NXP Semiconductors
Reference Manual

Document Number: KW01RCDRDSWRM
Rev. 0, 10/2016

KW01 Remote Control Dimmer Reference Design Software

1. Introduction

This document provides a detailed firmware description for the Sub-GHz RC Dimmer reference design. It includes firmware explanation for end device (KW01-RCD-RD), routers and coordinator.

The Sub-GHz RC Dimmer is a reference design which demonstrates the functionality of the MKW01Z128 highly integrated, cost-effective, system-in-package (SIP), sub-1 GHz wireless node solution transceiver and low-power ARM® Cortex® M0+ CPU microcontroller operating in a custom IEEE 802.15.4 star network.

Contents

2. Application Command Frames

Application frames are sent between IEEE 802.15.4 devices to perform specific tasks in Sub-GHz RC Dimmer reference design. Next sub chapters describe application command frames and behaviors.

All payload sections in command frames are 1-byte size.

2.1. Toggle bulb command frame

Toggle bulb command frame is sent from end device to coordinator or router.

This command frame toggles the RGB LED bulb in GUI when received. It is a single byte command frame.

Command code = 0x01.

Figure 1. Toggle bulb command frame

2.2. Change HSL command frame

Change HSL command frame is sent from end device to coordinator or router.

This command frame configures the received HSL values for operational mode 1 in the RGB LED bulb GUI when received.

Command code = 0x02.

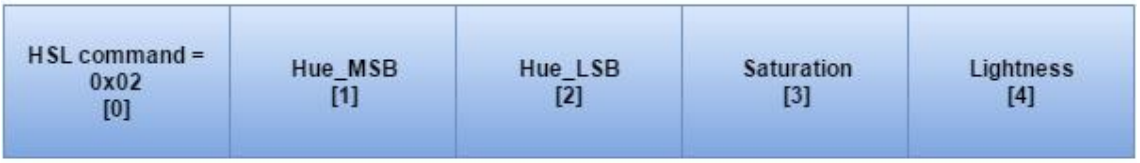

Figure 2. Change HSL command frame

2.3. Change time interval command frame

Change time interval command frame is sent from end device to coordinator or router.

This command frame configures the received time interval in milliseconds of the color sequence in the RGB LED bulb GUI for operational mode 2.

Command code = 0x03.

Figure 3. Change time interval command frame

2.4. Toggle mode command frame

Toggle mode command frame is sent from end device to coordinator or router.

This command frame toggles between mode 1 (solid color) or mode 2 (color sequence).

Command code = 0x04.

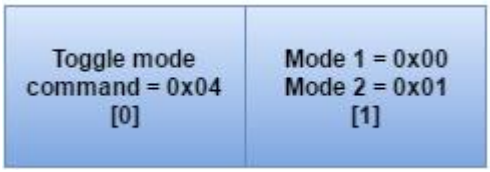

Figure 4. Toggle mode command frame

2.5. Ask for connected bulbs command frame

Ask for connected bulbs command frame is a single byte frame sent from end device to coordinator in order to receive a response from coordinator with the connected routers to it.

This command is not handled by GUI.

Command code = 0x05.

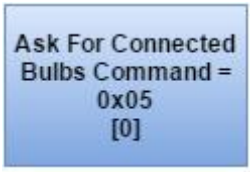

Figure 5. Ask for connected bulbs command frame

2.6. Keep alive command frame

Keep alive command frame is a single byte frame that is sent from routers to coordinator in order to inform they continue alive in the network.

This command is not handled by GUI.

Command code = 0x06.

Figure 6. Keep alive command frame

3. Software Architecture

The software architecture contemplates the main layers described in Figure 7.

The same software architecture applies in all IEEE 802.15.4 devices for this reference. Firmware's differences for each device (end device, routers and coordinator) are found in its respective application layer.

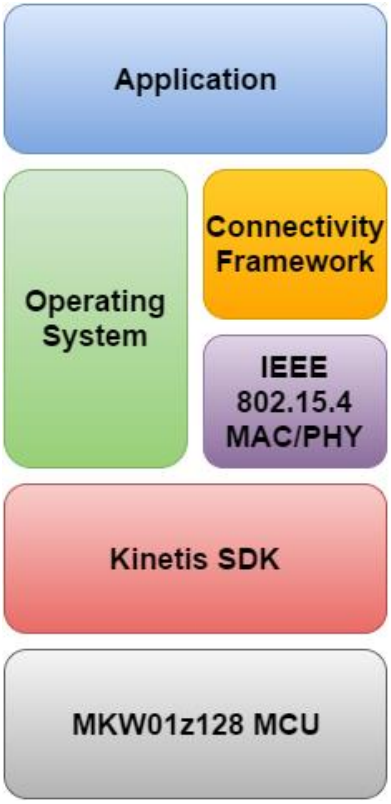

Figure 7. Software architecture diagram

3.1. Kinetis SDK layer

The Kinetis SDK layer includes all the necessary functions to interact with the microcontroller's modules. Access to modules is performed through the NXP Kinetis SDK APIs. All the documentation describing this layer functionality can be found in the Kinetis SDK documentation.

3.2. IEEE 802.15.4 MAC/PHY layer

This layer uses the IEEE 802.15.4 MAC/PHY library from KW01 connectivity software.

All the information related with the use of this software library can be obtained in the following documents from KW01 connectivity software documentation:

- **Application Developer's Guide**: Freescale 802.15.4 MAC/PHY
- **Reference Manual**: Freescale IEEE 802.15.4 MACPHY Software

3.3. OS layer

Kinetis KSD FreeRTOS library is used for coordinator, and Kinetis KSD Bare Metal library is used for routers and end device. All the documentation related to the use of the KSDK API is described in the KSDK documentation.

