Z-Stack Lighting Developer's Guide

Document Number: SWRA419

Texas Instruments, Inc.
San Diego, California USA

Version	Description	Date
1.0	Initial release	1/4/2013
1.1	Added section 5.5 on Application Selective Target Touch-Link. Editorial changes.	2/21/2013
1.2	Added Sample Bridge application and Thermal Shutdown feature. Updated workspace and files structure and platforms support.	8/30/2013

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1 Purpose

This document explains some of the components of the Texas Instruments Z-Stack Lighting and their functionality. It explains the configurable parameters in Z-Stack Lighting and how they may be changed by the application developer to suit the application requirements.

1.2 Scope

This document enumerates the parameters required to be modified in order to create a ZigBee Light Link compatible product, and the various ZigBee Light Link related parameters that may be modified by the developer. This document does not provide details of other profiles and functional domains, nor the underlying Z-Stack infrastructure. Furthermore, this document doesn't explain the ZigBee Light Link concepts.

1.3 Definitions, Abbreviations and Acronyms

Term	Definition
API	Application Programming Interface
MT	Z-Stack's Monitor and Test Layer
OSAL	Z-Stack's Operating System Abstraction Layer
RSSI	Received Signal Strength Indication
Sub-device	A self contained device functionality (e.g. dimmable light), resides in a ZLL device a device application on an endpoint.
Touch-link	A method of finding and pairing devices nearby, based on received signal strength. Its operation is composed of device discovery and network settings transfer, and is defined in the ZLL specification.
ZCL	ZigBee Cluster Library
ZLL	ZigBee Light Link

1.4 Applicable Documents

1. ZigBee document 11-0037, ZigBee Light Link Profile Specification
2. ZigBee document 11-0039, ZigBee Light Link Profile: Test Specification
3. ZigBee document 11-5398, ZigBee Light Link Security Agreement
4. Texas Instruments document SWRA418, Z-Stack Lighting API
5. Texas Instruments document SWRU334, Z-Stack Lighting Sample Application User's Guide

2. Overview

2.1 Introduction

The Z-Stack Lighting is based on Z-Stack Core ZigBee stack, which is fully compliant with ZigBee 2012 (r20) PRO feature set. Its ZLL functionality and features were enabled by enhancements and additions to the stack's upper layers.

2.2 Software Architecture

The design of Z-Stack Lighting encapsulates the management of the touch-link commissioning process within the profile layer, beyond the scope of the application layer. This follows the ZLL specification's definition that the commissioning cluster should not to be a part of a specific sub-device, and it also allows the user to focus his application's implementation on the operational functionality as much as possible.

All the touch-link Inter-PAN traffic is managed by a dedicated internal endpoint, acting as an initiator or as a target, according to the device type. For ZLL commissioning, a lighting device application should use only the API defined in *zll_target.h*, while a controller device application should use only the API defined in *zll_initiator.h*.

Each endpoint implemented in the application layer, which registers for ZLL services from the profile layer, is considered as a sub-device in ZLL commissioning context.

A full description of the APIs between the Application and the ZLL Profile can be found at Z-Stack Lighting API [4].

