**Zigbee组播通信实验——实验报告**

**（项目编号：07012029学时：2）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程** | 物联网传输技术 | **实验项目** | Zigbee组播通信 | **成绩** |  |
| **专业班级** | 15级物联网工程 | **学号** | **201531060570** | **批阅日期** |  |
| **姓名** | **石华** | **实验日期** | 2017/12/20 | **指导教师** | 邹正伟 |

**一【实验目的】**

1. 熟悉Zigbee协议栈Z-Stack
2. 熟悉Zigbee无线组播通信流程，掌握任务的事件添加方法，定时器方法

3、掌握组播通信的实现方法

**二【实验内容】**

**1、**协调器周期性地以**组**播的形式向终端节点发送数据（每个5秒发送组播数据一次），路由器（终端）节点接受到数据后，使开发板的LED状态翻转，同时向协调器发送字符串“Router received! ”。协调器接收到路由器节点发送回的数据后，通过串口输出到PC的串口调试助手。

**2、工具/原料**

* IAR Embedded Workbench for MCS-51
* CC2530 Zigbee开发套件
* CCDebuger调试器

**3、方法/步骤**

**3.1新建工程**

**3.2相关知识**

**3.3完整代码**

Coordinator.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Filename: SampleApp.c

Revised: $Date: 2009-03-18 15:56:27 -0700 (Wed, 18 Mar 2009) $

Revision: $Revision: 19453 $

Description: Sample Application (no Profile).

Copyright 2007 Texas Instruments Incorporated. All rights reserved.

IMPORTANT: Your use of this Software is limited to those specific rights

granted under the terms of a software license agreement between the user

who downloaded the software, his/her employer (which must be your employer)

and Texas Instruments Incorporated (the "License"). You may not use this

Software unless you agree to abide by the terms of the License. The License

limits your use, and you acknowledge, that the Software may not be modified,

copied or distributed unless embedded on a Texas Instruments microcontroller

or used solely and exclusively in conjunction with a Texas Instruments radio

frequency transceiver, which is integrated into your product. Other than for

the foregoing purpose, you may not use, reproduce, copy, prepare derivative

works of, modify, distribute, perform, display or sell this Software and/or

its documentation for any purpose.

YOU FURTHER ACKNOWLEDGE AND AGREE THAT THE SOFTWARE AND DOCUMENTATION ARE

PROVIDED 揂S IS?WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED,

INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE,

NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL

TEXAS INSTRUMENTS OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT,

NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER

LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE

OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT

OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

Should you have any questions regarding your right to use this Software,

contact Texas Instruments Incorporated at www.TI.com.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This application isn't intended to do anything useful, it is

intended to be a simple example of an application's structure.

This application sends it's messages either as broadcast or

broadcast filtered group messages. The other (more normal)

message addressing is unicast. Most of the other sample

applications are written to support the unicast message model.

Key control:

SW1: Sends a flash command to all devices in Group 1.

SW2: Adds/Removes (toggles) this device in and out

of Group 1. This will enable and disable the

reception of the flash command.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* INCLUDES

\*/

#include "OSAL.h"

#include "ZGlobals.h"

#include "AF.h"

#include "aps\_groups.h"

#include "ZDApp.h"

#include "SampleApp.h"

#include "SampleAppHw.h"

#include "OnBoard.h"

/\* HAL \*/

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MACROS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CONSTANTS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* TYPEDEFS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GLOBAL VARIABLES

\*/

// This list should be filled with Application specific Cluster IDs.

const cId\_t SampleApp\_ClusterList[SAMPLEAPP\_MAX\_CLUSTERS] =

{

SAMPLEAPP\_PERIODIC\_CLUSTERID,

SAMPLEAPP\_FLASH\_CLUSTERID

};

const SimpleDescriptionFormat\_t SampleApp\_SimpleDesc =

{

SAMPLEAPP\_ENDPOINT, // int Endpoint;