3.4. Connectivity Framework layer

This layer makes use of the Connectivity Framework library. It includes APIs to interface with a GUI using FSCI, APIs for Low Power, LED, Keyboard, Timers, among others.

All the information related with the use of the Connectivity Framework APIs must be obtained from the Connectivity Framework Reference Manual.

4. End Device Application Layer

This chapter describes functions, defines and variables in end device application layer for Sub-GHz RC Dimmer reference design.

This application is based in IEEE 802.15.4 MyWirelessApp demo (end device) from KW01 connectivity software. Demo was cloned using the project cloner tool which can be found in tools folder from KW01 connectivity software, and later modified for application devices in this reference design. Additional documentation can be found in doc folder from KW01 connectivity software.

4.1. Functions documentation for end device

This section describes the application layer's functions for end device. Some of the functions described in this chapter are also used by routers and coordinator in its respective application layer.

4.1.1. main_task

Prototype

extern void main_task (void const *argument);

Parameter

const *argument [in] – optional input argument

Return

None

Description

This is the first task created by the OS. This task will initialize the system. It also initialized the necessary modules for the application (timers, LEDs, etc).

4.1.2. App_Idle_Task

Prototype

void App_Idle_Task (uint32_t argument);

Parameters

uint32_t argument [in] – optional input argument

Return

None

Description

This is the application's Idle task which is in charge of checking if board can go into low power mode with PWR_CheckIfDeviceCanGoToSleep function and configuring it with Enter_LowPower function.

4.1.3. App_init

Prototype

void App_init (void);

Parameters

None

Return

None

Description

Initialization function for the application. It is called during initialization in main_task function and it contains application specific initialization. MAC, timers, keyboard, and application initializations are found in this function.

4.1.4. AppThread

Prototype

void AppThread (uint32_t argument);

Parameters

uint32_t argument [in] – optional argument

Return

None

Description

Mac Application Task event processor. This function is called to process all events for the task.

Events include timers, messages and application events. This function includes the application state machine.

For end device, the app state machine is described in the following chapter.

4.1.4.1. End device application state machine

The application state machine for end device begins scanning and associating with coordinator (states 1 to 4). After end device is succesfully associated with coordinator then the state machine goes to a listen state (state 5) where the application handles the MLME messages and it waits until an application button is pressed. Once the App_HandleKeys or App_TsiCallback function are called by pressing one of the application buttons, then the application state machine goes to stateTxPacket state where the packet is filled depending on the command that is going to be send using FillAppPkt, and App_TransmitData function is called to transmit data over the air. The following figure shows flow diagram of application state machine for end device.

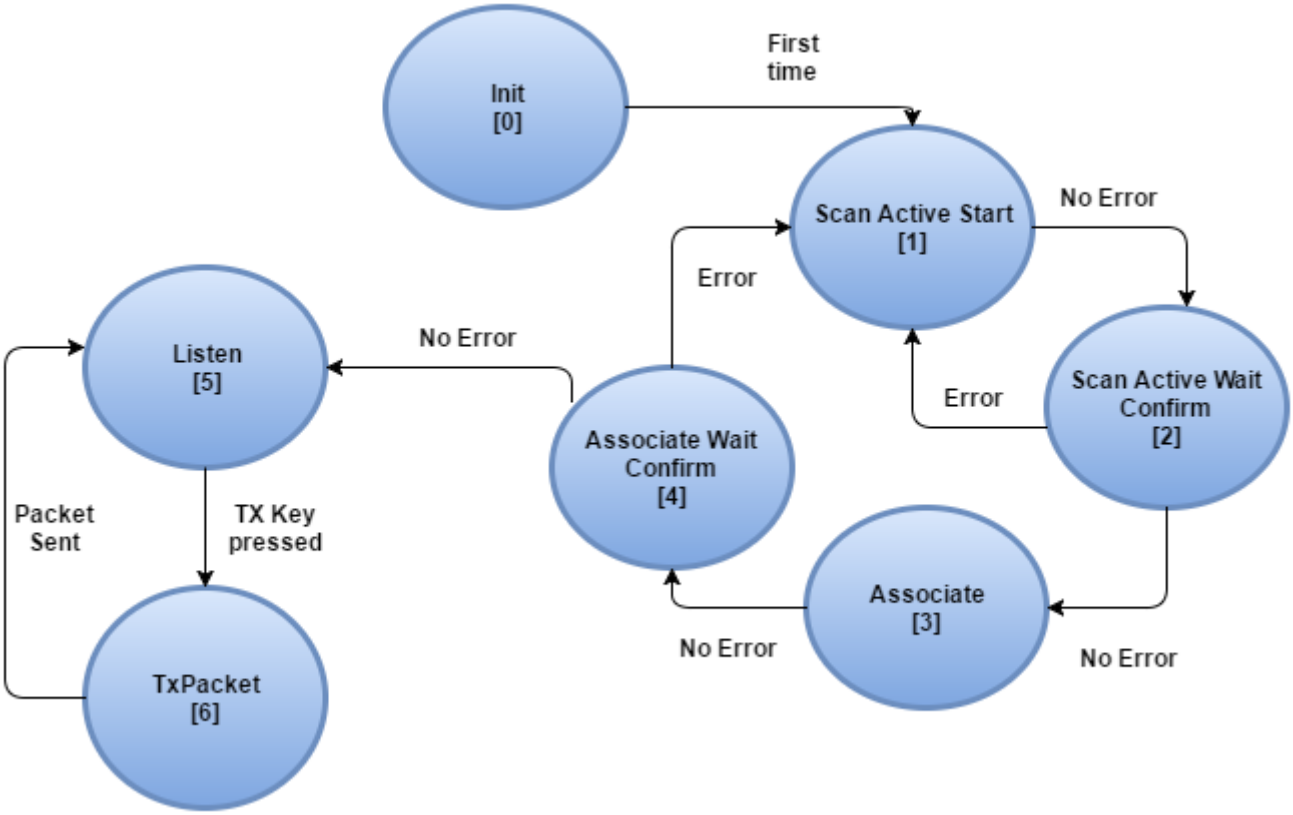

Figure 8. End device application state machine

4.1.5. App_StartScan

Prototype

static uint8_t App_StartScan (macScanType_t scanType);

