**Zigbee广播通信实验——实验报告**

**（项目编号：07012028学时：2）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程** | 物联网传输技术 | **实验项目** | Zigbee广播通信 | **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** | 2017-12-13 | **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无线广播通信流程，掌握任务的事件添加方法，定时器方法

3、掌握广播通信的实现方法

**二【实验内容】**

**1、**协调器周期性地以广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**

### Gennericapp：

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

Filename: GenericApp.c

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

Revision: $Revision: 19453 $

Description: Generic Application (no Profile).

Copyright 2004-2009 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 "Hello World" to another "Generic"

application every 15 seconds. The application will also

receive "Hello World" packets.

The "Hello World" messages are sent/received as MSG type message.

This applications doesn't have a profile, so it handles everything

directly - itself.

Key control:

SW1:

SW2: initiates end device binding

SW3:

SW4: initiates a match description request

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

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

\* INCLUDES

\*/

#include "OSAL.h"

#include "AF.h"

#include "ZDApp.h"

#include "ZDObject.h"

#include "ZDProfile.h"

#include "GenericApp.h"

#include "DebugTrace.h"

#if !defined( WIN32 )

#include "OnBoard.h"

#endif

/\* HAL \*/

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

#include "hal\_uart.h"

#define SEND\_TO\_ALL\_EVENT 0x01 //ss-1:定义发送事件

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

\* MACROS

\*/

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

\* CONSTANTS

\*/

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

\* TYPEDEFS

\*/

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

\* GLOBAL VARIABLES

\*/

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

const cId\_t GenericApp\_ClusterList[GENERICAPP\_MAX\_CLUSTERS] =

{

GENERICAPP\_CLUSTERID

};

const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc =

{

GENERICAPP\_ENDPOINT, // int Endpoint;

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

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

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

GENERICAPP\_FLAGS, // int AppFlags:4;

GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;

(cId\_t \*)GenericApp\_ClusterList, // byte \*pAppInClusterList;

GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;

(cId\_t \*)GenericApp\_ClusterList // byte \*pAppInClusterList;

};

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

// filled-in in GenericApp\_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 GenericApp\_epDesc;

devStates\_t GenericApp\_NwkState;//ss-2:存储网络状态的变量

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

\* EXTERNAL VARIABLES

\*/

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

\* EXTERNAL FUNCTIONS

\*/

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

\* LOCAL VARIABLES

\*/

byte GenericApp\_TaskID; // Task ID for internal task/event processing

// This variable will be received when

// GenericApp\_Init() is called.

devStates\_t GenericApp\_NwkState;

byte GenericApp\_TransID; // This is the unique message ID (counter)

afAddrType\_t GenericApp\_DstAddr;

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

\* LOCAL FUNCTIONS

\*/

void GenericApp\_ProcessZDOMsgs( zdoIncomingMsg\_t \*inMsg );

void GenericApp\_HandleKeys( byte shift, byte keys );

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

void GenericApp\_SendTheMessage( void );

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

\* NETWORK LAYER CALLBACKS

\*/

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

\* PUBLIC FUNCTIONS

\*/

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

\* @fn GenericApp\_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 GenericApp\_Init( byte task\_id )

{

halUARTCfg\_t uartConfig;//ss-03:串口定义

GenericApp\_TaskID = task\_id;

GenericApp\_NwkState = DEV\_INIT;

GenericApp\_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().

GenericApp\_DstAddr.addrMode = (afAddrMode\_t)AddrNotPresent;

GenericApp\_DstAddr.endPoint = 0;

GenericApp\_DstAddr.addr.shortAddr = 0;

// Fill out the endpoint description.