SAMPLEAPP\_PROFID, // uint16 AppProfId[2];

SAMPLEAPP\_DEVICEID, // uint16 AppDeviceId[2];

SAMPLEAPP\_DEVICE\_VERSION, // int AppDevVer:4;

SAMPLEAPP\_FLAGS, // int AppFlags:4;

SAMPLEAPP\_MAX\_CLUSTERS, // uint8 AppNumInClusters;

(cId\_t \*)SampleApp\_ClusterList, // uint8 \*pAppInClusterList;

SAMPLEAPP\_MAX\_CLUSTERS, // uint8 AppNumInClusters;

(cId\_t \*)SampleApp\_ClusterList // uint8 \*pAppInClusterList;

};

// This is the Endpoint/Interface description. It is defined here, but

// filled-in in SampleApp\_Init(). Another way to go would be to fill

// in the structure here and make it a "const" (in code space). The

// way it's defined in this sample app it is define in RAM.

endPointDesc\_t SampleApp\_epDesc;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EXTERNAL VARIABLES

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EXTERNAL FUNCTIONS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL VARIABLES

\*/

uint8 SampleApp\_TaskID; // Task ID for internal task/event processing

// This variable will be received when

// SampleApp\_Init() is called.

devStates\_t SampleApp\_NwkState;

uint8 SampleApp\_TransID; // This is the unique message ID (counter)

afAddrType\_t SampleApp\_Periodic\_DstAddr;

afAddrType\_t SampleApp\_Flash\_DstAddr;

aps\_Group\_t SampleApp\_Group;

uint8 SampleAppPeriodicCounter = 0;

uint8 SampleAppFlashCounter = 0;

aps\_Group\_t MyGroup1,MyGroup2;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL FUNCTIONS

\*/

void SampleApp\_HandleKeys( uint8 shift, uint8 keys );

void SampleApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pckt );

void SampleApp\_SendPeriodicMessage( void );

void SampleApp\_SendFlashMessage( uint16 flashTime );

void GenericApp\_SendTheMessageG1(void);

void GenericApp\_SendTheMessageG2(void);

void GenericApp\_SendTheMessageBc(void);

static void rxCB(uint8 p,uint8 e);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* NETWORK LAYER CALLBACKS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PUBLIC FUNCTIONS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_Init

\*

\* @brief Initialization function for the Generic App Task.

\* This is called during initialization and should contain

\* any application specific initialization (ie. hardware

\* initialization/setup, table initialization, power up

\* notificaiton ... ).

\*

\* @param task\_id - the ID assigned by OSAL. This ID should be

\* used to send messages and set timers.

\*

\* @return none

\*/

void SampleApp\_Init( uint8 task\_id )

{

SampleApp\_TaskID = task\_id;

SampleApp\_NwkState = DEV\_INIT;

SampleApp\_TransID = 0;

// Device hardware initialization can be added here or in main() (Zmain.c).

// If the hardware is application specific - add it here.

// If the hardware is other parts of the device add it in main().

#if defined ( BUILD\_ALL\_DEVICES )

// The "Demo" target is setup to have BUILD\_ALL\_DEVICES and HOLD\_AUTO\_START

// We are looking at a jumper (defined in SampleAppHw.c) to be jumpered

// together - if they are - we will start up a coordinator. Otherwise,

// the device will start as a router.

if ( readCoordinatorJumper() )

zgDeviceLogicalType = ZG\_DEVICETYPE\_COORDINATOR;

else

zgDeviceLogicalType = ZG\_DEVICETYPE\_ROUTER;

#endif // BUILD\_ALL\_DEVICES

#if defined ( HOLD\_AUTO\_START )

// HOLD\_AUTO\_START is a compile option that will surpress ZDApp

// from starting the device and wait for the application to

// start the device.

