Source Codes for Demo kit DAD04

V1.00

This document contains source codes for demo kit DAD04 of high power wireless voice transceiver module DRA808M and is used only for reference. In order to understand the codes better, users can refer to the application document ADW1003 for schematic of Demo kit DAD04.

```
#include<pic.h>
#include<math.h>

#define         ASCII_point         0x2e
#define         ASCII_comma         0x2c

#define ASCII(temp) \
            temp + 0x30;

void start_TX(void);
void stop_TX(void);
void start_RX(void);
void stop_RX(void);

//                                                  0     1     2     3     4     5     6     7     8
9     nop    "-"
const                                       unsigned                                        char
Tab_DispCode[12]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0xff,0xbf};

unsigned char EEPRead[13];

unsigned char tfv0;
unsigned char tfv1;
unsigned char tfv2;
unsigned char tfv3;
unsigned char rfv0;
unsigned char rfv1;
unsigned char rfv2;
unsigned char rfv3;
unsigned char ctcs_0;
unsigned char ctcs_1;
unsigned char sq;
unsigned char vol;

unsigned char pon_flag;

unsigned int    cnt_1s;
unsigned char rx_cnt;

const                                       unsigned                                        char
CMD_HAND[15]={0x41,0x54,0x2B,0x44,0x4D,0x4F,0x43,0x4F,0x4E,0x4E,0x45,0x43,0x54,0x0d,0x0a};
unsigned                                                                                    char
```

```
CMD_SET[15]={0x41,0x54,0x2b,0x44,0x4d,0x4f,0x53,0x45,0x54,0x47,0x52,0x4f,0x55,0x50,0x3d};
unsigned char CMD_VOLUME[16]={0x41,0x54,0x2B,0x44,0x4D,0x4F,0x53,0x45,0x54,0x56,0x4f,0x4c,0x55,0x4d,0x45
,0x3d};

unsigned char tx_buf[50]={0};
unsigned char rx_buf[30]={0};
unsigned char tx_len;
unsigned char rx_len;
unsigned char len_txnow;
unsigned char len_rxnow;
unsigned char status_cnt = 3;


typedef struct
{
    unsigned char reach_1s      : 1;
    unsigned char gbw           : 1;
    unsigned char in_rx         : 1;
    unsigned char in_tx         : 1;
    unsigned char cn_fail       : 1;
    unsigned char reset         : 1;
    unsigned char poweron       : 1;
    unsigned char poweron1      : 1;
    unsigned char pon_vol       : 1;
}   FlagType;

FlagType    Flag;

void para_init();
void uart_init(void);
void uart_trans_check(void);
void uart_recv_ack(void);
void check_uart(void);
void send_hand(void);
void send_set(void);
void send_vol(void);
void ASCII_TFV(void);
void ASCII_RFV(void);
void clr_tx_buf(void);
void clr_rx_buf(void);

void delay_xms(unsigned int xms);
```

```
void delay_1ms(void);
void port_init(void);
void timer_init(void);

/*********************************************************************
*********************************************************************
*********************************************************************/

void  main()
{
      OSCCON = 0x78;            // 14.7456M crystal

      WDTCON = 0x00;

      delay_xms(1000);     // Delay after powering-on

      port_init();

      timer_init();

      uart_init();

      INTCON = 0xc0;            // Enable interrupt

      Flag.cn_fail = 1;
      Flag.poweron = 1;
      pon_flag = 1;
      Flag.pon_vol = 0;
      Flag.poweron1 = 0;
      Flag.reset = 0;

      para_init();

      while(1)
      {
            uart_trans_check();
            uart_recv_ack();

            if((Flag.reset)&&(!Flag.in_tx)&&(!Flag.in_rx))
            {
                  Flag.reset = 0;
                  Flag.poweron1 = 1;
                  send_set();             // set parameter commands after handshaking command
```

```
        }

        if((Flag.pon_vol)&&(!Flag.in_tx)&&(!Flag.in_rx))