GenericApp\_epDesc.endPoint = GENERICAPP\_ENDPOINT;

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

GenericApp\_epDesc.simpleDesc

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

GenericApp\_epDesc.latencyReq = noLatencyReqs;

// Register the endpoint description with the AF

afRegister( &GenericApp\_epDesc );

//ss-03:串口初始化

uartConfig.configured = TRUE;

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

uartConfig.flowControl = FALSE;

uartConfig.callBackFunc = NULL;

HalUARTOpen(0,&uartConfig);

// Register for all key events - This app will handle all key events

RegisterForKeys( GenericApp\_TaskID );

// Update the display

#if defined ( LCD\_SUPPORTED )

HalLcdWriteString( "GenericApp", HAL\_LCD\_LINE\_1 );

#endif

ZDO\_RegisterForZDOMsg( GenericApp\_TaskID, End\_Device\_Bind\_rsp );

ZDO\_RegisterForZDOMsg( GenericApp\_TaskID, Match\_Desc\_rsp );

}

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

\* @fn GenericApp\_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 GenericApp\_ProcessEvent( byte task\_id, UINT16 events )

{

afIncomingMSGPacket\_t \*MSGpkt;

afDataConfirm\_t \*afDataConfirm;

// Data Confirmation message fields

byte sentEP;

ZStatus\_t sentStatus;

byte sentTransID; // This should match the value sent

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

if ( events & SYS\_EVENT\_MSG )

{

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

while ( MSGpkt )

{

switch ( MSGpkt->hdr.event )

{

case ZDO\_CB\_MSG:

GenericApp\_ProcessZDOMsgs( (zdoIncomingMsg\_t \*)MSGpkt );

break;

case KEY\_CHANGE:

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

break;

case AF\_DATA\_CONFIRM\_CMD:

// This message is received as a confirmation of a data packet sent.

// The status is of ZStatus\_t type [defined in ZComDef.h]

// The message fields are defined in AF.h

afDataConfirm = (afDataConfirm\_t \*)MSGpkt;

sentEP = afDataConfirm->endpoint;

sentStatus = afDataConfirm->hdr.status;

sentTransID = afDataConfirm->transID;

(void)sentEP;

(void)sentTransID;

// Action taken when confirmation is received.

if ( sentStatus != ZSuccess )

{

// The data wasn't delivered -- Do something

}

break;

case AF\_INCOMING\_MSG\_CMD:

GenericApp\_MessageMSGCB( MSGpkt );

break;

case ZDO\_STATE\_CHANGE:

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

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

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

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

{

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

/\*

osal\_start\_timerEx( GenericApp\_TaskID,

GENERICAPP\_SEND\_MSG\_EVT,

GENERICAPP\_SEND\_MSG\_TIMEOUT );

\*/

//ss-04:定时器加入自定义事件

osal\_start\_timerEx( GenericApp\_TaskID,

SEND\_TO\_ALL\_EVENT,

5000 );

}

break;

default:

break;

}

// Release the memory

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

// Next

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

}

// return unprocessed events

return (events ^ SYS\_EVENT\_MSG);

}

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

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

//ss-5:数据发送事件处理

if ( events & SEND\_TO\_ALL\_EVENT )

{

// Send "the" message

GenericApp\_SendTheMessage();

// Setup to send message again

osal\_start\_timerEx( GenericApp\_TaskID,

SEND\_TO\_ALL\_EVENT,

5000 );

// return unprocessed events

return (events ^ SEND\_TO\_ALL\_EVENT);

}

// Discard unknown events

return 0;

}

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

\* Event Generation Functions

\*/

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

\* @fn GenericApp\_ProcessZDOMsgs()

\*

\* @brief Process response messages

\*

\* @param none

\*

\* @return none

\*/

void GenericApp\_ProcessZDOMsgs( zdoIncomingMsg\_t \*inMsg )

{

switch ( inMsg->clusterID )

{

case End\_Device\_Bind\_rsp:

if ( ZDO\_ParseBindRsp( inMsg ) == ZSuccess )

{

// Light LED

HalLedSet( HAL\_LED\_4, HAL\_LED\_MODE\_ON );

}

#if defined(BLINK\_LEDS)

else

{

// Flash LED to show failure

HalLedSet ( HAL\_LED\_4, HAL\_LED\_MODE\_FLASH );