Parameters

- macScanType_t scanType [IN] – Options: gScanModeED_c, gScanModeActive_c, gScanModePassive_c, gScanModeOrphan_c or gScanModeFastED_c

Return

The function may return either of the following values:

* errorNoError: The Scan message was sent successfully.
* errorInvalidParameter: The MLME service access point rejected the message due to an invalid parameter.
* errorAllocFailed: A message buffer could not be allocated.

Description

The App_StartScan(scanType) function starts the scan process of the specified type in the MAC. This is accomplished by allocating a MAC message, which is then assigned the desired scan parameters and sent to the MLME service access point.

4.1.6. App_HandleScanActiveConfirm

Prototype

static uint8_t App_HandleScanActiveConfirm (nwkMessage_t *pMsg);

Parameters

- nwkMessage_t *pMsg [IN] – active scan confirm message received from the MLME

Return

The function may return either of the following values:

* errorNoError: A suitable pan descriptor was found.
* errorNoScanResults: No scan results were present in the confirm message.

Description

The App_HandleScanActiveConfirm(nwkMessage_t *pMsg) function will handle the Active Scan confirm message received from the MLME when the Active scan has completed. The message contains a list of PAN descriptors. Based on link quality information in the pan descriptors the nearest coordinator is chosen. The corresponding pan descriptor is stored in the global variable mCoordInfo.

This function is called in ScanActiveWaitConfirm state from application state machine.

4.1.7. App_SendAssociateRequest

Prototype

static uint8_t App_SendAssociateRequest (void);

Parameters

None

Return

The function may return either of the following values:
* errorNoError: The Associate Request message was sent successfully.
* errorInvalidParameter: The MLME service access point rejected the message due to an invalid parameter.
* errorAllocFailed: A message buffer could not be allocated.

Description

The App_SendAssociateRequest(void) will create an Associate Request message and send it to the coordinator it wishes to associate to. The function uses information gained about the coordinator during the scan procedure. This function is called in Associate state from application state machine.

4.1.8. App_HandleAssociateConfirm

Prototype

static uint8_t App_HandleAssociateConfirm(nwkMessage_t *pMsg);

Parameters

- nwkMessage_t *pMsg [IN] – associate confirm message received from the MLME

Return

The function will return the gSuccess_c in case of a success reception of the associate confirm message.

Description

The App_HandleAssociateConfirm(nwkMessage_t *pMsg) function will handle the Associate confirm message received from the MLME when the Association procedure has completed. The message contains the short address that the coordinator has assigned to us. This address is 0xfffe if we did not specify the gCapInfoAllocAddr_c flag in the capability info field of the Associate request. The address and address mode are saved in global variables. They will be used in the next demo application when sending data.

4.1.9. App_HandleMlmeInput

Prototype

static uint8_t App_HandleAssociateConfirm(nwkMessage_t *pMsg);

Parameters

- nwkMessage_t *pMsg [IN] – Received MLME message

Return

The function may return either of the following values:

* errorNoError: The message was processed.
* errorNoMessage: The message pointer is NULL.

Description

The App_HandleMlmeInput(nwkMessage_t *pMsg) function will handle various messages from the MLME, for instance poll confirm. This function is called in Listen state from application state machine.

4.1.10. App_HandleMcpsInput

Prototype

static void App_HandleMcpsInput(mcpsToNwkMessage_t *pMsgIn);

Parameters

- nwkMessage_t *pMsg [IN] – Received MCPS message

Return

None

Description

The App_HandleMcpsInput(mcpsToNwkMessage_t *pMsgIn) function will handle messages from the MCPS, for instance Data Confirm, and Data Indication.

4.1.11. App_WaitMsg

Prototype

static uint8_t App_WaitMsg(nwkMessage_t *pMsg, uint8_t msgType);

Parameters

- nwkMessage_t *pMsg [IN] – Received message
- uint8_t msgType [IN] – The expected type message

Return

The function may return either of the following values:

* errorNoError: The message was of the expected type.
* errorNoMessage: The message pointer is NULL.
* errorWrongConfirm: The message is not of the expected type.

Description

The App_WaitMsg(nwkMessage_t *pMsg, uint8_t msgType) function does not, as the name implies, wait for a message, therefore blocking the execution of the state machine. Instead the function analyzes the supplied message to determine whether or not the message is of the expected type.

4.1.12. Mac_SetExtendedAddress

Prototype

extern void Mac_SetExtendedAddress(uint8_t *pAddr, instanceId_t instanceId);

Parameters

- uint8_t *pAddr [IN]
- instanceId_t instanceId [IN]

Return

None

Description

It sets the MAC extended address.

4.1.13. App_TransmitData

Prototype

static void App_TransmitUartData(void);

Parameters

None

Return

None

Description

The App_TransmitData() function will perform (single/multi buffered) data transmissions.

The constant mDefaultValueOfMaxPendingDataPackets_c determines the maximum number of packets pending for transmission in the MAC. A global variable is incremented each time a data packet is sent to the MCPS, and decremented when the corresponding MCPS-Data Confirm message is received. If the counter reaches the defined maximum no more data buffers are allocated until the counter is decreased below the maximum number of pending packets.