        {
            Flag.pon_vol = 0;
            send_vol();                 // Send volume setting command
        }

        if(Flag.reach_1s == 1)
        {
            Flag.reach_1s = 0;
            if(pon_flag)
            {
                pon_flag = 0;
                check_uart();
            }
        }
    }
}


/*************************************************************
************************* Display *************************
*************************************************************/

void interrupt ISR_timer(void)
{
    unsigned char int_temp;

    if(TXIF)                                // Renew transmit data
    {
        if(Flag.in_tx == 0)
            stop_TX();

        else if(len_txnow <= tx_len)
        {
            TXREG = tx_buf[len_txnow];
            len_txnow ++;
        }

        else
            TXIE = 0;
    }
```

```
    if(RCIF)                                    // Receive data
    {
        NOP();
        if(Flag.in_rx)
        {
            rx_buf[len_rxnow] = RCREG;
            if((len_rxnow++) == (rx_len+1))
                stop_RX();
        }
        else
        {
            stop_RX();
            int_temp = RCREG;               // Ineffective data, clear-up
        }
    }

    if(TMR2IF)                                      // 300 hz
    {
        cnt_1s++;

        if(cnt_1s == 300)
        {
            cnt_1s = 0;
            Flag.reach_1s = 1;
        }

    TMR2IF=0;
    }

}

//************************************************************
//********************* UART FUNCTION ***********************
//************************************************************

void uart_init()        // Initialize serial port，9600,8bit,1stop,no check
{
    SPBRGH = 0;
    SPBRG = 23;
    TXSTA = 0;
    RCSTA = 0x90;
    BAUDCON = 0;
```

```
    TXIE = 0;
    RCIE = 0;
}
//================================================

void uart_trans_check(void)                    // Convert to receive when no transmit
{
    if((Flag.in_tx == 1)&&(len_txnow > tx_len)&&(TXIF == 1))
    {
        stop_TX();

        Flag.in_tx = 0;
        Flag.in_rx = 1;
        rx_cnt = 2;
        len_txnow = 0;
        len_rxnow = 0;
        tx_len = 0;
        clr_tx_buf();

        start_RX();
    }
}
//================================================
void uart_recv_ack(void)
{

    if(Flag.in_rx == 0)
        return;

    if(len_rxnow == rx_len)
    {
        Flag.in_rx = 0;

        if((rx_buf[rx_len-3] == 0x30)&&(rx_buf[rx_len-2] == 0x0d)&&(rx_buf[rx_len-1] == 0x0a))
        {
            status_cnt = 3;
            Flag.cn_fail = 0;
            stop_RX();

            if((rx_len == 15)&&(Flag.poweron))   // Receive handshaking response
            {
                Flag.reset = 1;
                Flag.poweron = 0;
```

```
                }


                if((rx_len == 16)&&(Flag.poweron1))         //Receive setting response
                {
                        Flag.pon_vol = 1;
                        Flag.poweron1 = 0;
                }

                return;
        }
        else if(Flag.cn_fail == 1)
                return;
        else
        {
                status_cnt -= 1;
                if(status_cnt == 0)
                {
                        Flag.cn_fail = 1;
                        Flag.poweron = 1;           // Hand-shaking failed，resend
                        pon_flag = 1;
                }
        }
    }
}
//=================================================
void check_uart()
{
    unsigned char i;

    if(Flag.in_rx == 1)
    {
        rx_cnt --;
        if(rx_cnt == 0)
        {
                Flag.in_rx = 0;
                Flag.in_tx = 0;
                Flag.cn_fail = 1;
