the soft and hard audio codec solutions. The VOICE button is intentionally placed on a row and column that are not shared with the audio front-end. When VOICE button is pressed, the GPIOs used for the audio front-end are reconfigured for the audio interface and remain in that configuration during audio streaming. The VOICE button is constantly monitored and, when released, the audio streaming is stopped and the GPIOs used for the audio front-end are reconfigured for the key matrix.

3.6 Identification Message

To locate a misplaced controller, the target can send it an identification message. When the controller receives this message it will beep the buzzer and blink the backlight LED 5 times, hopefully allowing the user to find it.

3.7 Action Mapping

The controller reference firmware is set up to associate a default RF4CE ZRC2.0 action command or an IR command for each button on the remote. Using ZRC2.0 action mappings, the controller can be programmed by the target to perform new actions when a key is pressed. In the controller reference firmware, most of the buttons on the controller are configured as a mappable action. The list of the supported mappable actions can be seen (and modified) in the RF4CE tab for your RF4CE project in Ember Desktop.

Action mappings are selected with the demo PC application and sent to the target. The target then waits for an incoming heartbeat notification from any of the controllers it is paired with. When a heartbeat is received, the target sends the new action mappings to the controller.

3.8 Storage

The remote control firmware stores the pairing information in simulated EEPROM using tokens. This allows the controller to remember its paired devices through a power cycle.

Any programmed action mappings are stored in RAM and are lost during a power cycle. To regain the action mappings automatically, the controller requests action mappings from all paired targets after it powers up. This allows changing batteries on the controller without having to re-pair with the target(s) or re-program the action mappings manually.

Statically-allocated tables are used to store the action mappings for the mappable actions supported by the controller. If the action mappings include variable-length data like IR descriptors, the variable-length data is stored in a RAM heap memory. The amount of RAM allocated for the action mapping heap is by default 1300 bytes and can be configured in the ZigBee Remote Control 2.0 Action Mapping Client plugin settings in Ember Desktop.

3.9 Power Consumption

The controller firmware is set up to enter Deep Sleep mode whenever no buttons are pressed. In this mode the controller consumes approximately 1.5 µA. The Full Featured Controller includes an accelerometer that measures movement every 50 ms and turns on the button backlight for 5 seconds if motion is detected. During Deep Sleep mode the accelerometer adds on average 5.5 µA.

The controller and target implement an asymmetric protocol where the target can only send data to the controller immediately following a heartbeat notification sent out by the controller. The controller, on the other hand, can send data to the target at any time. This is done to allow the controller to keep its RF receiver disabled and go to Deep Sleep as much as possible to conserve battery life. In a resting scenario where no RF4CE communication is done by the application (key presses, action mapping updates etc.) the only time the controller is awake is to send periodic heartbeat notifications to the target and listen for a response for a while before it goes to sleep. By default the controller firmware is set up to send a heartbeat to every paired target every 5 seconds. The energy consumed by the controller during a heartbeat cycle (with no response from the target) is about 1.5 mJ (@3 V). This equates to an added average consumption of 100 µA for each paired target with the default 5-second heartbeat interval. Increasing the heartbeat interval can therefore save power, but at the expense of longer worst-case latency when the target wants to start communication with the controller (e.g. when programming action mappings and sending identification messages).

4 Target Implementation

The EmberZnet stack release includes an RF4CE sample application called Full Featured Target that supports both the Low Cost Controller and the Full Featured Controller. See UG113, *EM34x Development Kit User's Guide*, for instructions on how to create a new application based on this.

The Full Featured Target sample application can be configured to run on the following target devices:

- EM346 breakout board in the EM34x Development Kit. With this setup the serial connection to the PC is done either over USB-CDC or the RS-232 port on the breakout board.
- EM3588 Telegesis USB stick. This device implements a USB Communication Device Class (CDC) which is used to communicate with the PC.

Both targets have a button for allowing incoming pairing requests. In the EM34x Development Kit, BUTTON1 is used for this. Both targets also support pairing requests from a host sending the Bind Request as described in Section 5.5.1.8, **Bind Request**.

The target can be paired with up to five controllers. If a pairing attempt is made when the pairing table is full the pairing table is erased and the new pairing is added.

Two statically-allocated tables are used to store data, one for the mappable actions that are read from the paired controllers, and the other for storing the action mappings to remap the mappable actions. The size of both tables is set to 50 bytes for the EM346 breakout board and 100 bytes for the EM3588 USB stick. If the action mappings include variable length data like IR descriptors, this is stored in a RAM heap memory. The amount of RAM allocated for the action mapping heap is by default 700 bytes for the EM346 and 4000 bytes for the EM3588. The

size for the mappable actions, remap table, and the heap memory can be configured in the ZigBee Remote Control 2.0 Action Mapping Server plugin settings in Ember Desktop.