}

#endif

break;

case Match\_Desc\_rsp:

{

ZDO\_ActiveEndpointRsp\_t \*pRsp = ZDO\_ParseEPListRsp( inMsg );

if ( pRsp )

{

if ( pRsp->status == ZSuccess && pRsp->cnt )

{

GenericApp\_DstAddr.addrMode = (afAddrMode\_t)Addr16Bit;

GenericApp\_DstAddr.addr.shortAddr = pRsp->nwkAddr;

// Take the first endpoint, Can be changed to search through endpoints

GenericApp\_DstAddr.endPoint = pRsp->epList[0];

// Light LED

HalLedSet( HAL\_LED\_4, HAL\_LED\_MODE\_ON );

}

osal\_mem\_free( pRsp );

}

}

break;

}

}

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

\* @fn GenericApp\_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\_4

\* HAL\_KEY\_SW\_3

\* HAL\_KEY\_SW\_2

\* HAL\_KEY\_SW\_1

\*

\* @return none

\*/

void GenericApp\_HandleKeys( byte shift, byte keys )

{

zAddrType\_t dstAddr;

// Shift is used to make each button/switch dual purpose.

if ( shift )

{

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

{

}

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

{

}

if ( keys & HAL\_KEY\_SW\_3 )

{

}

if ( keys & HAL\_KEY\_SW\_4 )

{

}

}

else

{

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

{

}

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

{

HalLedSet ( HAL\_LED\_4, HAL\_LED\_MODE\_OFF );

// Initiate an End Device Bind Request for the mandatory endpoint

dstAddr.addrMode = Addr16Bit;

dstAddr.addr.shortAddr = 0x0000; // Coordinator

ZDP\_EndDeviceBindReq( &dstAddr, NLME\_GetShortAddr(),

GenericApp\_epDesc.endPoint,

GENERICAPP\_PROFID,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

FALSE );

}

if ( keys & HAL\_KEY\_SW\_3 )

{

}

if ( keys & HAL\_KEY\_SW\_4 )

{

HalLedSet ( HAL\_LED\_4, HAL\_LED\_MODE\_OFF );

// Initiate a Match Description Request (Service Discovery)

dstAddr.addrMode = AddrBroadcast;

dstAddr.addr.shortAddr = NWK\_BROADCAST\_SHORTADDR;

ZDP\_MatchDescReq( &dstAddr, NWK\_BROADCAST\_SHORTADDR,

GENERICAPP\_PROFID,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

FALSE );

}

}

}

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

\* LOCAL FUNCTIONS

\*/

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

\* @fn GenericApp\_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 GenericApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pkt )

{

//ss-6:取出协调器接收到的数据，写入串口

char buf[42];

unsigned char buffer[2] = {0x0A,0x0D};//回车换行符ASCII码

switch ( pkt->clusterId )

{

case GENERICAPP\_CLUSTERID:

osal\_memcpy(buf,pkt->cmd.Data,42);

HalUARTWrite(0,buf,42);

HalUARTWrite(0,buffer,2);

break;

}

}

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

\* @fn GenericApp\_SendTheMessage

\*

\* @brief Send "the" message.

\*

\* @param none

\*

\* @return none

\*/

void GenericApp\_SendTheMessage( void )

{

//ss-7:发送数据,广播

unsigned char \*theMessageData = "Coordinator send!";

afAddrType\_t my\_DstAddr;

my\_DstAddr.addrMode = (afAddrMode\_t)AddrBroadcast;

my\_DstAddr.endPoint = GENERICAPP\_ENDPOINT;

my\_DstAddr.addr.shortAddr = 0xFFFF;

AF\_DataRequest( &my\_DstAddr, &GenericApp\_epDesc,

GENERICAPP\_CLUSTERID,

osal\_strlen( theMessageData ) + 1,

theMessageData,

&GenericApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS );

}

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

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

### Enddevice：

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

Filename: GenericApp.c

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

Revision: $Revision: 19453 $

Description: Generic Application (no Profile).

