1.Zigbee2006 串口互发

学习 Zigbee2006 发送、接收数据的方法，学习串口的使用，掌握 Zigbee 数据的处理方法。

1.1 实 验 目 的

- 通过 Zigbee 实现计算机与计算机之间的通讯
- 学习 Zigbee 的数据处理方法
- 学习各自发送数据的方式

1.2 实 验 内 容

通过串口发送数据，并通过串口接收数据，中间的介质采用 Zigbee2006.

1.3 实 验 设 备

- C51RF-3-PK/C51RF-3-JKS(三个 CC2430 模块、开发底板两个、电源、USB 线)
- IAR 集成开发环境
- C51RF-3 仿真器

1.4 串 口 互 发 例 程 的 实 现

1.4.1 串口通信的实现

串口采用异步通讯方式，参数设计为 384000、8、n、1，具体设计方法请参考基础实验 13~16，这几个实验概括的串口的基本用法，本实验中，串口的接收部分延用了协议栈内容设计的接收处理函数函数，（SPIMgr.c 中的 void SPIMgr_ProcessZToolData (uint8 port, uint8 event)函数），而发送函数在本实验中代码如下。

```
//cd wxl          串口 0 发数据
#include <ioCC2430.h>
#include <wxl_uart.h>
//#include <string.h>


/*****************************************************************
*函数功能 ：初始化串口 1
```

```
*入口参数 ： 无
*返 回 值 ： 无
*说      明 ： 57600-8-n-1
**********************************************************************/
void initUARTtest(void)
{

    CLKCON &= ~0x40;                    //晶振
    while(!(SLEEP & 0x40));         //等待晶振稳定
    CLKCON &= ~0x47;                    //TICHSPD128 分频，CLKSPD 不分频
    SLEEP |= 0x04;          //关闭不用的 RC 振荡器
    PERCFG = 0x00;                          //位置 1 P0 口
    P0SEL |= 0x0C;                      //P0 用作串口
    P2DIR &= ~0xC0;                                         //P0 优先作为串口 0
    U0CSR |= 0x80;                      //UART 方式
    UTX0IF = 0;

}
/**********************************************************************
*函数功能 ： 串口发送字符串函数
*入口参数 : data:数据
*                 len :数据长度
*返 回 值 ： 无
*说      明 ：
**********************************************************************/
void UartTX_Send_String(char *Data,int len)
{
  int j;
  for(j=0;j<len;j++)
  {
    U0DBUF = *Data++;
    while(UTX0IF == 0);
    UTX0IF = 0;
  }
}
void UartTX_Send_Single(char single_Data)
{
    U0DBUF = single_Data;
    while(UTX0IF == 0);
    UTX0IF = 0;
}
/*****************************************************************************
描述：
    串口接收一个字符
```

```
函数名：char UartRX_Receive_Char (void)
*****************************************************************************/
char UartRX_Receive_Char (void)
{
    char c;
    unsigned char status;
    status = U0CSR;
    U0CSR |= UART_ENABLE_RECEIVE;
    while (!URX0IF);
    c = U0DBUF;
    URX0IF = 0;
    U0CSR = status;
    return c;
}
/*****************************************************************************
描述：
    波特率的设置
函数名：void Uart_Baud_rate(int Baud_rate)
*****************************************************************************/
void Uart_Baud_rate(int Baud_rate)
{
  switch (Baud_rate)
  {
    case 24:
      U0GCR |= 6;
      U0BAUD |= 59;                    //波特率设置
    break;
    case 48:
      U0GCR |= 7;
      U0BAUD |= 59;                    //波特率设置
    break;
    case 96:
      U0GCR |= 8;
      U0BAUD |= 59;                    //波特率设置
    break;
    case 144:
      U0GCR |= 8;
      U0BAUD |= 216;                   //波特率设置
    break;
    case 192:
      U0GCR |= 9;
      U0BAUD |= 59;                    //波特率设置
    break;
    case 288:
```

```
            U0GCR |= 9;
            U0BAUD |= 216;                    //波特率设置
          break;
          case 384:
            U0GCR |= 10;
            U0BAUD |= 59;                     //波特率设置
          break;
          case 576:
            U0GCR |= 10;
            U0BAUD |= 216;                    //波特率设置
          break;
          case 768:
            U0GCR |= 11;
            U0BAUD |= 59;                     //波特率设置
          break;
          case 1152:
            U0GCR |= 11;
            U0BAUD |= 216;                    //波特率设置
          break;
          case 2304:
            U0GCR |= 12;
            U0BAUD |= 216;                    //波特率设置
          break;
          default:
          break;
        }
      }
```