5 Target-to-Host Protocol

The ZigBee Remote Control profile, ZRC, specifies the communication between the ZRC Originator and Recipient, which are the Controller and Target in this case. However, it does not specify how the communication should be utilized at the application level. The Target-to-Host Protocol (THP) defines how required information is communicated between the application at the Target and the Host computers in the system.

This specification assumes that the THP will be transported over a full duplex serial interface. It can be transported over either a UART interface or a USB port implementing a CDC (Communication Device Class) that emulates a UART.

5.1 Protocol Layers

The OSI Model (Open Systems Interconnection Basic Reference Model) is an abstract description for layered communications and computer network protocol design. The OSI Model divides network architecture into seven layers which, from top to bottom, are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layers.

The RF4CE remote control target application and the host computer use a set of protocols to send their data down the layers, encapsulating the data at each level. The THP consists of two layers in the OSI Model: the application layer and the presentation layer. The session/transport layers are represented by the serial communication.

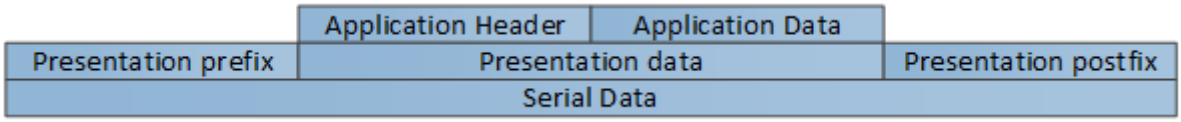

Figure 6. Protocol Layers

The Application header and data are a message in the application layer. The presentation layer prefix, data and postfix are a messages in the presentation layer. The data part of the layer is called the payload.

The RF4CE target will be connected to the host computer using the layers shown in Figure 7.

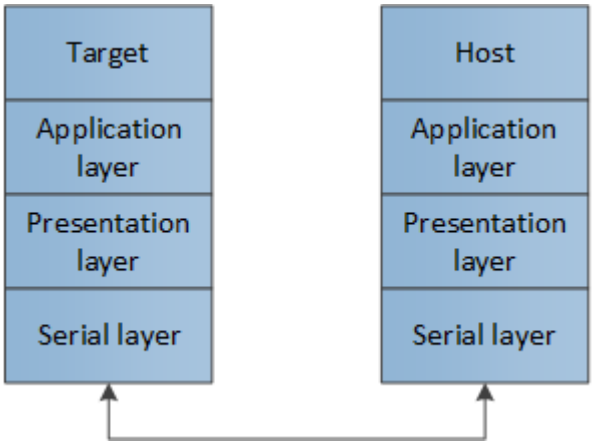

Figure 7. Communication Layers

5.2 Protocol Operation

The protocol operation at the application layer between the Target and Host is connectionless. A message is sent, and a response may be sent back depending on the Message id.

Unless otherwise stated, the protocol assumes that all communication requests are started by the target and sent to the host computer. The host computer cannot send anything to the target without first being asked to do so. In order to let the host make the target aware that it has information for it, the target should regularly poll the host for status. The host may reply with a `Check Condition` status and one of the "attention" codes in the "Conditional" field. When the condition has been read, it may be reset by the host if it no longer applies. One example is the reset/power on that normally will be read only once, unless the host is reset during operation.

5.3 Presentation Layer

The presentation layer ensures that a start of a message can be found. Numbers with prefix "0x" below are given in hexadecimal notation.

5.3.1 Message Format

The message format for the presentation layer is shown below:

StartFrame	Payload	Checksum	EndFrame

The StartFrame and EndFrame are the Presentation Prefix and Presentation Postfix respectively.

Field description:

Size	Range/Value	Name	Description
1	0xC0	StartFrame	The start frame unique identifier.
0..n		Payload	The payload.
1		Checksum	8-bit checksum.
1	0xC1	EndFrame	The end frame unique identifier.

5.3.2 Framing

Each message begins with the byte 0xC0 and ends with the byte 0xC1. In between the framing bytes are the payload and the checksum.

5.3.3 Checksum

The last byte of a message (without the framing) contains an 8-bit checksum. The checksum is calculated over the entire message, without framing and without including the checksum byte. The checksum is obtained by XORing one and one byte.

5.3.4 Escaping

To make sure that 0xC0 and 0xC1 are not contained within the payload an escape sequence is used. The value 0x7E is used to XOR the following byte with the value 0x20. This is used for the following three combinations:

- 0x7E 0x5E is used for 0x7E
- 0x7E 0xE0 is used for 0xC0
- 0x7E 0xE1 is used for 0xC1

5.3.5 Example

The “0x” prefix is omitted for readability in the example below.