ZDOInitDevice(0);

#endif

// Setup for the periodic message's destination address

// Broadcast to everyone

/\*SampleApp\_Periodic\_DstAddr.addrMode = (afAddrMode\_t)AddrBroadcast;

SampleApp\_Periodic\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Periodic\_DstAddr.addr.shortAddr = 0xFFFF;

// Setup for the flash command's destination address - Group 1

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)afAddrGroup;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = SAMPLEAPP\_FLASH\_GROUP;

\*/

// Fill out the endpoint description.

SampleApp\_epDesc.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_epDesc.task\_id = &SampleApp\_TaskID;

SampleApp\_epDesc.simpleDesc

= (SimpleDescriptionFormat\_t \*)&SampleApp\_SimpleDesc;

SampleApp\_epDesc.latencyReq = noLatencyReqs;

// Register the endpoint description with the AF

afRegister( &SampleApp\_epDesc );

halUARTCfg\_t myuartCfg;

myuartCfg.configured=TRUE;

myuartCfg.baudRate=HAL\_UART\_BR\_115200;

myuartCfg.flowControl=FALSE;

myuartCfg.callBackFunc=rxCB;

HalUARTOpen(0,&myuartCfg);

MyGroup1.ID=0x0001;

MyGroup1.name[0]=6;

osal\_memcpy(&(MyGroup1.name[1]),"Group1",6);

MyGroup2.ID=0x0002;

MyGroup2.name[0]=6;

osal\_memcpy(&(MyGroup2.name[1]),"Group2",6);

}

static void rxCB(uint8 p,uint8 e)

{

uint8 cmd[2];

HalUARTRead(0,cmd,2);

if(osal\_memcmp(cmd,"G1",2))

{

GenericApp\_SendTheMessageG1();

HalUARTWrite(0,"s\_G1",4);

}

else if(osal\_memcmp(cmd,"G2",2))

{

GenericApp\_SendTheMessageG2();

HalUARTWrite(0,"s\_G2",4);

}

else if(osal\_memcmp(cmd,"Bc",2))

{

GenericApp\_SendTheMessageBc();

HalUARTWrite(0,"s\_Bc",4);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_ProcessEvent

\*

\* @brief Generic Application Task event processor. This function

\* is called to process all events for the task. Events

\* include timers, messages and any other user defined events.

\*

\* @param task\_id - The OSAL assigned task ID.

\* @param events - events to process. This is a bit map and can

\* contain more than one event.

\*

\* @return none

\*/

uint16 SampleApp\_ProcessEvent( uint8 task\_id, uint16 events )

{

afIncomingMSGPacket\_t \*MSGpkt;

(void)task\_id; // Intentionally unreferenced parameter

if ( events & SYS\_EVENT\_MSG )

{

MSGpkt = (afIncomingMSGPacket\_t \*)osal\_msg\_receive( SampleApp\_TaskID );

while ( MSGpkt )

{

switch ( MSGpkt->hdr.event )

{

// Received when a key is pressed

case KEY\_CHANGE:

SampleApp\_HandleKeys( ((keyChange\_t \*)MSGpkt)->state, ((keyChange\_t \*)MSGpkt)->keys );

break;

// Received when a messages is received (OTA) for this endpoint

case AF\_INCOMING\_MSG\_CMD:

SampleApp\_MessageMSGCB( MSGpkt );

break;

// Received whenever the device changes state in the network

case ZDO\_STATE\_CHANGE:

SampleApp\_NwkState = (devStates\_t)(MSGpkt->hdr.status);

if ( (SampleApp\_NwkState == DEV\_ZB\_COORD)

|| (SampleApp\_NwkState == DEV\_ROUTER)

|| (SampleApp\_NwkState == DEV\_END\_DEVICE) )

{

// Start sending the periodic message in a regular interval.

osal\_start\_timerEx( SampleApp\_TaskID,

SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT,

SAMPLEAPP\_SEND\_PERIODIC\_MSG\_TIMEOUT );