                pon_flag = 1;
                Flag.poweron = 1;
                return;
        }
    }
```

```
    if((Flag.in_tx == 1)||(Flag.in_rx == 1))
        return;

    send_hand();
}
//=================================================
void send_hand()                    // Send handshaking command
{
    unsigned char i;

    for(i=0;i<=14;i++)
        tx_buf[i] = CMD_HAND[i];

    rx_len = 15;
    tx_len = 15;
    len_txnow = 0;
    Flag.in_tx = 1;
    clr_rx_buf();

    start_TX();
}
//-------------------------------------------------
void send_set()                     // Send setting command
{
    unsigned char i;

    for(i=0;i<=14;i++)
        tx_buf[i] = CMD_SET[i];

    tx_buf[15] = ASCII(Flag.gbw);
    tx_buf[16] = ASCII_comma;
    ASCII_TFV();
    tx_buf[25] = ASCII_comma;
    ASCII_RFV();
    tx_buf[34] = ASCII_comma;
    tx_buf[35] = ASCII(ctcs_1);
    tx_buf[36] = ASCII(ctcs_0);
    tx_buf[37] = ASCII_comma;
    tx_buf[38] = ASCII(sq);
    tx_buf[39] = 0x0d;
    tx_buf[40] = 0x0a;
```

```
        rx_len = 16;
        tx_len = 41;
        len_txnow = 0;
        Flag.in_tx = 1;
        clr_rx_buf();

        start_TX();
}
//-------------------------------------------------
void send_vol()                         // Setting volume command
{
        unsigned char i;

        for(i=0;i<=15;i++)
                tx_buf[i] = CMD_VOLUME[i];

        tx_buf[16] = ASCII(vol);
        tx_buf[17] = 0x0d;
        tx_buf[18] = 0x0a;

        rx_len = 17;
        tx_len = 19;
        len_txnow = 0;
        Flag.in_tx = 1;
        clr_rx_buf();

        start_TX();
}
//-------------------------------------------------
void ASCII_TFV()                        // ASCII code convert
{
        tx_buf[17] = 0x34;
        tx_buf[18] = ASCII(tfv3);
        tx_buf[19] = ASCII(tfv2);
        tx_buf[20] = ASCII_point;
        tx_buf[21] = ASCII(tfv1);

        switch(tfv0)
        {
                case 0:
                        tx_buf[22] = 0x30;
                        tx_buf[23] = 0x30;
                        tx_buf[24] = 0x30;
```

```
            break;
            case 1:
                tx_buf[22] = 0x31;
                tx_buf[23] = 0x32;
                tx_buf[24] = 0x35;
            break;
            case 2:
                tx_buf[22] = 0x32;
                tx_buf[23] = 0x35;
                tx_buf[24] = 0x30;
            break;
            case 3:
                tx_buf[22] = 0x33;
                tx_buf[23] = 0x37;
                tx_buf[24] = 0x35;
            break;
            case 4:
                tx_buf[22] = 0x35;
                tx_buf[23] = 0x30;
                tx_buf[24] = 0x30;
            break;
            case 5:
                tx_buf[22] = 0x36;
                tx_buf[23] = 0x32;
                tx_buf[24] = 0x35;
            break;
            case 6:
                tx_buf[22] = 0x37;
                tx_buf[23] = 0x35;
                tx_buf[24] = 0x30;
            break;
            case 7:
                tx_buf[22] = 0x38;
                tx_buf[23] = 0x37;
                tx_buf[24] = 0x35;
            break;
        }
}
//-----------------------------------------------
void ASCII_RFV()                    // ASCII code convert
{
    tx_buf[26] = 0x34;
    tx_buf[27] = ASCII(rfv3);
```

```
tx_buf[28] = ASCII(rfv2);
tx_buf[29] = ASCII_point;
tx_buf[30] = ASCII(rfv1);

switch(rfv0)
{
    case 0:
        tx_buf[31] = 0x30;
        tx_buf[32] = 0x30;
        tx_buf[33] = 0x30;
    break;
    case 1:
        tx_buf[31] = 0x31;
        tx_buf[32] = 0x32;
        tx_buf[33] = 0x35;
    break;
    case 2:
        tx_buf[31] = 0x32;
        tx_buf[32] = 0x35;
        tx_buf[33] = 0x30;