The function uses the short address assigned to us by coordinator and the information of destination bulb (coordinator or router), for building an MCPS-Data Request message. The message is sent to the MCPS service access point in the MAC.

4.1.14. AppPollWaitTimeout

Prototype

static void AppPollWaitTimeout(void *);

Parameters

None

Return

None

Description

The AppPollWaitTimeout () function will check if it is time to send out an MLME-Poll request in order to receive data from the coordinator. If its time, and we are permitted then a poll request is created and sent.

The function uses the coordinator information gained during the Active Scan for building the MLME-Poll Request message. The message is sent to the MLME service access point in the MAC.

In the application this function is a callback of a timer that runs after App_TransmitUartData is called in Ask4ConnectedBulbs function to obtain the coordinator response.

4.1.15. App_HandleKeys

Prototype

static void App_HandleKeys (key_event_t events);

Parameters

key_event_t events [IN] – Event from keyboard module

Return

None

Description

Handles all push button events for end device and calls TxPacket state from end device application state machine.

4.1.16. MLME_NWK_SapHandler

Prototype

resultType_t MLME_NWK_SapHandler (nwkMessage_t* pMsg, instanceId_t instanceId);

Parameters

- nwkMessage_t* pMsg [IN] – MLME message
- instanceId_t instanceId [IN] - Instance

Return

The function may return a gSuccess_c status.

Description

This function is called by the MAC to put MLME messages into the application's queue.

4.1.17. MCPS_NWK_SapHandler

Prototype

resultType_t MCPS_NWK_SapHandler (mcpsToNwkMessage_t* pMsg, instanceId_t instanceId);

Parameters

- nwkMessage_t* pMsg [IN] – MCPS message
- instanceId_t instanceId [IN] - Instance

Return

The function may return a gSuccess_c status.

Description

This function is called by the MAC to put MCPS messages into the application's queue.

4.1.18. App_TsiCallback

Prototype

static void App_TsiCallback (tsi_sensor_electrode_flags_t* pElectrodeFlags);

Parameters

- tsi_sensor_electrode_flags_t* pElectrodeFlags [IN] – Flag to identify the pressed electrode.

Return

None

Description

Handles all touch buttons (electrodes) events for end device and call TxPacket state from end device application state machine.

4.1.19. StartLedBlinking

Prototype

static void StartLedBlinking (uint8_t times);

Parameters

- uint8_t times [IN] – Number of times the LED will blink.

Return

None

Description

It starts the mBlinkLedTimerID timer to call the StartLedBlinkingCallBack function where the RGB LED in KW01-RCD-RD board blinks every gBlinkLedInterval_c milliseconds.

4.1.20. StartLedBlinkingCallBack

Prototype

static void StartLedBlinking (uint8_t times);

Parameters

- uint8_t times [IN] – Number of times the LED will blink.

Return

None

Description

Callback function of MBlinkLedTimerID timer that runs when StartLedBlinking function is called. It blinks the RGB LED in KW01-RCD-RD board the desired number of times stored in times variable. The color of blinking it's defined in led_color variable. Timer MBlinkLedTimerID will stops when count_blinking is equal to times variable.

4.1.21. BlinkLedOnce

Prototype

static void BlinkLedOnce (void);

Parameters

None

Return

None

Description

This function blinks only once the RGB LED in KW01-RCD-RD board. The color of blinking it's defined in led_color variable.

4.1.22. LongSingleBlinkLed

Prototype

static void LongSingleBlinkLed (uint8_t color);

Parameters

- uint8_t color [IN] – The blinking color.

Return

None

Description

This function turns on the RGB LED in KW01-RCD-RD board with the color defined in the input parameter. It starts the mTurnOffLedTimerID timer which callback LongSingleBlinkLedCallback turns off the RGB LED. The time that the RGB LED is on is defined in gLedOnTime_c.

4.1.23. LongSingleBlinkLedCallBack

Prototype

static void LongSingleBlinkLedCallBack (uint8_t color);

Parameters

- uint8_t color [IN] – The blinking color.

Return

None

Description

This function is the callback of mTurnOffLedTimerID timer that runs when LongSingleBlinkLed is called. It turns off the RGB LED in KW01-RCD-RD board that was turned on in LongSingleBlinkLed function.

4.1.24. TimerEnterLP_CallBack

Prototype

static void TimerEnterLP_CallBack (void);

Parameters

None

Return

None

Description

This is the callback function of mEnterLPTimerID timer and it is called when there is no user activity in KW01-RCD-RD to configure the board into low power mode.

The function calls the PWR_AllowDeviceToSleep function and sets the LP_Indicator_flag to LP_TMR to identify that the entering to low power is by timer.

4.1.25. AllowTsiWrkCallBack

Prototype

static void AllowTsiWrkCallBack (void);

Parameters

None

Return

None

Description

This is the callback function of mWaitTSITimerID timer and it is called to set the ready2enterTSI_flag to TRUE in order to allow TSI working again.

4.1.26. AllowChangeAddrCallBack

Prototype

static void AllowChangeAddrCallBack (void);

Parameters

None

Return

None

Description

This is the callback function of mWaitChangeAddrTimerID timer and it is called to set the ready2ChangeAddr_flag to TRUE in order to allow the change address option working again.

4.1.27. FillAppPkt

Prototype

static void FillAppPkt (uint8_t command);

Parameters

- uint8_t command [IN] – The Sub-GHz RC Dimmer command code.

Return

None

Description

This function fills the AppPayloadFrame buffer with the data for the command frame specified in the input parameter.

