

Gesture control PPT based on ble

1 Introduction

Two development boards transmit control information through ble. One development board connects to paj7620 and provides gesture information through IIC bus. The other development board uses ble and USB HID. Ble is used to receive data, and USB HID is used to simulate keyboard input and control ppt

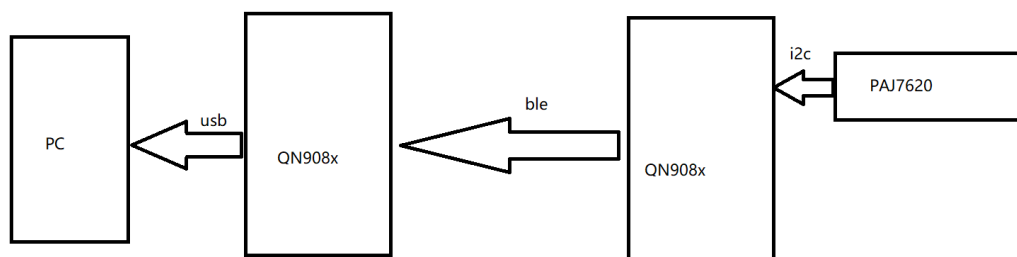


Figure 1

2 Preparation

We need two development boards qn908x and gesture control device paj7620. We use IAR as development environment. The example we use is temperature_sensor, and temperature_collector. The SDK version is 2.2.3

3 Code

3.1 temperature_sensor code

We want to implement IIC to read gesture information from paj7620 and send data.

The pins used by IIC are PA6 and PA7

Simply encapsulate the IIC reading and writing code in the code to create i2c_operation.c and i2c_operation.h. Realize IIC initialization and reading / writing register function in it

```

bool LPI2C_ReadRegs(I2C_Type *base, uint8_t device_addr,
                   uint8_t reg_addr, uint8_t *rxBuff,
                   uint32_t rxSize)
{
    i2c_master_transfer_t masterXfer;
    status_t reVal = kStatus_Fail;

    memset(&masterXfer, 0, sizeof(masterXfer));
    masterXfer.slaveAddress = device_addr;
    masterXfer.direction = kI2C_Read;
    masterXfer.subaddress = reg_addr;
    masterXfer.subaddressSize = 1;
    masterXfer.data = rxBuff;
    masterXfer.dataSize = rxSize;
    masterXfer.flags = kI2C_TransferDefaultFlag;

    /* direction=write : start+device_write;cmdbuff;xBuff; */
    /* direction=receive : start+device_write;cmdbuff;repeatStart+device_read;xBuff; */

    reVal = I2C_MasterTransferNonBlocking(BOARD_ACCEL_I2C_BASEADDR, &g_m_handle, &masterXfer);
    if (reVal != kStatus_Success)
    {
        return false;
    }
    /* wait for transfer completed. */
    while ((!nakFlag) && (!completionFlag))
    {
    }

    nakFlag = false;

    if (completionFlag == true)
    {
        completionFlag = false;
        return true;
    }
    else
    {
        return false;
    }
}

```

Figure 2

```

bool LPI2C_WriteReg(I2C_Type *base, uint8_t device_addr, uint8_t reg_addr, uint8_t value)
{
    i2c_master_transfer_t masterXfer;
    status_t reVal = kStatus_Fail;

    memset(&masterXfer, 0, sizeof(masterXfer));

    masterXfer.slaveAddress = device_addr;
    masterXfer.direction = kI2C_Write;
    masterXfer.subaddress = reg_addr;
    masterXfer.subaddressSize = 1;
    masterXfer.data = &value;
    masterXfer.dataSize = 1;
    masterXfer.flags = kI2C_TransferDefaultFlag;

    /* direction=write : start+device_write;cmdbuff;xBuff; */
    /* direction=receive : start+device_write;cmdbuff;repeatStart+device_read;xBuff; */

    reVal = I2C_MasterTransferNonBlocking(BOARD_ACCEL_I2C_BASEADDR, &g_m_handle, &masterXfer);
    if (reVal != kStatus_Success)
    {
        return false;
    }

    /* wait for transfer completed. */
    while ((!nakFlag) && (!completionFlag))
    {
    }

    nakFlag = false;

    if (completionFlag == true)
    {
        completionFlag = false;
        return true;
    }
    else
    {
        return false;
    }
}