}

else

{

// Device is no longer in the network

}

break;

default:

break;

}

// Release the memory

osal\_msg\_deallocate( (uint8 \*)MSGpkt );

// Next - if one is available

MSGpkt = (afIncomingMSGPacket\_t \*)osal\_msg\_receive( SampleApp\_TaskID );

}

// return unprocessed events

return (events ^ SYS\_EVENT\_MSG);

}

// Send a message out - This event is generated by a timer

// (setup in SampleApp\_Init()).

if ( events & SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT )

{

// Send the periodic message

SampleApp\_SendPeriodicMessage();

// Setup to send message again in normal period (+ a little jitter)

osal\_start\_timerEx( SampleApp\_TaskID, SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT,

(SAMPLEAPP\_SEND\_PERIODIC\_MSG\_TIMEOUT + (osal\_rand() & 0x00FF)) );

// return unprocessed events

return (events ^ SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT);

}

// Discard unknown events

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Event Generation Functions

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_HandleKeys

\*

\* @brief Handles all key events for this device.

\*

\* @param shift - true if in shift/alt.

\* @param keys - bit field for key events. Valid entries:

\* HAL\_KEY\_SW\_2

\* HAL\_KEY\_SW\_1

\*

\* @return none

\*/

void SampleApp\_HandleKeys( uint8 shift, uint8 keys )

{

(void)shift; // Intentionally unreferenced parameter

if ( keys & HAL\_KEY\_SW\_1 )

{

/\* This key sends the Flash Command is sent to Group 1.

\* This device will not receive the Flash Command from this

\* device (even if it belongs to group 1).

\*/

SampleApp\_SendFlashMessage( SAMPLEAPP\_FLASH\_DURATION );

}

if ( keys & HAL\_KEY\_SW\_2 )

{

/\* The Flashr Command is sent to Group 1.

\* This key toggles this device in and out of group 1.

\* If this device doesn't belong to group 1, this application

\* will not receive the Flash command sent to group 1.

\*/

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, SAMPLEAPP\_FLASH\_GROUP );

if ( grp )

{

// Remove from the group

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, SAMPLEAPP\_FLASH\_GROUP );

}

else

{

// Add to the flash group

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &SampleApp\_Group );

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL FUNCTIONS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_MessageMSGCB

\*

\* @brief Data message processor callback. This function processes

\* any incoming data - probably from other devices. So, based

\* on cluster ID, perform the intended action.

\*

\* @param none

\*

\* @return none

\*/

void SampleApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pkt )

{

char \*recBuf;

osal\_memcpy(recBuf,pkt->cmd.Data,4);

HalUARTWrite(0,recBuf,4);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_SendPeriodicMessage

\*

\* @brief Send the periodic message.

\*

\* @param none

\*

\* @return none

\*/

void SampleApp\_SendPeriodicMessage( void )

{

if ( AF\_DataRequest( &SampleApp\_Periodic\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_PERIODIC\_CLUSTERID,

1,

(uint8\*)&SampleAppPeriodicCounter,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_SendFlashMessage

\*

\* @brief Send the flash message to group 1.

\*

\* @param flashTime - in milliseconds

\*

\* @return none

\*/

void SampleApp\_SendFlashMessage( uint16 flashTime )

{

uint8 buffer[3];

buffer[0] = (uint8)(SampleAppFlashCounter++);

buffer[1] = LO\_UINT16( flashTime );

buffer[2] = HI\_UINT16( flashTime );

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

3,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

void GenericApp\_SendTheMessageG1()

{

uint8 buffer[2];

buffer[0]='G';

buffer[1]='1';

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID);

}

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &MyGroup1);

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)afAddrGroup;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = MyGroup1.ID;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

2,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

void GenericApp\_SendTheMessageG2()

{

uint8 buffer[2];

buffer[0]='G';

buffer[1]='2';

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID);

}

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &MyGroup2);

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)afAddrGroup;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = MyGroup2.ID;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

2,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

void GenericApp\_SendTheMessageBc()

{

uint8 buffer[2];

buffer[0]='B';

buffer[1]='c';

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID);

}

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID);

}

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)AddrBroadcast;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = 0xFFFF;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

2,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Router.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Filename: SampleApp.c

Revised: $Date: 2009-03-18 15:56:27 -0700 (Wed, 18 Mar 2009) $

Revision: $Revision: 19453 $

Description: Sample Application (no Profile).