4.1.28. Ask4ConnectedBulbs

Prototype

static void Ask4ConnectedBulbs (void);

Parameters

None

Return

None

Description

This function fills and sends the AppPayloadFrame buffer with the data for the command frame which asks the coordinator for the connected routers to it (0x05).

4.1.29. ChangeBulbAddress

Prototype

static bool_t ChangeBulbAddress (void);

Parameters

None

Return

The output parameters are Success that indicates the address was changed successfully or ChangeAddrAgain that indicates ChangeBulbAddress function needs to be called again to increment address.

Description

This function increments the current bulb address stored in Current_DestAddr variable. It’s called in App_HandleMcpsInput function when a data indication from the coordinator is received with the connected bulbs in payload.

4.1.30. GetLastConnectedBulb

Prototype

static uint8_t GetLastConnectedBulb (void);

Parameters

None

Return

This function returns the number of the last connected bulb from the CurrentConnectedBulbs array.

Description

This function returns the last connected bulb in network.

4.1.31. Enter_LowPower

Prototype

static void Enter_LowPower(void);

Parameters

None

Return

None

Description

It configures the MCU into LLS low power mode.

4.1.32. LP_LED_Indicator

Prototype

static void LP_LED_Indicator(bool_t LP_Indicator);

Parameters

- bool_t LP_Indicator [IN] – LP_Indicator_flag variable. It can contain LP_GPIO or LP_TMR options.

Return

None

Description

Behavior of LED indicator when MCU is entering into low power mode,

4.2. Variables documentation for end device

This section describes the application layer's variables for end device.

4.2.1. static panDescriptor_t mCoordInfo

Information about the PAN we are part of.

4.2.2. static uint8_t maMyAddress[8]

This is either the short address assigned by the PAN coordinator during association, or our own extended MAC address.

4.2.3. static uint8_t mMsduHandle

The MSDU handle is a unique data packet identifier.

4.2.4. static uint8_t mcPendingPackets

Number of pending data packets.

4.2.5. static bool_t mWaitPollConfirm

Signals that an MLME-Poll request is pending, and that we must wait for the MLME-Poll confirm message before sending the next poll request.

4.2.6. static uint16_t mPollInterval

Time between MLME-Poll requests.

4.2.7. static uint64_t mExtendedAddress

End device's extended address.

4.2.8. static uint8_t interfaceId;

Variable to store interface ID.

4.2.9. uint8_t CurrentConnectedBulbs [5]

Array for connected bulbs to network.

4.2.10. uint64_t Current_DestAddr

The current destination bulb address.

4.2.11. uint8_t key128 [128/8]

The AES 128 key.

4.2.12. uint8_t AppPayloadFrame [16]

The app payload buffer.

4.2.13. uint8_t AppPayload_Encrypted [16]

The encrypted app payload buffer.

4.2.14. uint32_t KeyBoard_Status

Variable to verify if GPIO is still pressed.

4.2.15. bool_t LP_Indicator_flag

Indicator flag to know if device is going to go to LP from timer or GPIO.

4.2.16. uint8_t led_color;

The RGB LED color variable.

4.2.17. bool_t ready2enterTSI_flag;

This flag helps to control the pressing of TSIs and avoid issues.

4.2.18. bool_t ready2ChangeAddr_flag

This flag helps to control the pressing of change address button and avoid issues.

4.2.19. uint8_t gState

The current state of the applications state machine.

4.2.20. uint8_t count_blinking

Variable used to count the number of LED blinking.

4.2.21. uint8_t RCD_Command

Variable to store the application command.

4.2.22. uint16_t ColorTable [6]

Array that contains the pre-defined Hue values {Red, Green, Yellow, Blue, Magenta, Cyan}.

4.2.23. uint16_t CurrentHSL [3]

Array that contains the current application HSL values.

4.2.24. uint8_t Current_color

The current application color for RGB LED.

4.2.25. bool_t app_mode

The current operational mode of the application.

4.2.26. uint16_t Sequence_Interval_Time

The current time interval for operational mode 2 of the application.

4.2.27. uint8_t LastConnectedBulb

Variable used to store the result of GetLastConnectedBulb function.

4.2.28. uint8_t ReceivedPayload

Buffer to store the received application payload.

4.2.29. uint8_t DecryptedPayload

Buffer to store the received application payload once decrypted in case of AES128 enabled.

4.2.30. static addrModeType_t mAddrMode

The devices address mode. If 2, then maMyAddress contains the short address assigned by the PAN coordinator. If 3, then maMyAddress is equal to the extended address.

4.2.31. static nwkToMcpsMessage_t *mpPacket

Data request packet for sending UART input to the coordinator.

4.2.32. static anchor_t mMlmeNwkInputQueue

Application input queues.

4.2.33. static anchor_t mMcpsNwkInputQueue

Application input queues.

4.2.34. static instanceId_t macInstance

Variable to store the mac instance.

4.2.35. event_t mAppEvent

Variable to store the application event.

4.2.36. task_handler_t mAppTaskHandler

The app task handler.

4.3. Application defines for end device

This section describes the application layer defines for end device.

4.3.1. #define mMacExtendedAddress_c

The extended address of the end device. It should not be modified for the correct behavior of the application.

Default value: 0xFFFFFFFFFFFFFFED.

4.3.2. #define gEnableAES128

Set to 1 to enable AES 128 encryption. Sub-GHz RC Dimmer GUI does not decrypt payload messages received.

Default value: 0.

4.3.3. #define gBlinkLedInterval_c

The time interval between LED blinks in StartLedBlinking function.

Default value: 300.