Copyright 2004-2009 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 "Hello World" to another "Generic"

application every 15 seconds. The application will also

receive "Hello World" packets.

The "Hello World" messages are sent/received as MSG type message.

This applications doesn't have a profile, so it handles everything

directly - itself.

Key control:

SW1:

SW2: initiates end device binding

SW3:

SW4: initiates a match description request

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

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

\* INCLUDES

\*/

#include "OSAL.h"

#include "AF.h"

#include "ZDApp.h"

#include "ZDObject.h"

#include "ZDProfile.h"

#include "GenericApp.h"

#include "DebugTrace.h"

#if !defined( WIN32 )

#include "OnBoard.h"

#endif

/\* HAL \*/

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

#include "hal\_uart.h"

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

\* MACROS

\*/

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

\* CONSTANTS

\*/

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

\* TYPEDEFS

\*/

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

\* GLOBAL VARIABLES

\*/

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

const cId\_t GenericApp\_ClusterList[GENERICAPP\_MAX\_CLUSTERS] =

{

GENERICAPP\_CLUSTERID

};

const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc =

{

GENERICAPP\_ENDPOINT, // int Endpoint;

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

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

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

GENERICAPP\_FLAGS, // int AppFlags:4;

GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;

(cId\_t \*)GenericApp\_ClusterList, // byte \*pAppInClusterList;

GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;

(cId\_t \*)GenericApp\_ClusterList // byte \*pAppInClusterList;

};

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

// filled-in in GenericApp\_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 GenericApp\_epDesc;

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

\* EXTERNAL VARIABLES

\*/

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

\* EXTERNAL FUNCTIONS

\*/

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

\* LOCAL VARIABLES

\*/

byte GenericApp\_TaskID; // Task ID for internal task/event processing

// This variable will be received when

// GenericApp\_Init() is called.

devStates\_t GenericApp\_NwkState;

byte GenericApp\_TransID; // This is the unique message ID (counter)

afAddrType\_t GenericApp\_DstAddr;

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

\* LOCAL FUNCTIONS

\*/

void GenericApp\_ProcessZDOMsgs( zdoIncomingMsg\_t \*inMsg );

void GenericApp\_HandleKeys( byte shift, byte keys );

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

void GenericApp\_SendTheMessage( void );

void delay(unsigned int time);

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

\* NETWORK LAYER CALLBACKS

\*/

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

\* PUBLIC FUNCTIONS

\*/

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

\* @fn GenericApp\_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 GenericApp\_Init( byte task\_id )

{

GenericApp\_TaskID = task\_id;

GenericApp\_NwkState = DEV\_INIT;

GenericApp\_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().

GenericApp\_DstAddr.addrMode = (afAddrMode\_t)AddrNotPresent;

GenericApp\_DstAddr.endPoint = 0;

GenericApp\_DstAddr.addr.shortAddr = 0;

// Fill out the endpoint description.

GenericApp\_epDesc.endPoint = GENERICAPP\_ENDPOINT;

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

GenericApp\_epDesc.simpleDesc

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

GenericApp\_epDesc.latencyReq = noLatencyReqs;

// Register the endpoint description with the AF

afRegister( &GenericApp\_epDesc );

// Register for all key events - This app will handle all key events

RegisterForKeys( GenericApp\_TaskID );

// Update the display

#if defined ( LCD\_SUPPORTED )

HalLcdWriteString( "GenericApp", HAL\_LCD\_LINE\_1 );

#endif

ZDO\_RegisterForZDOMsg( GenericApp\_TaskID, End\_Device\_Bind\_rsp );

ZDO\_RegisterForZDOMsg( GenericApp\_TaskID, Match\_Desc\_rsp );

}

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

\* @fn GenericApp\_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 GenericApp\_ProcessEvent( byte task\_id, UINT16 events )

{

afIncomingMSGPacket\_t \*MSGpkt;

afDataConfirm\_t \*afDataConfirm;