Copyright 2007 Texas Instruments Incorporated. All rights reserved.

IMPORTANT: Your use of this Software is limited to those specific rights

granted under the terms of a software license agreement between the user

who downloaded the software, his/her employer (which must be your employer)

and Texas Instruments Incorporated (the "License"). You may not use this

Software unless you agree to abide by the terms of the License. The License

limits your use, and you acknowledge, that the Software may not be modified,

copied or distributed unless embedded on a Texas Instruments microcontroller

or used solely and exclusively in conjunction with a Texas Instruments radio

frequency transceiver, which is integrated into your product. Other than for

the foregoing purpose, you may not use, reproduce, copy, prepare derivative

works of, modify, distribute, perform, display or sell this Software and/or

its documentation for any purpose.

YOU FURTHER ACKNOWLEDGE AND AGREE THAT THE SOFTWARE AND DOCUMENTATION ARE

PROVIDED 揂S IS?WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED,

INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE,

NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL

TEXAS INSTRUMENTS OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT,

NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER

LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE

OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT

OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

Should you have any questions regarding your right to use this Software,

contact Texas Instruments Incorporated at www.TI.com.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This application isn't intended to do anything useful, it is

intended to be a simple example of an application's structure.

This application sends it's messages either as broadcast or

broadcast filtered group messages. The other (more normal)

message addressing is unicast. Most of the other sample

applications are written to support the unicast message model.

Key control:

SW1: Sends a flash command to all devices in Group 1.

SW2: Adds/Removes (toggles) this device in and out

of Group 1. This will enable and disable the

reception of the flash command.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* INCLUDES

\*/

#include "OSAL.h"

#include "ZGlobals.h"

#include "AF.h"

#include "aps\_groups.h"

#include "ZDApp.h"

#include "SampleApp.h"

#include "SampleAppHw.h"

#include "OnBoard.h"

/\* HAL \*/

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MACROS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CONSTANTS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* TYPEDEFS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GLOBAL VARIABLES

\*/

// This list should be filled with Application specific Cluster IDs.

const cId\_t SampleApp\_ClusterList[SAMPLEAPP\_MAX\_CLUSTERS] =

{

SAMPLEAPP\_PERIODIC\_CLUSTERID,

SAMPLEAPP\_FLASH\_CLUSTERID

};

const SimpleDescriptionFormat\_t SampleApp\_SimpleDesc =

{

SAMPLEAPP\_ENDPOINT, // int Endpoint;

SAMPLEAPP\_PROFID, // uint16 AppProfId[2];

SAMPLEAPP\_DEVICEID, // uint16 AppDeviceId[2];

SAMPLEAPP\_DEVICE\_VERSION, // int AppDevVer:4;

SAMPLEAPP\_FLAGS, // int AppFlags:4;

SAMPLEAPP\_MAX\_CLUSTERS, // uint8 AppNumInClusters;

(cId\_t \*)SampleApp\_ClusterList, // uint8 \*pAppInClusterList;

SAMPLEAPP\_MAX\_CLUSTERS, // uint8 AppNumInClusters;

(cId\_t \*)SampleApp\_ClusterList // uint8 \*pAppInClusterList;

};

// This is the Endpoint/Interface description. It is defined here, but

// filled-in in SampleApp\_Init(). Another way to go would be to fill

// in the structure here and make it a "const" (in code space). The

// way it's defined in this sample app it is define in RAM.