- Payload: 00 C0 05
- Add checksum:00 C0 05 C5
- Add escaping: 00 7E E0 05 C5
- Add framing: C0 00 7E E0 05 C5 C1

Note that the 0xC0 of the payload is replaced with the 0x7E 0xE0 escaping sequence.

5.4 Application Layer

The THP is defined as variable length packets that are transferred between the Target and the Host.

5.4.1 Message Format

The message format for the application layer is shown below:

Application Header	Application Data

The Application Header is mandatory. Some messages may consist of an application header only. The Application Data is the payload of the application layer message.

Application Header format

Size	Range/Value	Name	Description
1	0	Version	Version of the message format.
1	See 5.5.2	Message Id	Message type identifier. Each message type has a unique message id.
1	[0,255]	Data Length	The length of the Application Data.

Application Data format

Size	Range/Value	Name	Description
0..101		Application Data	Payload of the message.

5.5 Application Layer Messages Defined

The application message format is defined in section 5.4, **Application Layer**, and is used for all messages that are transported between the target and the host. Because some of the payloads may be used for more than one Message Id, the payloads are defined first. Next the different Message Id’s are defined.

Messages can be sent either way and this document the following notation is used:

Requests: Messages going from the target to the host.
Acknowledges: Messages going from the host to the target.

5.5.1 Message Payload

The following sections defines the different payloads. Please refer to section 5.5.2, **Message Id,** to find which messages are using the different payloads.

5.5.1.1 Get Status

Request status from the host.

Status – request

Size	Range/Value	Name	Description
1	0	VersionMajor	Target protocol version major number.
1	0	VersionMinor	Target protocol version minor number.
0..1		Status	Status from the target to the host. The target may notify the host by adding the Status and Conditional fields.
0..1		Conditional	

Status - acknowledge

Size	Range/Value	Name	Description
1	0	VersionMajor	Host protocol version major number.
1	0	VersionMinor	Host protocol version minor number.
1		Status	The host status.
1		Conditional	The value reported when the Status is Check Condition.

Status definitions

Status	Description
0	Status OK
1	Check Condition
2	Busy

Conditional definitions

Conditional	Description
0	Condition OK
1	Undefined
10	Not ready
20	Hardware error
30	Illegal request
40	The device has been reset or off
41	Action mappings have changed
42	Identify controller

5.5.1.2 Action

Whenever a key is pressed on the controller, and the key is mapped to an RF-action, the target will receive the corresponding action code. The target forwards the code to the host by using this Action package.

Action – request

Size	Range/Value	Name	Description
1		ActionType	
1		ActionModifier	
1		ActionBank	
1		ActionCode	
2		VendorId	

Action – acknowledge

No acknowledge.

5.5.1.3 Action Mapping

The target may at any time request action mappings from the host, but a sequence is typically initiated by the host signaling the target that new mappings are available by replying with Status=`Check Condition` and Conditional=`Action mappings have changed`.

One action mapping can be requested at a time. Reading all action mappings from the host will require repeated requests. The action mapping message sequence is shown in Figure 8.

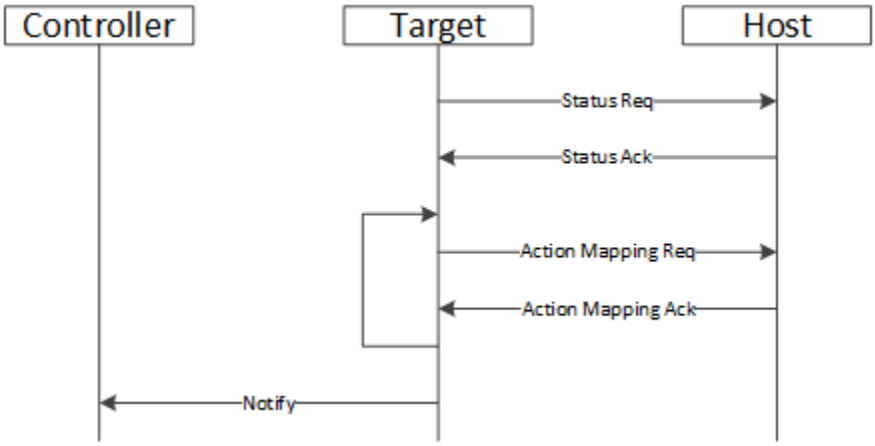

Figure 8. Request Action Mappings Message Sequence Chart

Action Mapping – request

Size	Range/Value	Name	Description
1		Device Type	
1		Bank	
1		Action	The action code to map from.

Action Mapping - acknowledge

Size	Range/Value	Name	Description
1		Device Type	
1		Bank	
1	[0x00, 0x7f]	Action	The action code to map from.
1		MappingFlags	
1		RF Data Length	The number of RF data bytes that are included in this message.
1		IR Data Length	The number of IR data bytes that are included in this message.
0..m		RF Data	If present, the RF data.
0..n		IR Data	If present, the IR data.