// Data Confirmation message fields

byte sentEP;

ZStatus\_t sentStatus;

byte sentTransID; // This should match the value sent

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

if ( events & SYS\_EVENT\_MSG )

{

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

while ( MSGpkt )

{

switch ( MSGpkt->hdr.event )

{

case ZDO\_CB\_MSG:

GenericApp\_ProcessZDOMsgs( (zdoIncomingMsg\_t \*)MSGpkt );

break;

case KEY\_CHANGE:

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

break;

case AF\_DATA\_CONFIRM\_CMD:

// This message is received as a confirmation of a data packet sent.

// The status is of ZStatus\_t type [defined in ZComDef.h]

// The message fields are defined in AF.h

afDataConfirm = (afDataConfirm\_t \*)MSGpkt;

sentEP = afDataConfirm->endpoint;

sentStatus = afDataConfirm->hdr.status;

sentTransID = afDataConfirm->transID;

(void)sentEP;

(void)sentTransID;

// Action taken when confirmation is received.

if ( sentStatus != ZSuccess )

{

// The data wasn't delivered -- Do something

}

break;

case AF\_INCOMING\_MSG\_CMD:

GenericApp\_MessageMSGCB( MSGpkt );

break;

case ZDO\_STATE\_CHANGE:

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

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

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

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

{

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

osal\_start\_timerEx( GenericApp\_TaskID,

GENERICAPP\_SEND\_MSG\_EVT,

GENERICAPP\_SEND\_MSG\_TIMEOUT );

}

break;

default:

break;

}

// Release the memory

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

// Next

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

}

// return unprocessed events

return (events ^ SYS\_EVENT\_MSG);

}

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

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

if ( events & GENERICAPP\_SEND\_MSG\_EVT )

{

// Send "the" message

GenericApp\_SendTheMessage();

// Setup to send message again

osal\_start\_timerEx( GenericApp\_TaskID,

GENERICAPP\_SEND\_MSG\_EVT,

GENERICAPP\_SEND\_MSG\_TIMEOUT );

// return unprocessed events

return (events ^ GENERICAPP\_SEND\_MSG\_EVT);

}

// Discard unknown events

return 0;

}

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

\* Event Generation Functions

\*/

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

\* @fn GenericApp\_ProcessZDOMsgs()

\*

\* @brief Process response messages

\*

\* @param none

\*

\* @return none

\*/

void GenericApp\_ProcessZDOMsgs( zdoIncomingMsg\_t \*inMsg )

{

switch ( inMsg->clusterID )

{

case End\_Device\_Bind\_rsp:

if ( ZDO\_ParseBindRsp( inMsg ) == ZSuccess )

{

// Light LED

HalLedSet( HAL\_LED\_4, HAL\_LED\_MODE\_ON );

}

#if defined(BLINK\_LEDS)

else

{

// Flash LED to show failure

HalLedSet ( HAL\_LED\_4, HAL\_LED\_MODE\_FLASH );

}

#endif

break;

case Match\_Desc\_rsp:

{

ZDO\_ActiveEndpointRsp\_t \*pRsp = ZDO\_ParseEPListRsp( inMsg );

if ( pRsp )

{

if ( pRsp->status == ZSuccess && pRsp->cnt )

{

GenericApp\_DstAddr.addrMode = (afAddrMode\_t)Addr16Bit;

GenericApp\_DstAddr.addr.shortAddr = pRsp->nwkAddr;

// Take the first endpoint, Can be changed to search through endpoints

GenericApp\_DstAddr.endPoint = pRsp->epList[0];

// Light LED

HalLedSet( HAL\_LED\_4, HAL\_LED\_MODE\_ON );

}

osal\_mem\_free( pRsp );

}

}

break;

}

}

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

\* @fn GenericApp\_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\_4

\* HAL\_KEY\_SW\_3

\* HAL\_KEY\_SW\_2

\* HAL\_KEY\_SW\_1

\*

\* @return none

\*/

void GenericApp\_HandleKeys( byte shift, byte keys )