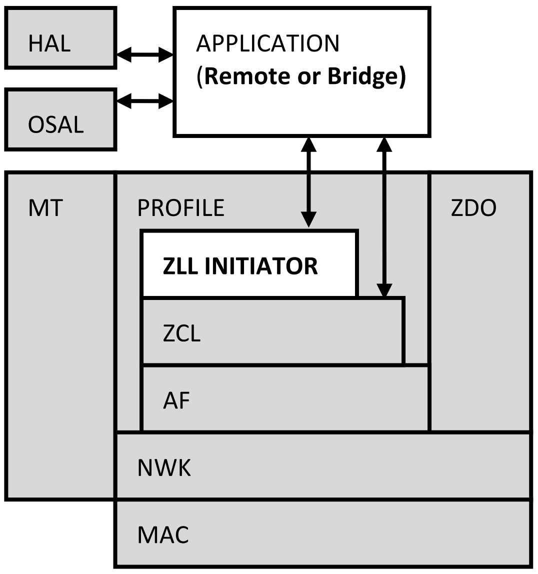

Figure 1: Z-Stack Lighting controller software modules

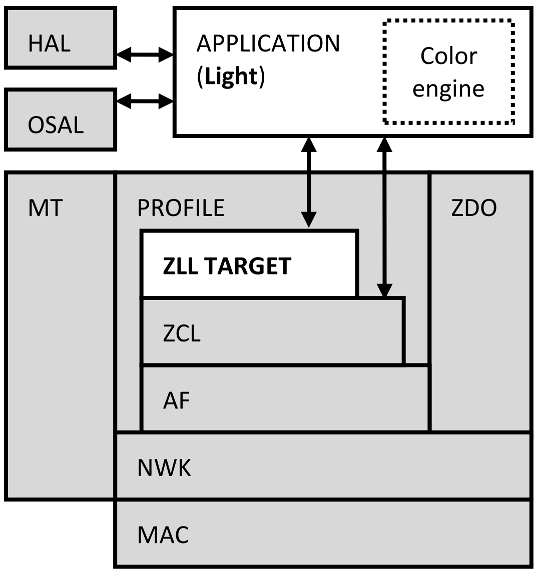

Figure 2: Z-Stack Lighting light software modules

2.3 File Structure

The following directories hold most of the ZLL-specific code:

`\Projects\zstack\ZLL\SampleApp\CC253x` – IAR workspace files for the following projects:

- *SampleLight* – Sample project for a lighting device (lamp), with support for the CC2531 *Zlight2* reference platform.
- *SampleRemote* – Sample project for a remote controller device, with support for the CC2530 *Advanced Remote Control (ARC)* platform and the CC2530 *ZLL Remote Control (ZLLRC)* platform.
- *SampleBridge* – Sample project for a control-bridge device, with support for the CC2531 USB Dongle.

All projects also support the *SmartRF05EB+CC2530EM* development kit platform.

`\Projects\zstack\ZLL\SampleApp\Source\Light` – Lighting device sample application, including color engine source code.

`\Projects\zstack\ZLL\SampleApp\Source\Remote` – Remote Controller sample application source code.

`\Projects\zstack\ZLL\SampleApp\Source\Bridge` – Control-Bridge sample application source code.

`\Projects\zstack\ZLL\Source` – ZLL Profile and MT extensions source code.

`\Components\stack\zcl` – ZLL additions to the ZCL functional domains.

3. Essential configurations

The following configurations must be modified by the user in order to create a valid ZLL device. They are all found in the file *zll.h*:

3.1 Master Keys Installation

All commercial ZLL products use the "ZLL master key" and the "ZLL pre-installed link key" set. This set of keys could be available to manufacturers which have:

1. Successfully certified a ZLL product, using the certification keys set provided by default.
2. Signed a safekeeping contract, described in ZigBee Light Link Security Agreement [3].

Note that any ZLL implementation will not be able to interoperate with commercial ZLL devices without the ZLL master keys. Once the ZLL master keys have been achieved, they should be installed in the code with the following modifications:

1. Uncomment the definitions of `ZLL_MASTER_KEY` and `ZLL_MASTER_LINK_KEY`.
2. Overwrite the "`0x??`" placeholders with the actual secret values.
3. Comment the "for certification only:" section and uncomment the "for production:" section, to enable the following definitions:

```
#define ZLL_ENC_KEY      ZLL_MASTER_KEY
#define ZLL_LINK_KEY     ZLL_MASTER_LINK_KEY
#define ZLL_KEY_INDEX    ZLL_KEY_INDEX_MASTER
```

3.2 RSSI calibration

According to the ZLL specification, a device should respond to scan requests only if their RSSI is above a certain threshold. In addition, the scan request of a device will carry an RSSI correction field, to compensate for RF signal loss due to form-factor limitations, see section 8.4.1.2 in the Light Link Profile Specification [1].