endPointDesc\_t SampleApp\_epDesc;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EXTERNAL VARIABLES

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EXTERNAL FUNCTIONS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL VARIABLES

\*/

uint8 SampleApp\_TaskID; // Task ID for internal task/event processing

// This variable will be received when

// SampleApp\_Init() is called.

devStates\_t SampleApp\_NwkState;

uint8 SampleApp\_TransID; // This is the unique message ID (counter)

afAddrType\_t SampleApp\_Periodic\_DstAddr;

afAddrType\_t SampleApp\_Flash\_DstAddr;

aps\_Group\_t SampleApp\_Group;

uint8 SampleAppPeriodicCounter = 0;

uint8 SampleAppFlashCounter = 0;

aps\_Group\_t MyGroup1,MyGroup2;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL FUNCTIONS

\*/

void SampleApp\_HandleKeys( uint8 shift, uint8 keys );

void SampleApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pckt );

void SampleApp\_SendPeriodicMessage( void );

void SampleApp\_SendFlashMessage( uint16 flashTime );

void GenericApp\_SendTheMessageG1(void);

void GenericApp\_SendTheMessageG2(void);

void GenericApp\_SendTheMessageBc(void);

static void rxCB(uint8 p,uint8 e);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* NETWORK LAYER CALLBACKS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PUBLIC FUNCTIONS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_Init

\*

\* @brief Initialization function for the Generic App Task.

\* This is called during initialization and should contain

\* any application specific initialization (ie. hardware

\* initialization/setup, table initialization, power up

\* notificaiton ... ).

\*

\* @param task\_id - the ID assigned by OSAL. This ID should be

\* used to send messages and set timers.

\*

\* @return none

\*/

void SampleApp\_Init( uint8 task\_id )

{

SampleApp\_TaskID = task\_id;

SampleApp\_NwkState = DEV\_INIT;

SampleApp\_TransID = 0;

// Device hardware initialization can be added here or in main() (Zmain.c).

// If the hardware is application specific - add it here.

// If the hardware is other parts of the device add it in main().

#if defined ( BUILD\_ALL\_DEVICES )

// The "Demo" target is setup to have BUILD\_ALL\_DEVICES and HOLD\_AUTO\_START

// We are looking at a jumper (defined in SampleAppHw.c) to be jumpered

// together - if they are - we will start up a coordinator. Otherwise,

// the device will start as a router.

if ( readCoordinatorJumper() )

zgDeviceLogicalType = ZG\_DEVICETYPE\_COORDINATOR;

else

zgDeviceLogicalType = ZG\_DEVICETYPE\_ROUTER;

#endif // BUILD\_ALL\_DEVICES

#if defined ( HOLD\_AUTO\_START )

// HOLD\_AUTO\_START is a compile option that will surpress ZDApp

// from starting the device and wait for the application to

// start the device.

ZDOInitDevice(0);

#endif

// Setup for the periodic message's destination address

// Broadcast to everyone

/\*SampleApp\_Periodic\_DstAddr.addrMode = (afAddrMode\_t)AddrBroadcast;

SampleApp\_Periodic\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Periodic\_DstAddr.addr.shortAddr = 0xFFFF;

// Setup for the flash command's destination address - Group 1

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)afAddrGroup;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = SAMPLEAPP\_FLASH\_GROUP;

\*/

// Fill out the endpoint description.