```

Figure 3

3.1.1 After having these functions, we begin to write gesture recognition code. First, we add two blank files paj7620.c and paj7620.h into our project.

Select bank register area

```
//PAJ7620U2 selectBank
void paj7620u2_selectBank(bank_e bank)
{
    switch(bank)
    {
        case BANK0: LPI2C_WriteReg(I2C0, PAJ7620_ID, PAJ_REGITER_BANK_SEL, PAJ_BANK0);break;//BANK0 area
        case BANK1: LPI2C_WriteReg(I2C0, PAJ7620_ID, PAJ_REGITER_BANK_SEL, PAJ_BANK1);break;//BANK1 area
    }
}
```

Figure 4

Wake up paj7620 to read device state

```
//PAJ7620U2 wakeup
uint8_t paj7620u2_wakeup(void)
{
    uint8_t data = 0;
    paj7620u2_selectBank(PAJ_BANK0);
    LPI2C_ReadRegs(I2C0, PAJ7620_ID, 0, &data, 1);
    if(data!=0x20)
        return 0;
    return 1;
}
```

Figure 5

Initialize device

```
//PAJ7620U2 init
//return: 0:fail 1:succcess
uint8_t paj7620u2_init(void)
{
    uint8_t status;
    uint32_t i;
    status = paj7620u2_wakeup();//wakeup PAJ7620U2
    if(!status)
        return 0;
    paj7620u2_selectBank(BANK0);//enter BANK0 area
    for(i=0;i<INIT_SIZE;i++)
    {
        LPI2C_WriteReg(I2C0, PAJ7620_ID, init_Array[i][0], init_Array[i][1]);
    }
    paj7620u2_selectBank(BANK0);//
    return 1;
}
```

Figure 6

Gesture test function

```
uint16_t Gesture_test(void)
{
    uint8_t status;
    uint8_t i;
    uint8_t index;
    uint8_t data[2]={0x00};
    uint16_t gesture_data;
    if(paj_init == false)
    {
        paj7620u2_selectBank(BANK0); //enter BANK0 area
        for(i=0; i<GESTURE_SIZE; i++)
        {
            status = LPI2C_WriteReg(I2C0, PAJ7620_ID, gesture_array[i][0], gesture_array[i][1]);
            if(status == true)
            {
                index++;
            }
        }
        if(index == GESTURE_SIZE)
        {
            // AppPrintString("write ok\r\n");
        }
        paj7620u2_selectBank(BANK0); //
        paj_init = true;
    }

    status = LPI2C_ReadRegs(I2C0, PAJ7620_ID, PAJ_GET_INT_FLAG1, data, 2);
    if(status)
    {
        gesture_data =(uint16_t)data[1]<<8 | data[0];
        return gesture_data;
    }
    return 0;
}
```

Figure 7

3.1.2 When you are ready to read the device information,

You should initialize IIC and paj7620 in BleApp_Init function

```
*****
void BleApp_Init(void)
{
    /* Initialize application support for drivers */
    BOARD_InitAdc();
    I2C_init(BOARD_ACCEL_I2C_BASEADDR);
    if(paj7620u2_init())
    {
        AppPrintString("PAJ7620 init success.\r\n");
    }
}
```

Figure 8

In principle, we need to create a custom service for the PAJ device, but we replace the temperature data as our gesture control data. If you want to create a custom service, refer to this link [custom profile](#)