According to the ZLL test specification, a ZLL device should respond to scan request and response sent from a certified device in a distance of 10cm, and discard such messages when sent from a certified device in a distance of 2-10 meters. See initial test steps of the preamble tests, in section 2 of the Light Link Profile Test Specification [2].

As a result, the user should calibrate the following definitions for each hardware device:

`ZLL_TL_WORST_RSSI` – threshold value in dBm above which incoming scan messages (combined with their correction offset value included) will not be discarded. This value can also be set in the project's options as a compilation flag.

`ZLL_RSSI_CORRECTION` – RSSI correction offset value to be sent with scan response, to compensate, if required, for RF signal loss specific to the device form factor. This value can also be set in the project's options as a compilation flag.

4. Compilation Flags

4.1 Mandatory Compilation Flags

The following flags must be defined in any Z-Stack Lighting project:

- `NV_RESTORE`
- `INTER_PAN`
- `SECURE=1`
- `ZCL_IDENTIFY`
- `ZCL_ON_OFF`
- `ZCL_GROUPS`
- `ZCL_SCENES –` This flag enables the capability to send or receive ZCL Scenes cluster commands, and it is optional for controller devices.
- `ZCL_LIGHT_LINK_ENHANCE`
- `NUM_DISC_ATTEMPTS=0 –` Required for controller devices only.
- `ZIGBEE_FREQ_AGILITY`
- `TC_LINKKEY_JOIN`
- `ZDSECMGR_TC_DEVICE_MAX=2`

The following flag must be defined in any Z-Stack Lighting project using pre-compiled TIMAC libraries:

- `FEATURE_GREEN_POWER`

Note that in IAR Embedded Workbench for 8051, the definitions in the project's preprocessor defined symbols list will override definitions set in the files *f8wConfig.cfg* and *f8wZCL.cfg*.

4.2 Optional Compilation Flags

The following flags could be included in a Z-Stack Lighting project's options preprocessor defined symbols list, each one adding code to enable different feature.

The following flags are set by default on all the sample projects:

- `ZCL_LEVEL_CTRL –` enables the capability to send or receive ZCL level cluster commands. It is required for dimmable lights and controllers.
- `ZCL_COLOR_CTRL –` enables the capability to send or receive ZCL color cluster commands. It is required for color lights and controllers, and requires `ZCL_LEVEL_CTRL` to be defined as well.
- `HOLD_AUTO_START –` prevents the device from automatically trying to join a network after power-up when Factory New, and causes it to wait for an external event instead.

The following flags are set by default on all the SmartRF05EB projects:

- `MT_TASK –` enables any MT functionality, via serial link.
- `MT_APP_FUNC –` enables MT_APP functionality for communication with a host processor, allowing instigation of ZLL remote procedure calls.
- `MT_SYS_FUNC –` enables interaction at system level, required for communicating with Z-Tool.
- `ZTOOL_P1 –` required for communicating with Z-Tool.

Additional MT flags:

- `MT_UTIL_FUNC –` enables MT_UTIL functionality, e.g. key press events emulation.

- `MT_ZDO_FUNC –` enables MT_ZDO functionality, e.g. sending *Simple_Desc_req* command.
- `MT_ZDO_CB_FUNC –` enables received ZDO commands to be forwarded to MT, for bridge use case.
- `MT_ZDO_MGMT –` enables MT_ZDO management functionality, e.g. sending *Mgmt_Permit_Joining_req* command.

The following flags are set by default on the ARC and ZLLRC projects:

- `POWER_SAVING –` enables an end-device to sleep and conserve power, when inactive. Should be defined on any battery-powered device.
- `POLL_RATE=0` – override the default poll rate of polling end-devices, allowing them to sleep until wakened by an external event, e.g. user key press on a controller board.
- `ISR_KEYINTERRUPT –` set the device to rely on key interrupts instead of polling for key inputs, allowing to fall to deep-sleep (PM3) if defined on end-device with `POWER_SAVING` and `POLL_RATE=0`.