SampleApp\_epDesc.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_epDesc.task\_id = &SampleApp\_TaskID;

SampleApp\_epDesc.simpleDesc

= (SimpleDescriptionFormat\_t \*)&SampleApp\_SimpleDesc;

SampleApp\_epDesc.latencyReq = noLatencyReqs;

// Register the endpoint description with the AF

afRegister( &SampleApp\_epDesc );

/\*

halUARTCfg\_t myuartCfg;

myuartCfg.configured=TRUE;

myuartCfg.baudRate=HAL\_UART\_BR\_115200;

myuartCfg.flowControl=FALSE;

myuartCfg.callBackFunc=rxCB;

HalUARTOpen(0,&myuartCfg);

\*/

MyGroup1.ID=0x0001;

MyGroup1.name[0]=6;

osal\_memcpy(&(MyGroup1.name[1]),"Group1",6);

MyGroup2.ID=0x0002;

MyGroup2.name[0]=6;

osal\_memcpy(&(MyGroup2.name[1]),"Group2",6);

}

static void rxCB(uint8 p,uint8 e)

{

uint8 cmd[2];

HalUARTRead(0,cmd,2);

if(osal\_memcmp(cmd,"G1",2))

{

GenericApp\_SendTheMessageG1();

HalUARTWrite(0,"s\_G1",4);

}

else if(osal\_memcmp(cmd,"G2",2))

{

GenericApp\_SendTheMessageG2();

HalUARTWrite(0,"s\_G2",4);

}

else if(osal\_memcmp(cmd,"Bc",2))

{

GenericApp\_SendTheMessageBc();

HalUARTWrite(0,"s\_Bc",4);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_ProcessEvent

\*

\* @brief Generic Application Task event processor. This function

\* is called to process all events for the task. Events

\* include timers, messages and any other user defined events.

\*

\* @param task\_id - The OSAL assigned task ID.

\* @param events - events to process. This is a bit map and can

\* contain more than one event.

\*

\* @return none

\*/

uint16 SampleApp\_ProcessEvent( uint8 task\_id, uint16 events )

{

afIncomingMSGPacket\_t \*MSGpkt;

(void)task\_id; // Intentionally unreferenced parameter

if ( events & SYS\_EVENT\_MSG )

{

MSGpkt = (afIncomingMSGPacket\_t \*)osal\_msg\_receive( SampleApp\_TaskID );

while ( MSGpkt )

{

switch ( MSGpkt->hdr.event )

{

// Received when a key is pressed

case KEY\_CHANGE:

SampleApp\_HandleKeys( ((keyChange\_t \*)MSGpkt)->state, ((keyChange\_t \*)MSGpkt)->keys );

break;

// Received when a messages is received (OTA) for this endpoint

case AF\_INCOMING\_MSG\_CMD:

SampleApp\_MessageMSGCB( MSGpkt );

break;

// Received whenever the device changes state in the network

case ZDO\_STATE\_CHANGE:

SampleApp\_NwkState = (devStates\_t)(MSGpkt->hdr.status);

if ( SampleApp\_NwkState == DEV\_ROUTER

)

{

//aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &MyGroup2);

}

else

{

// Device is no longer in the network

}

break;

default:

break;

}

// Release the memory

osal\_msg\_deallocate( (uint8 \*)MSGpkt );

// Next - if one is available

MSGpkt = (afIncomingMSGPacket\_t \*)osal\_msg\_receive( SampleApp\_TaskID );

}

// return unprocessed events

return (events ^ SYS\_EVENT\_MSG);

}

// Send a message out - This event is generated by a timer

// (setup in SampleApp\_Init()).

if ( events & SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT )

{

// Send the periodic message

SampleApp\_SendPeriodicMessage();

// Setup to send message again in normal period (+ a little jitter)

osal\_start\_timerEx( SampleApp\_TaskID, SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT,

(SAMPLEAPP\_SEND\_PERIODIC\_MSG\_TIMEOUT + (osal\_rand() & 0x00FF)) );

// return unprocessed events

return (events ^ SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT);

}

// Discard unknown events

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Event Generation Functions

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_HandleKeys

\*

\* @brief Handles all key events for this device.

\*

\* @param shift - true if in shift/alt.