3.1.3 Create a timer that sends gesture data regularly.

In file temerature_sensor.c

Define a timer, static tmrTimerID_t dataTimerId;

Allocate a timer, dataTimerId = TMR_AllocateTimer();

Define the callback function of this timer

```
static void dataTimerCallback(void* pParam)
{
    uint16_t gesture_data = Gesture_test();
    if(gesture_data)
    {
        (void)Tms_RecordTemperatureMeasurement((uint16_t)service_temperature, gesture_data*100);
    }
}
```

Figure 9

Start timer

```
static void BleApp_ConnectionCallback (deviceId_t peerDeviceId, gapConnectionEvent_t* pConnectionEvent)
{
    /* Connection Manager to handle Host Stack interactions */
    BleConnManager_GapPeripheralEvent(peerDeviceId, pConnectionEvent);

    switch (pConnectionEvent->eventType)
    {
        case gConnEvtConnected_c:
        {
            /* Advertising stops when connected */
            mAdvState.advOn = FALSE;
            (void)TMR_StopTimer(appTimerId);

            /* Subscribe client*/
            mPeerDeviceId = peerDeviceId;
            (void)Bas_Subscribe(&basServiceConfig, peerDeviceId);
            (void)Tms_Subscribe(peerDeviceId);

            AppPrintString("Connected!\r\n");
            (void)TMR_StartLowPowerTimer(appTimerId,
                gTmrLowPowerSecondTimer_c,
                TmrSeconds(gGoToSleepAfterDataTime_c),
                DisconnectTimerCallback, NULL);
            TMR_StartLowPowerTimer(dataTimerId, gTmrLowPowerIntervalMillisTimer_c,
                TmrMilliseconds(1),
                dataTimerCallback, NULL);

            /* Set low power mode */
        }
    }
}
```

Figure 10

Close the low power mode. #define cPWR_UsePowerDownMode 0

3.2 temperature_collector code

The most important thing here is to port USB HID into our project. The USB example we use is the USB keyboard and mouse.

3.2.1 Add the OSA and USB folder under the example to the project directory, and copy the file to the corresponding folder according to the file structure of the original example.