5. General Parameters

5.1 Specification-Defined Profile Attributes Defaults and Constants

The ZLL profile defines constants and internal attributes defaults to allow a device to manage the way the ZLL profile operates (see sections 8.1.8 and 8.1.9 in ZigBee Light Link Profile Specification [1]).

Definition	Specification's Constant / Attribute default	Value
`ZLL_APLC_INTER_PAN_TRANS_ID_LIFETIME`	*aplcInterPANTransIdLifetime*	8000
`ZLL_APLC_MAX_PERMIT_JOIN_DURATION`	*aplcMaxPermitJoinDuration*	60
`ZLL_APLC_MIN_STARTUP_DELAY_TIME`	*aplcMinStartupDelayTime*	2000
`ZLL_APLC_RX_WINDOW_DURATION`	*aplcRxWindowDuration*	5000
`ZLL_APLC_SCAN_TIME_BASE_DURATION`	*aplcScanTimeBaseDuration*	250
`ZLL_ADDR_MIN`	*aplFreeNwkAddrRangeBegin*	0x0001
`ZLL_ADDR_MAX`	*aplFreeNwkAddrRangeEnd*	0xfff7
`ZLL_GRP_ID_MIN`	*aplFreeGroupIDRangeBegin*	0x0001
`ZLL_GRP_ID_MAX`	*aplFreeGroupIDRangeEnd*	0xfeff

Table 1: Definitions Derived From the ZLL Profile Specification

5.2 Profile Endpoints Setup

Since the touch-link commissioning is managed by a dedicated task separated from the applications, its endpoint could be re-defined to any valid value, which is not in use by any application endpoint on the device, and its device ID, appearing in the simple descriptor, should not be equal to any valid value to prevent accidental match.

`ZLL_INTERNAL_ENDPOINT` (default = 13).

`ZLL_INTERNAL_DEVICE_ID` (default = 0xE15E).

5.3 Identify Sequence Time Interval

In the touch-link commissioning sequence, if an appropriate scan response command is received, the initiator will send an Identify command to the chosen target and then wait for a time interval defined by the following parameter (in milliseconds) before sending a network start or network join command.

`ZLL_INITIATOR_IDENTIFY_INTERVAL` – interval time between target selection and network start/join command (default = 500).

It is possible to gracefully abort a touch-link process (see section 8.4.1.1 in ZigBee Light Link Profile Specification [1]), until the end of this time interval. Beyond that, target state may change irreversibly. If abort is employed and controlled by a human interaction, it is recommended to increase this value (e.g. to 2000). Please note that increasing it to a higher value than the default also increases the risk of touch-link failure due to transaction lifetime expiration, especially if done on the secondary channel set.

5.4 Identify Default Time

When an Identify Request command is received with identify duration field value set to 0xffff (default time known by the receiver), the application's Identify callback function will be called with a duration value set according to the following parameter (in seconds):

`ZLL_DEFAULT_IDENTIFY_TIME` – identify duration if not specified in the received command (default = 3).

5.5 Free Ranges Split Thresholds

When initiating touch-link commissioning with devices which are capable of assigning addresses, ranges of free network addresses and group identifiers held by the initiator could be split and passed to the target.
The initiator can split its ranges as long as the remaining range and the target range are no less than the minimum size, defined by the following parameters:

`ZLL_ADDR_THRESHOLD` – the minimum size of addresses range after split (default = 10).

`ZLL_GRP_ID_THRESHOLD` – the minimum size of group identifiers range after split (default = 10).

5.6 Application Selective Target Touch-Link

This feature allows overriding the default RSSI-based target selection during touch-link, with an application-specific selection function. An application selection function could be used in scenarios where multiple target are expected to have similar RSSI (e.g. multiple lights bundled together), and allows integrating other parameters in the selection (e.g. Factory New state, previously selected device, etc.).