1.4.2 发送函数

发送数据采用短地址的方式发送，在该函数中设计的传递参数有发送的数据、目的地址、数据长度。实现的代码如下：

```
//*************************************************************************
//**以短地址方式发送数据
//buf ::发送的数据
//addr::目的地址
//Leng::数据长度
//*************************************************************************
uint8 SendData(uint8 *buf, uint16 addr, uint8 Leng)
{
    afAddrType_t SendDataAddr;
    SendDataAddr.addrMode = (afAddrMode_t)Addr16Bit;          //短地址发送
    SendDataAddr.endPoint = SAMPLEAPP_ENDPOINT;
    SendDataAddr.addr.shortAddr = addr;
```

```
            if ( AF_DataRequest( &SendDataAddr, //发送的地址和模式
                                 &SampleApp_epDesc,     //终端（比如操作系统中任务 ID 等）
                                 2,//发送串 ID
                                 Leng,
                                 buf,
                                 &SampleApp_TransID,   //信息 ID（操作系统参数）
                                 AF_DISCV_ROUTE,
                                //   AF_ACK_REQUEST,
                                 AF_DEFAULT_RADIUS ) == afStatus_SUCCESS )
        {
            return 1;
        }
        else
        {
            return 0;// Error occurred in request to send.
        }
    }
```

1.4.3 接收处理函数

接收处理函数是将接收的数据进行处理，在实验中不同的设备收到的数据由不同的处理方式。

协调器收到数据后，首先判断是不是由新节点加入的网络，如果有的话，判断设备的类型，知道类型后，判断是不是已经加入了网络，如果是，就不再处理，如果不是，将该节点的物理地址和网络地址放到一个 buffer 里面。小灯闪烁两次。

如果不是新节点加入，将数据直接通过串口发送出去。

如果是路由器收到数据后，直接将数据通过串口发送。本实验中没有对终端节点处理。

```
void SampleApp_MessageMSGCB( afIncomingMSGPacket_t *pkt )
{
  int new_node_flag;
  new_node_flag = 0;
  if(clear == 0)
  {
    JoinNode.RouterCount = 0;                    //路由器计数清零
    JoinNode.RfdCount = 0;                       //终端计数清零
    clear = 1;
  }
  switch ( pkt->clusterId )
  {
    case SAMPLEAPP_PERIODIC_CLUSTERID:
      break;

    case SAMPLEAPP_FLASH_CLUSTERID:
```

```
        memcpy(RxBuf,pkt->cmd.Data,pkt->cmd.DataLength);
#if defined( ZDO_COORDINATOR )                  //如果是协调器收到数据
        if((RxBuf[0] == 'n') && (RxBuf[1] == 'e') &&(RxBuf[2] == 'w'))          //新节点加入
        {
          if((RxBuf[3] == 'R') && (RxBuf[4] == 'O') && (RxBuf[5] == 'U'))   //判断是路由器节点
          {
            for(int i=0;i<JoinNode.RouterCount;i++)
            {
              for(int j=0;j<8;j++)
         {
                //判断是否有相同地址
                if(JoinNode.RouterAddr[JoinNode.RouterCount][j] == RxBuf[j+6])
                {
                  new_node_flag++;              //判断位相同标志加 1
                }
                else
                {
                  new_node_flag = 0;            //判断位不同，表示地址不同，标志清 0
                  j += 8;
                }
              }
              if(new_node_flag == 8)
              {
                i += JoinNode.RouterCount;                    //退出查询
              }
            }
            if(new_node_flag == 0)
            {
              for(int i=0;i<8;i++)
              {
                JoinNode.RouterAddr[JoinNode.RouterCount][7-i] = RxBuf[i+6];      //存放物理地址
              }
              JoinNode.RouterAddr[JoinNode.RouterCount][8] = RxBuf[6+8];          //存放网络地址
              JoinNode.RouterAddr[JoinNode.RouterCount][9] = RxBuf[6+9];
              JoinNode.RouterCount ++;
            }
          }
          UartTX_Send_String( RxBuf,6);
           //通过串口发送数据
          UartTX_Send_String( JoinNode.RouterAddr[JoinNode.RouterCount-1],10);
        }
        else
        UartTX_Send_String(RxBuf,pkt->cmd.DataLength);
#elif defined( RTR_NWK ) && (!defined(ZDO_COORDINATOR))             //选择路由器
```

```
            UartTX_Send_String(RxBuf,pkt->cmd.DataLength);              //通过串口发送数据
      #else                                                 //剩下的就是终端节点
      #endif
            HalLedBlink( HAL_LED_4, 2, 50, 100 );                       //小灯闪烁
            break;
        }
      }
```