{

zAddrType\_t dstAddr;

// Shift is used to make each button/switch dual purpose.

if ( shift )

{

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

{

}

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

{

}

if ( keys & HAL\_KEY\_SW\_3 )

{

}

if ( keys & HAL\_KEY\_SW\_4 )

{

}

}

else

{

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

{

}

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

{

HalLedSet ( HAL\_LED\_4, HAL\_LED\_MODE\_OFF );

// Initiate an End Device Bind Request for the mandatory endpoint

dstAddr.addrMode = Addr16Bit;

dstAddr.addr.shortAddr = 0x0000; // Coordinator

ZDP\_EndDeviceBindReq( &dstAddr, NLME\_GetShortAddr(),

GenericApp\_epDesc.endPoint,

GENERICAPP\_PROFID,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

FALSE );

}

if ( keys & HAL\_KEY\_SW\_3 )

{

}

if ( keys & HAL\_KEY\_SW\_4 )

{

HalLedSet ( HAL\_LED\_4, HAL\_LED\_MODE\_OFF );

// Initiate a Match Description Request (Service Discovery)

dstAddr.addrMode = AddrBroadcast;

dstAddr.addr.shortAddr = NWK\_BROADCAST\_SHORTADDR;

ZDP\_MatchDescReq( &dstAddr, NWK\_BROADCAST\_SHORTADDR,

GENERICAPP\_PROFID,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

GENERICAPP\_MAX\_CLUSTERS, (cId\_t \*)GenericApp\_ClusterList,

FALSE );

}

}

}

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

\* LOCAL FUNCTIONS

\*/

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

\* @fn GenericApp\_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 GenericApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pkt )

{

//ss-1:对接收到的数据处理

char \*recvbuf;

switch ( pkt->clusterId )

{

case GENERICAPP\_CLUSTERID:

osal\_memcpy(recvbuf,pkt->cmd.Data,osal\_strlen("Coordinator send!")+1);

if(osal\_memcmp(recvbuf,"Coordinator send!",osal\_strlen("Coordinator send!")+1)){

GenericApp\_SendTheMessage();

}

else{

//这里可以添加相应的出错的代码

}

break;

}

}

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

\* @fn GenericApp\_SendTheMessage

\*

\* @brief Send "the" message.

\*

\* @param none

\*

\* @return none

\*/

void GenericApp\_SendTheMessage( void )

{

//ss-2:向协调器发送数据

unsigned char \*theMessageData = "| EndDevice--1 received! ";

afAddrType\_t my\_DstAddr;

my\_DstAddr.addrMode = (afAddrMode\_t)Addr16Bit;

my\_DstAddr.endPoint = GENERICAPP\_ENDPOINT;

my\_DstAddr.addr.shortAddr = 0x0000;

AF\_DataRequest( &my\_DstAddr, &GenericApp\_epDesc,

GENERICAPP\_CLUSTERID,

osal\_strlen( theMessageData ) + 1,

theMessageData,

&GenericApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS );

//亮灯

P1DIR = 0x0C;

P1\_2 = 0;

P1\_3 = 0;

delay(1000);

P1\_2 = 1;

P1\_3 = 1;

}

void delay(unsigned int time)

{

unsigned int i;

unsigned char j;

for(i = 0; i < time; i++)

{

for(j = 0; j < 240; j++)

{

asm("NOP"); //nop是内嵌汇编做空指令 延时用

asm("NOP"); //是一个空等待汇编指令，这个指令执行时，单片机是什么也不做，仅仅起一个时间延时作用。

asm("NOP"); //是指函数内部的参数翻译为汇编指令，其作用是在C语言环境下直接使用汇编指令执行

}

}

}

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

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

**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。

**截图：**



**总结：**

通过本次实验，对广播通信有了更深的认识，知道了协调器对终端进行广播通信的原理和实现方法。