For more details see *Initiator Register Target Selection Callback Function*, in Z-Stack Lighting API [4].

6. Development-Only Parameters

The following parameters, if enabled, will break ZLL conformity and security rules. They may be used to assist during development, but must be disabled before release. All the parameters could be uncommented in *zll.h* file, instead of being defined globally in the project.

6.1 Channel Offset

The flags `Ch_Plus_1`, `Ch_Plus_2`, or `Ch_Plus_3`, if enabled during compilation, will shift the primary channel set, allowing testing of multiple ZLL device sets in the same space without interference.

6.2 Fixed Channel Selection

The flag `ZLL_DEV_SELECT_FIRST_CHANNEL`, if enabled during compilation, will override the random channel selection mechanism employed by the ZLL device, and will set it to always select the first primary channel.

6.3 Fixed Network Key

The flag `ZLL_DEV_FIXED_NWK_KEY`, if enabled during compilation, will override the random NWK key selection mechanism employed by the ZLL device, and will set it according to the flag's value.

7. Notable Sample Application Parameters

The sample applications provided with this release demonstrate some of the key lighting capabilities "out of the box". If those samples are used as reference to inspire your application's design, please notice the following notable parameters, affecting the provided sample application behavior.

For additional information about how to operate the provided sample applications, please refer to Z-Stack Lighting Sample Application User's Guide [5].

7.1 Sub-device's Endpoint Setup

The endpoint of the operational sub-device, which interacts with the on/off functionality, is defined by `SAMPLELIGHT_ENDPOINT`, `SAMPLEREMOTE_ENDPOINT` and `SAMPLEBRIDGE_ENDPOINT` for the lighting, controller and control-bridge sample applications, respectively.

7.2 Sub-device's Attributes

The basic attributes, simple descriptors and device information records of the sample applications are defined in the files *zll_samplelight_data.c*, *zll_sampleremote_data.c* and *zll_samplebridge_data.c* of the lighting, controller and control-bridge sample applications, respectively.

Note that the Z-Stack Lighting framework does not require correlation between the sub-device information record and its application's simple descriptor. This allows maximum flexibility when integrating with ZLL and non-ZLL networks.

7.2.1 Hub Compatibility

The ZLL Profile Specification v1.0 defined a set of device IDs for lighting devices which is incompatible with the standard set of device IDs for lighting devices, defined in the ZCL Specification and used by other profiles. The ZLL device IDs set is planned to be replaced with the standard ZCL in the following ZLL version.
In order to prevent ambiguity or identification issues by existing hub devices during device discovery, a lighting device could be compiled with `ZLL_1_0_HUB_COMPATIBILITY` flag (set by default in the Sample Light application), which exposes two endpoints for device discovery – the first is standard Home Automation, the second is using the ZLL v1.0 device ID, to improve interim profile interoperability.

7.3 Sub-device's Required Group Identifiers

According to the ZLL specification, a ZLL device may request and receive a set of unique group identifiers during commissioning, for the use of its sub-devices.
In the provided sample applications, the number of unique group identifiers to be requested upon commissioning is set by `SAMPLELIGHT_NUM_GRPS` and `SAMPLEREMOTE_NUM_GRPS`.

Note that in the provided sample applications' implementation those identifiers are not used, and therefore the parameters mentioned are set by default to 0.

7.4 Application's Stored Data

According to the ZLL test specification, the controller device should hold its linked targets information after a power cycle, or when its batteries are replaced (see section 5.3, step 25 in [2]).

The Sample Remote uses non-volatile memory (NV), by defining NV items in the range reserved for user applications (0x0401 and up, according to *ZComDef.h*): `ZCD_NV_ZLL_REMOTE_LINK_TARGETS` for linked targets' unique addresses and `ZCD_NV_ZLL_REMOTE_CTRL_GROUPS` for successfully added groups.