1.4.4 串口接收处理函数

串口接收到数据后，如果是协调器设备则判断有没有相同地址（在发送的数据的时候需要在数据前加上接收设备的物理地址），如果由则发送此数据（去除物理地址），如果不是则不处理。

如果是路由器则直接将数据发送到网管（在一个网络中，网络的网络地址为0x0000）。

```
      void SPIMgr_ProcessZToolData ( uint8 port, uint8 event )
      {
        int s;
        Uart_len = 0;
      #ifdef ZDO_COORDINATOR
        int k,f;
        int new_node_flag = 0;
      #endif

        /* Verify events */
        if (event == HAL_UART_TX_FULL)
        {
          // Do something when TX if full
          return;
        }

        if (event & (HAL_UART_RX_FULL | HAL_UART_RX_ABOUT_FULL |
HAL_UART_RX_TIMEOUT))
        {
          while (Hal_UART_RxBufLen(SPI_MGR_DEFAULT_PORT))
          {
            HalUARTRead (SPI_MGR_DEFAULT_PORT, &Uart_Rx_Data[Uart_len], 1);     //读取串口数据
          switch (state)
            {
              case SOP_STATE:
                if (Uart_Rx_Data[Uart_len] == SOP_VALUE)
                  state = CMD_STATE1;
                break;
              case CMD_STATE1:
```

```
                    CMD_Token[0] = Uart_Rx_Data[Uart_len];
                    state = CMD_STATE2;
                    break;
                case CMD_STATE2:
                    CMD_Token[1] = Uart_Rx_Data[Uart_len];
                    state = LEN_STATE;
                    break;
                case LEN_STATE:
                    LEN_Token = Uart_Rx_Data[Uart_len];
                    if (Uart_Rx_Data[Uart_len] == 0)
                        state = FCS_STATE;
                    else
                        state = DATA_STATE;
                    tempDataLen = 0;
                    // Allocate memory for the data
                    SPI_Msg = (mtOSALSerialData_t *)osal_msg_allocate( sizeof ( mtOSALSerialData_t ) +
2+1+LEN_Token );
                    if (SPI_Msg)
                    {
                        // Fill up what we can
                        SPI_Msg->hdr.event = CMD_SERIAL_MSG;
                        SPI_Msg->msg = (uint8*)(SPI_Msg+1);
                        SPI_Msg->msg[0] = CMD_Token[0];
                        SPI_Msg->msg[1] = CMD_Token[1];
                        SPI_Msg->msg[2] = LEN_Token;
                    }
                    else
                    {
                        state = SOP_STATE;
                        return;
                    }
                    break;
                case DATA_STATE:
                        SPI_Msg->msg[3 + tempDataLen++] = Uart_Rx_Data[Uart_len];
                        if ( tempDataLen == LEN_Token )
                            state = FCS_STATE;
                    break;
                case FCS_STATE:
                    FSC_Token = Uart_Rx_Data[Uart_len];
                    //Make sure it's correct
                    if ((SPIMgr_CalcFCS ((uint8*)&SPI_Msg->msg[0], 2 + 1 + LEN_Token) == FSC_Token))
                    {
                        osal_msg_send( MT_TaskID, (byte *)SPI_Msg );
                    }
```

```
                else
                {
                  // deallocate the msg
                  osal_msg_deallocate ( (uint8 *)SPI_Msg);
                }
                //Reset the state, send or discard the buffers at this point
                state = SOP_STATE;
                break;
              default:
               break;
            }
            Uart_len++;
          }
#ifdef ZDO_COORDINATOR
                for(k=0;k<JoinNode.RouterCount;k++)
                {
                  for( s=0;s<8;s++)
                  {
                    if(JoinNode.RouterAddr[k][s] == Uart_Rx_Data[s])          //判断是否有相同地址
                    {
                      new_node_flag++;          //判断位相同标志加 1
                    }
                    else
                    {
                      new_node_flag = 0;                    //判断位不同，表示地址不同，标志清 0
                      s += 8;
                    }
                  }
                    if(new_node_flag == 8)
                    {
                      f = k;
                      Short_Add = JoinNode.RouterAddr[k][9];                  //取短地址地位
                      k += JoinNode.RouterCount;
                      Short_Add <<= 8;                                          //退出查询
                    }
                }
                if(new_node_flag == 8)
                {
                  Short_Add |= JoinNode.RouterAddr[f][8];                     //取短地址高位
                  for(s=0;s<(Uart_len - 8);s++)
                  {
                    RfTx.TXDATA.DataBuf[s] = Uart_Rx_Data[8+s];
                     //取数据前 8 位是物理地址这里用 ASCII 表示
                  }
```

```
                SendData(RfTx.TXDATA.DataBuf,Short_Add,Uart_len-8);              //发送数据
            }
#elif defined( RTR_NWK ) && (!defined(ZDO_COORDINATOR))
                for(s=0;s<Uart_len;s++)
                {
                  RfTx.TXDATA.DataBuf[s] = Uart_Rx_Data[s];        //取串口接收的数据到发送 buf 中
                }
                SendData(RfTx.TXDATA.DataBuf,0x0000,Uart_len);     //将所有数据到网管
#else
#endif
  }
}
#endif //ZTOOL
```