RF data format

Size	Range/Value	Name	Description
1		Config	
1		Option	
3		RF action mapping data	

RF action mapping data format

Size	Range/Value	Name	Description
1		Not used	
1		Bank	
1		Action code	

IR action data format

Size	Range/Value	Name	Description
1		Config	
2		Vendor	
n		IR action mapping data	IR action mapping data according to the format defined by the application.

IR action mapping data format

One actual format can be the one defined in the ZigBee RF4CE: ZRC Profile Specification Version 2.0, Annex B: Standard IR Database format, but other formats can be used as long as the host and controller agrees.

5.5.1.4 Audio data

Send compressed audio data (from target to host).

Audio data – request

Size	Range/Value	Name	Description
101		Audio Data	The audio data as received from the controller to the target.

Audio data – acknowledge

No acknowledge.

5.5.1.5 *Heartbeat*

This is a message that is send for every heartbeat the target receives from the controller.

Heartbeat – request

Size	Range/Value	Name	Description
1		Pairing Index	The pairing index for the heartbeat.
1		Triggers	Additional triggers for the heartbeat.

Heartbeat – acknowledge

No acknowledge.

5.5.1.6 *Identify*

The target can signal that the controller should blink and beep for a defined period. This is normally initiated by the host signaling the target that an Identify is in progress by setting Status=`Check Condition` and Conditional=`Identify controller`. The Identify message sequence is shown in Figure 9.

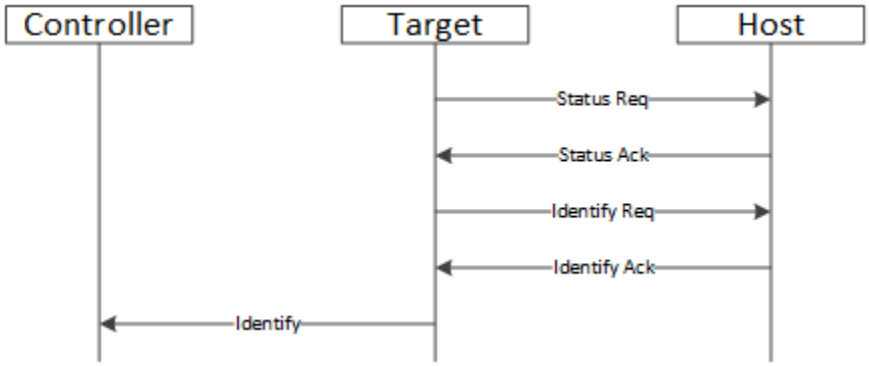

Figure 9. Identify Controller Message Sequence chart

Identify – request

No parameters.

Identify – acknowledge

Size	Range/Value	Name	Description
1		Flags	Identify flags, according to RF4CE: 0x01 = stop on action 0x02 = flash light 0x04 = sound 0x08 = vibrate (not supported by HW)
1		Time	The time in seconds the controller shall perform the identification.

5.5.1.7 Bind Information

This is a message sent from the target to the host computer when there is an event related to a bind process on the target.

Bind Information – request

Size	Range/Value	Name	Description
1	See table below.	Bind Info	A byte describing the bind event.

Bind Info definitions

Value	Name	Description
0	Init	A binding is requested by either pushing the button or a Bind Request is sent to the target.
1	Success	The bind operation was successful.
2	Failure	The bind operation failed.
3	Attempt	A controller has initiated a bind operation.
4	Proxy	

Bind Information – acknowledge

No acknowledge.

5.5.1.8 Bind Request

This is a message that can be sent from the host computer to the target to initiate a bind on the target. On some targets this operation can be done by pressing the bind button, but on others no button is available. Sending a bind request message is identical to pushing the bind button on the target.

Bind Request – request

No request.

Bind Request – acknowledge

No parameters.

5.5.2 Message Id

The Message Id is described in the following table:

Id	Name	Payload in section
0	Get Status Req	5.5.1.1 **Get Status**
1	Get Status Ack	5.5.1.1 **Get Status**
10	Action Req	5.5.1.2 **Action**
14	Action Mapping Req	5.5.1.3 **Action Mapping**
15	Action Mapping Ack	5.5.1.3 **Action Mapping**
20	Audio Data Req	5.5.1.4 **Audio data**
30	Heartbeat Req	5.5.1.5 **Heartbeat**
40	Identify Req	5.5.1.6 Identify
41	Identify Ack	5.5.1.6 **Identify**
50	Bind Info Req	5.5.1.7 **Bind Information**
53	Bind Request Ack	5.5.1.8 **Bind Request**

CONTACT INFORMATION

Silicon Laboratories Inc.

400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

For additional information please visit the Silicon Labs Technical Support page: http://www.silabs.com/support/