4.3.4. #define gLongLedOnTime_c

The time that the LED will turn on in LongSingleBlinkLed function.

Default value: 1000.

4.3.5. #define gWaitTSITime_c

The time that the application waits to call AllowTSI function.

Default value: 500.

4.3.6. #define gWaitChangeAddrTime_c

The time that the application waits to call AllowChangeAddr function.

Default value: 1000.

4.3.7. #define gEnterLPTime_c

The time that the MCU will enter to low power due to the user inactivity.

Default value: 60 seconds.

4.3.8. #define Lightness_Step

The lightness increment/decrement step.

Default value: 10.

4.3.9. #define MaxLightnessLimit_c

The maximum limit of lightness.

Default value: 100.

4.3.10. #define MinLightnessLimit_c

The minimum limit of lightness.

Default value: 0.

4.3.11. #define PredefinedLightnessValue

The pre-defined lightness value.

Default value: 50.

4.3.12. #define PredefinedSaturationValue

The pre-defined saturation value.

Default value: 100.

4.3.13. #define SequenceIntervalTimeStep

The increment/decrement time interval step for operational mode 2.

Default value: 250.

4.3.14. #define DefaultSeqTimeInMs

The default time for operational mode 2.

Default value: 1000.

4.3.15. #define Mode2Time_MaxLimit_c

The maximum limit of time in operational mode 2.

Default value: 60000.

4.3.16. #define Mode2Time_MinLimit_c

The minimum limit of time in operational mode 2.

Default value: 0.

4.4. Application enums for end device

- **typedef enum _rcd_command_t**

```
{
   kRCD_Command_Idle                  = 0U, /* Idle */
   kRCD_Command_ToggleBulb            = 1U, /* Toggle Bulb */
   kRCD_Command_ChangeHSL             = 2U, /* Change HSL */
   kRCD_Command_TimeInterval          = 3U, /* Start Toggle Sequence */
   kRCD_Command_ToggleMode            = 4U, /* Toggle between bulb mode 1 and 2 */
   kRCD_Command_CurrentConnectedBulbs = 5U, /* Ask for current connected bulbs*/
   kRCD_Command_KeepAlive             = 6U /*Keep alive command */
} rcd_command_t;                  /*The application commands*/
```

- **typedef enum _hsl_parameters_t**

```
{
   Hue        = 0U,
   Saturation = 1U,
   Lightness  = 2U
} hsl_parameters_t;          /*The frame position of HSL parameters*/
```

- **typedef enum _colors_predefined_t**

```
{
   Red       = 0U,
   Green     = 1U,
   Yellow    = 2U,
   Blue      = 3U,
   Magenta = 4U,
   Cyan      = 5U
} colors_predefined_t;        /*The application pre-defined color list*/
```

- **typedef enum _white_color_t**

```
{
   White     = 6U
} white_color_t;              /*White color (not from pre-defined color list) */
```

- **typedef enum _app_modes_t**

```
{
   AppOpMode1     = 0U,
   AppOpMode2     = 1U
} app_modes_t;                /*The application operational modes*/
```

- **typedef enum _bulbs_addr_t**

```
{
   Bulb1_Addr     = 1U,
   Bulb2_Addr     = 2U,
   Bulb3_Addr     = 3U,
   Bulb4_Addr     = 4U,
   Bulb5_Addr     = 5U,
} bulbs_addr_t;               /*The bulb short address*/
```

- **enum**
  {
  ChangeAddrAgain,
  Success
  }; /*return status for ChangeBulbAddr function*/
- **enum** {
  LastColorInTable = Cyan
  }; /*The last color in color table*/

- **enum** {
  FirstColorInTable = Red
  }; /*The first color in color table*/

- **enum** {
  BulbConnected = 1
  }; /*The bulb is connected*/

- **enum** {
  LP_GPIO = 0,
  LP_TMR = 1
  }; /*Low Power source indicators*/

- **enum** {
  Broadcast_Addr = 0xFFFF
  }; /*The broadcast address*/

5. Routers Application Layer

This chapter describe the functions, defines and variables in routers application layer for Sub-GHz RC Dimmer reference design.

This application is based in IEEE 802.15.4 MyWirelessApp demo (end device) from KW01 connectivity software. Demo was cloned using the project cloner tool which can be found in tools folder from KW01 connectivity software, and later modified for application devices in this reference design. Additional documentation can be found in the doc folder from KW01 connectivity software.

5.1. Functions documentation for routers

This section describes the application layer's functions for routers.

5.1.1. App_StartRouter

Prototype

static void App_StartRouter(void);

Parameters

None

Return

None

Description

This function initializes a router device. It sets MAC PIB attributes required for a router.

5.1.2. KeepAlive

Prototype

static void KeepAlive (uint8_t timerId);

Parameters

None

Return

None

Description

This is the callback function from mKeepAliveID timer. It is called every gKeepAliveInterval_c milliseconds and it sends a KeepAlive command frame to the coordinator.

The next list of functions are also used for end device in its respective application layer and they are described in end device documentation chapter.

Functions

- static uint8_t App_StartScan(macScanType_t scanType);
- static uint8_t App_HandleScanActiveConfirm(nwkMessage_t *pMsg);
- static uint8_t App_WaitMsg(nwkMessage_t *pMsg, uint8_t msgType);
- static uint8_t App_SendAssociateRequest(void);
- static uint8_t App_HandleAssociateConfirm(nwkMessage_t *pMsg);
- static uint8_t App_HandleMlmeInput(nwkMessage_t *pMsg);
- static void App_TransmitData(void);
- static void AppPollWaitTimeout(void *);
- void App_init (void);
- void AppThread (uint32_t argument);
- void App_Idle_Task(uint32_t argument);
- resultType_t MLME_NWK_SapHandler (nwkMessage_t* pMsg, instanceId_t instanceId);
- resultType_t MCPS_NWK_SapHandler (mcpsToNwkMessage_t* pMsg, instanceId_t instanceId);
- extern void Mac_SetExtendedAddress(uint8_t *pAddr, instanceId_t instanceId);
- static void App_HandleMcpsInput(mcpsToNwkMessage_t *pMsgIn);

5.1 Variables documentation for Routers

This section describes the application layer's variables for routers.