\* @param keys - bit field for key events. Valid entries:

\* HAL\_KEY\_SW\_2

\* HAL\_KEY\_SW\_1

\*

\* @return none

\*/

void SampleApp\_HandleKeys( uint8 shift, uint8 keys )

{

(void)shift; // Intentionally unreferenced parameter

if ( keys & HAL\_KEY\_SW\_1 )

{

/\* This key sends the Flash Command is sent to Group 1.

\* This device will not receive the Flash Command from this

\* device (even if it belongs to group 1).

\*/

SampleApp\_SendFlashMessage( SAMPLEAPP\_FLASH\_DURATION );

}

if ( keys & HAL\_KEY\_SW\_2 )

{

/\* The Flashr Command is sent to Group 1.

\* This key toggles this device in and out of group 1.

\* If this device doesn't belong to group 1, this application

\* will not receive the Flash command sent to group 1.

\*/

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, SAMPLEAPP\_FLASH\_GROUP );

if ( grp )

{

// Remove from the group

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, SAMPLEAPP\_FLASH\_GROUP );

}

else

{

// Add to the flash group

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &SampleApp\_Group );

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL FUNCTIONS

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_MessageMSGCB

\*

\* @brief Data message processor callback. This function processes

\* any incoming data - probably from other devices. So, based

\* on cluster ID, perform the intended action.

\*

\* @param none

\*

\* @return none

\*/

void SampleApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pkt )

{

char \*buffer="ImBc";

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)Addr16Bit;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = 0x0000;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

5,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_SendPeriodicMessage

\*

\* @brief Send the periodic message.

\*

\* @param none

\*

\* @return none

\*/

void SampleApp\_SendPeriodicMessage( void )

{

if ( AF\_DataRequest( &SampleApp\_Periodic\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_PERIODIC\_CLUSTERID,

1,

(uint8\*)&SampleAppPeriodicCounter,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @fn SampleApp\_SendFlashMessage

\*

\* @brief Send the flash message to group 1.

\*

\* @param flashTime - in milliseconds

\*

\* @return none

\*/

void SampleApp\_SendFlashMessage( uint16 flashTime )

{

uint8 buffer[3];

buffer[0] = (uint8)(SampleAppFlashCounter++);

buffer[1] = LO\_UINT16( flashTime );

buffer[2] = HI\_UINT16( flashTime );

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

3,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

void GenericApp\_SendTheMessageG1()

{

uint8 buffer[2];

buffer[0]='G';

buffer[1]='1';

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID);

}

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &MyGroup1);

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)afAddrGroup;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = MyGroup1.ID;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

2,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

void GenericApp\_SendTheMessageG2()

{

uint8 buffer[2];

buffer[0]='G';

buffer[1]='2';

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID);

}

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &MyGroup2);

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)afAddrGroup;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = MyGroup2.ID;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

2,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

void GenericApp\_SendTheMessageBc()

{

uint8 buffer[2];

buffer[0]='B';

buffer[1]='c';

aps\_Group\_t \*grp;

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup2.ID);

}

grp = aps\_FindGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID );

if ( grp )

{

aps\_RemoveGroup( SAMPLEAPP\_ENDPOINT, MyGroup1.ID);

}

SampleApp\_Flash\_DstAddr.addrMode = (afAddrMode\_t)AddrBroadcast;

SampleApp\_Flash\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_Flash\_DstAddr.addr.shortAddr = 0xFFFF;

if ( AF\_DataRequest( &SampleApp\_Flash\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_FLASH\_CLUSTERID,

2,

buffer,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

}

else

{

// Error occurred in request to send.

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**2.4下载调试**

**注意：**项目配置

配置完成，选择对应的调试模块（按键选择），下载调试（调试器灯为红色，调试器工作正常，可按“reset”键重置；若调试器工作不正常，请查看电脑“设备管理”，卸载并重新安装调试器驱动）。

运行程序，完成实验要求。

**运行截图：**

**总结：**

通过本次实验，学会了ZigBee协议栈的组播通信，对组播通信的原理和实现方法有了了解。