    break;
    case 3:
        tx_buf[31] = 0x33;
        tx_buf[32] = 0x37;
        tx_buf[33] = 0x35;
    break;
    case 4:
        tx_buf[31] = 0x35;
        tx_buf[32] = 0x30;
        tx_buf[33] = 0x30;
    break;
    case 5:
        tx_buf[31] = 0x36;
        tx_buf[32] = 0x32;
        tx_buf[33] = 0x35;
    break;
    case 6:
        tx_buf[31] = 0x37;
        tx_buf[32] = 0x35;
        tx_buf[33] = 0x30;
    break;
    case 7:
        tx_buf[31] = 0x38;
```

```
                tx_buf[32] = 0x37;
                tx_buf[33] = 0x35;
            break;
        }
}
//-----------------------------------------------
void clr_tx_buf()                   // Clear transmit buffer
{
    unsigned char i;

    for(i=0;i<=39;i++)
        tx_buf[i]=0;
}
//-----------------------------------------------
void clr_rx_buf()                   // Clear receive buffer
{
    unsigned char i;

    for(i=0;i<=18;i++)
        rx_buf[i] = 0;
}
//-----------------------------------------------
void start_TX()                     // Enable transmit and terminal
{
    TXEN = 1;
    TXIE = 1;
}

void stop_TX()                      // Disable transmit and terminal
{
    TXEN = 0;
    TXIE = 0;
}

void start_RX()                     // Enable receive and terminal
{
    CREN = 1;
    RCIE = 1;
}

void stop_RX()                      // Disable receive and terminal
{
    CREN = 0;
```

```
    RCIE = 0;
}
//-----------------------------------------------
//************************************************************
//********************** SUBROUTINE **********************
//************************************************************

void port_init(void)                    // IO port initializing
{
    ANSELA = 0;
    ANSELB = 0;
    ANSELD = 0;
    TRISA = 0b00000111;
    TRISB = 0;
    WPUB = 0;
    TRISC = 0b10000000;
    TRISD = 0b00000000;
    TRISE = 0;

    PORTA = 0b00000111;
    PORTB = 0b11101111;
    PORTC = 0b11000000;
    PORTD = 0;
    PORTE = 0;
}
//-----------------------------------------------
void timer_init(void)                   // Timer initializing
{                                       //(14745600/4/64/16)*12=300hz
    T2CON = 0x7f;
    PR2 = 12;
    TMR2IE = 1;
    NOP();
}
//-----------------------------------------------
void delay_1ms(void)
{
    int i;
    for(i = 0; i<61; i++)
    {
        ;
    }
}
//-----------------------------------------------
```

```
void delay_xms( unsigned int xms )
{
	int	i;
	for(i = 0; i<xms; i++)
	{
		delay_1ms();
	}
}
//-------------------------------------------
void para_init()			// Parameter initializing
{
		Flag.gbw = 0;
		tfv0 = 4;
		tfv1 = 7;
		tfv2 = 9;
		tfv3 = 0;
		rfv0 = 4;
		rfv1 = 7;
		rfv2 = 9;
		rfv3 = 0;
		ctcs_0 = 0;
		ctcs_1 = 0;
		sq = 1;
		vol = 4;
}
```

Dorji Applied Technologies

A division of ***Dorji Industrial Group Co., Ltd***

Add.: Xinchenhuayuan 2, Dalangnanlu, Longhua, Baoan district, Shenzhen, China 518109

Tel: 0086-755-28156122

Fax.: 0086-755-28156133

Email: sales@dorji.com

Web: http://www.dorji.com

Dorji Industrial Group Co., Ltd reserves the right to make corrections, modifications, improvements and other changes to its products and services at any time and to discontinue any product or service without notice. Customers are expected to visit websites for getting newest product information before placing orders.

These products are not designed for use in life support appliances, devices or other products where malfunction of these products might result in personal injury. Customers using these products in such applications do so at their own risk and agree to fully indemnify Dorji Industrial Group for any damages resulting from improper use.