5.1.1 static const gFsciSerialConfig_t mFsciSerials[ ]

FSCI Interface Configuration structure.

5.1.2 static uint8_t mFsciInterface[gMacInstancesCnt_c];

This table contains indexes into the mFsciSerials[] table.

Variables

The next list of variables are also used for end device in its respective application layer and they are described in end device documentation chapter.

- static panDescriptor_t mCoordInfo;
- static uint8_t maMyAddress[8];
- static addrModeType_t mAddrMode;
- static nwkToMcpsMessage_t *mpPacket;
- static uint8_t mMsduHandle;
- static uint8_t mcPendingPackets;

- static bool_t mWaitPollConfirm;
- static uint16_t mPollInterval;
- static anchor_t mMlmeNwkInputQueue;
- static anchor_t mMcpsNwkInputQueue;
- static uint64_t mExtendedAddress;
- static instanceId_t macInstance;
- static uint8_t interfaceId;
- event_t mAppEvent;
- task_handler_t mAppTaskHandler;
- uint8_t AppPayloadFrame [16];
- uint8_t AppPayload_Encrypted [16];
- uint8_t ReceivedPayload [16];
- uint8_t DecryptedPayload [16];
- uint8_t key128[128/8]
- uint8_t gState;

5.2. Application defines for routers

This section describes the application layer's defines for routers.

5.2.1. define gKeepAliveInterval_c

The interval time of KeepAlive messages.

Default value: 8000 milliseconds.

5.2.2. #define AppDevice_ExtAddr_c

The selected application router number.

Options: Router1_ExtAddr, Router2_ExtAddr, Router3_ExtAddr, Router4_ExtAddr

5.2.3. #define Router1_ExtAddr

The Router 1 extended address.

Default value: 0xFFFFFFFFFFFFFFFA.

5.2.4. #define Router2_ExtAddr

The Router 2 extended address.

Default value: 0xFFFFFFFFFFFFFFFB.

5.2.5. #define Router3_ExtAddr

The Router 3 extended address.

Default value: 0xFFFFFFFFFFFFFFFC.

5.2.6. #define Router4_ExtAddr

The Router 4 extended address.

Default value: 0xFFFFFFFFFFFFFFFD.

5.2.7. #define gEnableAES128

Set to 1 to enable AES 128 encryption. Sub-GHz RC Dimmer GUI does not support encrypted messages.

Default value: 0.

6. Coordinator Application Layer

This chapter describe the functions, defines and variables in coordinator application layer for Sub-GHz RC Dimmer reference design.

This application is based in IEEE 802.15.4 MyWirelessApp demo (coordinator) from KW01 connectivity software. Demo was cloned using the project cloner tool stored in tools folder from KW01 connectivity software, and later modified for application devices in this reference design. Additional documentation can be found in doc folder from KW01 connectivity software.

6.1. Functions documentation for coordinator

This section describes the application layer's functions for coordinator.

6.1.1. App_StartCoordinator

Prototype

static uint8_t App_StartCoordinator(uint8_t appInstance);

Parameters

uint8_t appInstance [IN] – The app instance

Return

The function may return either of the following values:

* errorNoError: The Scan message was sent successfully.
* errorInvalidParameter: The MLME service access point rejected the
* message due to an invalid parameter.
* errorAllocFailed: A message buffer could not be allocated.

Description

The App_StartScan(scanType) function will start the scan process of the specified type in the MAC. This is accomplished by allocating a MAC message, which is then assigned the desired scan parameters and sent to the MLME service access point. The MAC PIB attributes "macShortAddress", and “macAssociatePermit" are modified.

6.1.2. CheckIfAlive_Router1

Prototype

static void CheckIfAlive_Router1(uint8_t timerId);

Parameters

uint8_t timerId [IN] – The time ID.

Return

None

Description

Callback function for timer which checks if router 1 is still alive.

6.1.3. CheckIfAlive_Router2

Prototype

static void CheckIfAlive_Router2(uint8_t timerId);

Parameters

uint8_t timerId [IN] – The time ID.

Return

None

Description

Callback function for timer which checks if router 2 is still alive.

6.1.4. CheckIfAlive_Router3

Prototype

static void CheckIfAlive_Router3(uint8_t timerId);

Parameters

uint8_t timerId [IN] – The time ID.

Return

None

Description

Callback function for timer which checks if router 3 is still alive.

6.1.5. CheckIfAlive_Router4

Prototype

static void CheckIfAlive_Router4(uint8_t timerId);

Parameters

uint8_t timerId [IN] – The time ID.

Return

None

Description

Callback function for timer which checks if router 4 is still alive.

6.1.6. TransmitConnectedBulbs

Prototype

static void TransmitConnectedBulbs (void);

Parameters

None

Return

None

Description

Function which transmits the array of routers connected to coordinator

6.1.7. App_HandleScanEdConfirm

Prototype

static void App_HandleScanEdConfirm(nwkMessage_t *pMsg);

Parameters

nwkMessage_t *pMsg [IN] – The input packet.