1.6 代 码 实 现

略。（请参考具体源代码，内部有详细的注释，如有疑问请联系我公司技术支持，028-86786586-157/158）

本例程对无线串口通信进行了诠释，在稍加修改后，就可以实现点对点，点对多点，单播、广播等不同的发送方式，本手册只为一个使用向导，具体功能实现请参考源代码，在源代码中有纤细的注释。

1.7 实 验 步 骤

1. 连接好硬件，请参阅系统说明书。
2. 根据使用说明书的方法下载协调器代码，协调器代码运行后黄灯常亮，红灯闪烁 4 次后熄灭，表示网络已经建立成功，并通过串口发送”haha!Nework found succeed”。此时协调器处于等待节点加入的状态。
3. 用同样的方法下载路由器代码，路由器运行加入网络后，黄灯常亮，红灯闪烁两次后熄灭，路由器会通过串口发送“haha!Rou jiond succeed”，同时协调器在收到加入信息后红灯也闪烁两次，并将路由器的地址信息发送到 PC “newROU00000005 ”（”new”:表示新节点、“ROU”表示是路由器，“00000005”：表示设备的物理地址。在数据中的“ ”是非可见字符，他是网络地址在串口助手中用 16 进制的方式可以看到它的数据为 0x0001）。
4. 如果协调器向路由器发送数据，数据前必须要加入路由器设备的物理地址（这里是通过物理地址去寻到网络地址，再通过网络地址发送，这是因为在网络中协调器没有存储信息，如果设备调电，网络地址可能改变），以上面的新节点数据为例，如果发送的数据为“hello world！”哪么发送的数据为“00000005hello world！”。
5. 如果路由器向协调器发送数据，则直接发送即可，因为协调器的网络地址不会改变，为“0x0000”所以他们是一个透明传输过程。