The number of stored addresses and groups is set by `MAX_LINKED_TARGETS` (default = 10) and `MAX_LINKED_GROUPS` (default = 3), respectively, and may be modified according to memory availability.

7.5 ZLL Commissioning Utility Commands Enablement

The ZLL specification defines that all controller devices shall respond to ZLL commissioning utility client commands. The utility commands are exchanged between controllers which are most likely sleeping end-devices, so if sent, it is recommended to instigate them automatically immediately after touch-link is done, while the end-devices are still polling. However, it is not mandatory to generate such commands, especially if controller replication is not implemented. A user should consider the appropriate behavior of his specific implementation.

In the Sample Remote application, Endpoint Information command is sent after successful touch-link to a target controller if `ZLL_UTILITY_SEND_EPINFO_ENABLED` is set. Upon receiving Endpoint Information command Get Endpoint List Request and Get Group Identifiers commands are sent if `ZLL_UTILITY_SEND_GETEPLIST_ENABLED` and `ZLL_UTILITY_SEND_GETGRPIDS_ENABLED`, respectively, are set.

7.6 Sample Light's Color Engine

Any color LED-based lighting device would require an algorithmic calculation engine to convert the received discrete color control ZCL commands into smooth color light transitions by the LEDs. The conversion should deal with color space transformations, gamut adjustments, gamma corrections, temperature and specific LED compensations over time and frequencies etc., all in an efficient manner.
The Sample Light application uses a color engine with API defined in *hw_light_ctrl.h*. Unlike the rest of the Sample Light code, this module was not written as reference for a complete general case accurate embedded color engine, although it could be used as a source of inspiration for designing such a module. Its purpose is to demonstrate color commands execution on the *Zlight2* HW reference platform.

7.7 Sample Light's Thermal Shutdown

Any electronic lighting device should have a mechanism to monitor its temperature and prevent overheating by shutting down the light, when required.
The Sample Light application implements such monitoring, using the ADC component in the CC253x SoC to provide temperature indication, with API defined in *hw_thermal_ctrl.h*. The threshold temperature over which the light will shut down is defined by `SAMPLELIGHT_THERMAL_THRESHOLD` in Celsius degrees (default = 90). The temperature sampling rate is derived from the time interval between samples, defined by `SAMPLELIGHT_THERMAL_SAMPLE_INTERVAL` in milliseconds (default = 1000).

7.8 Sample Bridge's Device Discovery

The Sample Bridge application demonstrates a ZLL processor for a stationary host, controlled by MT messages over serial interface. The application monitors ZDO messages in the network, in order to identify any newly joined device by its Device_announcement message. The application then queries each of its active endpoints for its simple descriptor, in order to provide the host with the necessary information to determine whether to add the new endpoint to its targets bank.
The following diagram describes this process:

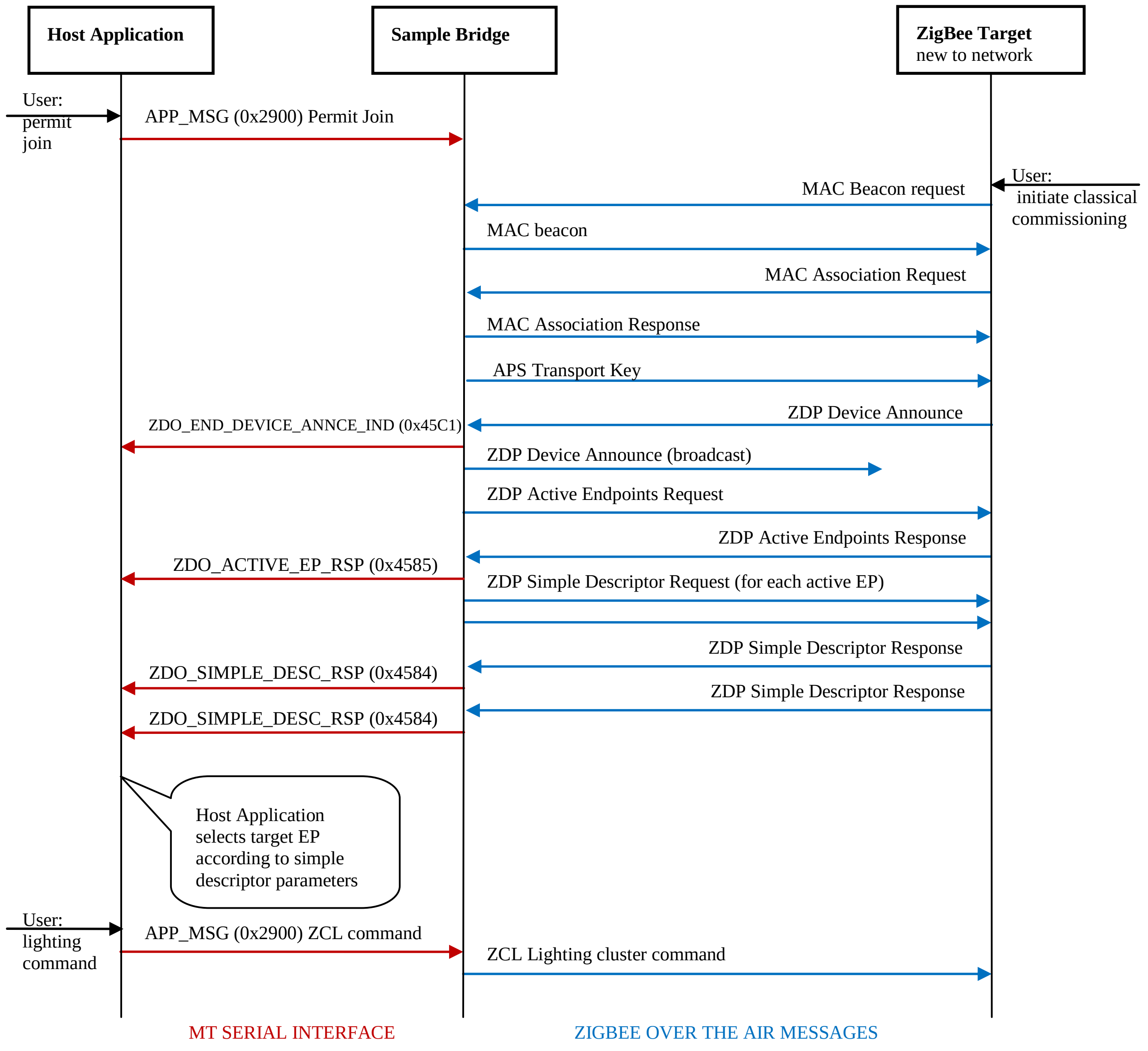
Host Application
Sample Bridge
ZigBee Target new to network
User: permit join
APP_MSG (0x2900) Permit Join
User: initiate classical commissioning
MAC Beacon request
MAC beacon
MAC Association Request
MAC Association Response
APS Transport Key
ZDP Device Announce
ZDO_END_DEVICE_ANNCE_IND (0x45C1)
ZDP Device Announce (broadcast)
ZDP Active Endpoints Request
ZDP Active Endpoints Response
ZDO_ACTIVE_EP_RSP (0x4585)
ZDP Simple Descriptor Request (for each active EP)
ZDP Simple Descriptor Response
ZDO_SIMPLE_DESC_RSP (0x4584)
ZDP Simple Descriptor Response
ZDO_SIMPLE_DESC_RSP (0x4584)
Host Application selects target EP according to simple descriptor parameters
User: lighting command
APP_MSG (0x2900) ZCL command
ZCL Lighting cluster command
MT SERIAL INTERFACE
ZIGBEE OVER THE AIR MESSAGES

Figure 3: Example of interaction with Sample Bridge for new target scenario