Return

None

Description

The App_HandleScanEdConfirm(nwkMessage_t *pMsg) function will handle the ED scan confirm message received from the MLME when the ED scan has completed. The message contains the ED scan result list. This function will search the list in order to select the logical channel with the least energy. The selected channel is stored in the global variable called 'maLogicalChannel'.

6.1.8. App_SendAssociateResponse

Prototype

static uint8_t App_SendAssociateResponse(nwkMessage_t *pMsgIn, uint8_t appInstance);

Parameters

nwkMessage_t *pMsg [IN] – The input packet.

uint8_t appInstance [IN] – The app instance.

Return

The function may return either of the following values:

* errorNoError: The Associate Response message was sent successfully.
* errorInvalidParameter: The MLME service access point rejected the message due to an invalid parameter.
* errorAllocFailed: A message buffer could not be allocated.

Description

The App_SendAssociateResponse(nwkMessage_t *pMsgIn) will create the response message to an Associate Indication (device sends an Associate Request to its MAC. The request is transmitted to the coordinator where it is converted into an Associate Indication). This function will extract the devices long address, and various other flags from the incoming indication message for building the response message.

Functions

The next list of functions are also used for end device in its respective application layer and they are described in end device documentation chapter.

- static uint8_t App_StartScan(macScanType_t scanType, uint8_t appInstance);
- static uint8_t App_HandleMlmeInput(nwkMessage_t *pMsg, uint8_t appInstance);
- static uint8_t App_WaitMsg(nwkMessage_t *pMsg, uint8_t msgType);
- void App_init(void);
- void AppThread (uint32_t argument);
- resultType_t MLME_NWK_SapHandler (nwkMessage_t* pMsg, instanceId_t instanceId);
- resultType_t MCPS_NWK_SapHandler (mcpsToNwkMessage_t* pMsg, instanceId_t instanceId);
- extern void Mac_SetExtendedAddress(uint8_t *pAddr, instanceId_t instanceId);
- static void App_HandleMcpsInput(mcpsToNwkMessage_t *pMsgIn, uint8_t appInstance);

6.2. Variables documentation for coordinator

This section describes the application layer's variables for coordinator.

6.2.1. static uint8_t mDeviceShortAddress[2]

These byte arrays store an associated short address.

6.2.2. static uint8_t mDeviceLongAddress[8]

These byte arrays store an associated long address.

6.2.3. static const uint8_t mShortAddress[2]

These coordinator short address.

6.2.4. static const uint8_t mPanId [2]

These current PAN ID.

6.2.5. static uint8_t mLogicalChannel

The current logical channel (frequency band).

6.2.6. static const uint8_t maxRoutersSupported

Size of Neighbor Table. Default value is 4.

6.2.7. struct neighborTable_struct neighborTable[5]

Application neighbor table declaration.

6.2.8. uint8_t currentNeighborTableSize

Variable to store the current neighbor table size.

6.2.9. uint8_t NoResponseRouter1_Count

Variables to store the number of times the router 1 doesn´t answer.

6.2.10. uint8_t NoResponseRouter2_Count

Variables to store the number of times the router 2 doesn´t answer.

6.2.11. uint8_t NoResponseRouter3_Count

Variables to store the number of times the router 3 doesn´t answer.

6.2.12. uint8_t NoResponseRouter4_Count

Variables to store the number of times the router 4 doesn´t answer.

6.2.13. uint8_t DeviceAddrOfPcktReceived [2]

Array where the source address of a received message is stored.

6.2.14. uint8_t ConnectedDevices [16]

Array which contains the status of the routers in network {Router1, Router2, Router3, Router4} where 0-Disconnected, 1-Connected.

6.2.15. uint8_t ConnectedDevices_Encrypted [16]

Array to store the ConnectedDevices array with encryption in case AES128 is enabled.

6.2.16. uint16_t RCD_EndDevice_Addr

Variable to store the end device short address.

6.2.17. static instanceId_t mMacInstance[gMacInstancesCnt_c]

The MAC instance.

Variables

The next list of variables are also used for end device or routers in its respective application layer and they are described in end device documentation chapter.

- static const gFsciSerialConfig_t mFsciSerials[ ]
- static uint8_t mFsciInterface[gMacInstancesCnt_c]
- static nwkToMcpsMessage_t *mpPacket;
- static uint8_t mMsduHandle;
- static uint8_t mcPendingPackets;
- static anchor_t mMlmeNwkInputQueue;
- static anchor_t mMcpsNwkInputQueue;
- static const uint64_t mExtendedAddress;
- static instanceId_t macInstance;
- static uint8_t interfaceId;
- event_t mAppEvent;
- uint8_t gState;
- uint8_t key128[128/8]
- uint8_t ReceivedPayload [16];
- uint8_t DecryptedPayload [16];

6.3. Application defines for coordinator

6.3.1. #define Check4Router_TimeInterval_c

This is the time interval in milliseconds for the timer which checks if a router is still alive.

Default value: 19000.

6.4. Application enums for coordinator

- **typedef enum**{

    Router1_ShortAddr = 2,

    Router2_ShortAddr = 3,

    Router3_ShortAddr = 4,

    Router4_ShortAddr = 5,

  } **Routers_Addr_t;** /*Short adrress of routers*/

- **typedef enum**{

    Disconnected = 0,

    Connected = 1,

  } **Routers_Status_t;** /*Connection status of routers*/

- typedef enum{

    ConnectedBulbs_Command = 5,

    KeepAlive_Command = 6,

  } RCD_Commands_Coordinator_t;

/*Application commands handled by coordinator in App_HandleMcpsInput funtion*

7. Revision History

Table 1. **Revision history**

Revision number	Date	Substantive changes
0	10/2016	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: KW01RCDRDSWRM
Rev. 0
10/2016