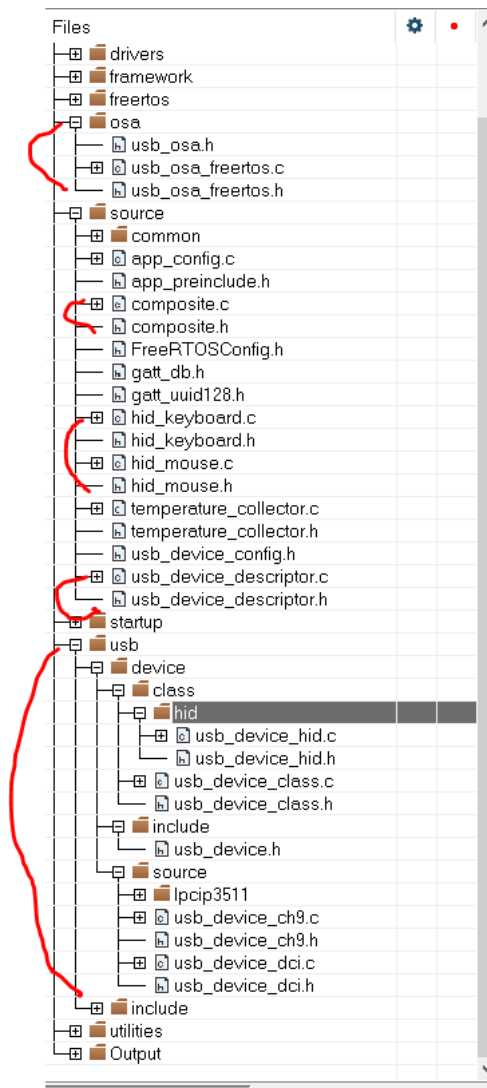


Figure 11

3.2.2 Add header file directory after completion

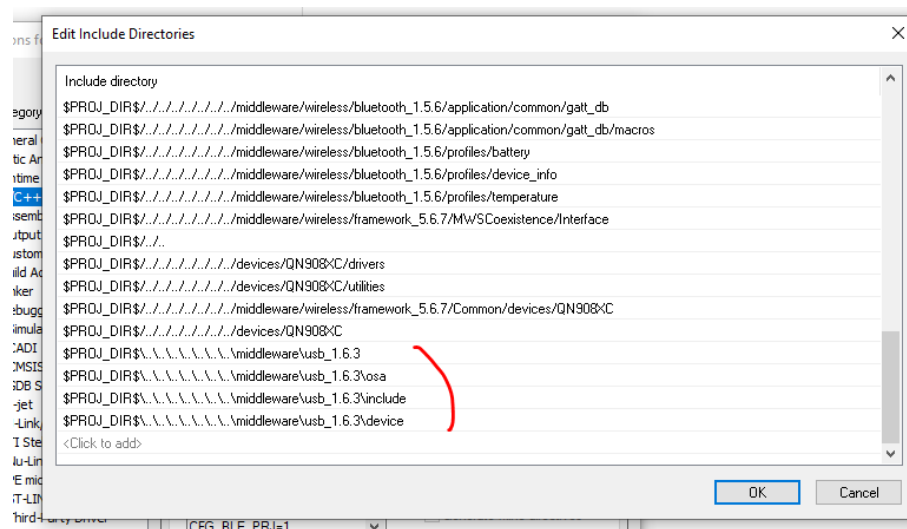


Figure 12

At the same time, in this tab, add two macro definitions

USB_STACK_FREERTOS_HEAP_SIZE=16384

USB_STACK_FREERTOS

3.2.3 Next, we need to modify the main function in usb example . Open composite.c file.

```
#if defined(__CC_ARM) || defined(__GNUC__)
int usb_main(void)
#else
void usb_main(void)
#endif
{
    // BOARD_InitPins();
    // BOARD_BootClockRUN();
    // BOARD_InitDebugConsole();
    POWER_DisablePD(kPDRUNCFG_PD_USBPLL); /* Turn on USB PLL */

    /* Disable USB PHY standby */
    SYSCON->USB_CFG = ((1 << SYSCON_USB_CFG_DPPUEN_B_PHY_POL_SHIFT) | (1 << SYSCON_USB_CFG_DPPUEN_B_PHY_SEL_SHIFT) |
        (1 << SYSCON_USB_CFG_USB_VBUS_SHIFT) | (0 << SYSCON_USB_CFG_USB_PHYSTDBY_SHIFT) |
        (0 << SYSCON_USB_CFG_USB_PHYSTDBY_WEN_SHIFT));

    SYSCON->MISC &= ~(1 << SYSCON_PIO_CFG_MISC_TRX_EN_INV_SHIFT);

    /* USB 48M clock calib */
    CLOCK_EnableClock(kCLOCK_Cal);
    CALIB_CalibPLL48M();

    APP_task(NULL);

    // if (xTaskCreate(APP_task, /* pointer to the task */
    //                 "app task", /* task name for kernel awareness debugging */
    //                 5000L / sizeof(portSTACK_TYPE), /* task stack size */
    //                 &g_UsbDeviceComposite, /* optional task startup argument */
    //                 4U, /* initial priority */
    //                 &g_UsbDeviceComposite.applicationTaskHandle /* optional task handle to create */
    //                 ) != pdPASS)
    // {
    //     usb_echo("app task create failed!\r\n");
    // }
    // #if (defined(__CC_ARM) || defined(__GNUC__))
    //     return 1U;
    // #else
    //     return;
    // #endif
    // }
    // vTaskStartScheduler();
}
```

Figure 13

It calls the APP_task. So this function also need to be modified.

```
void APP_task(void *handle)
{
    USB_DeviceApplicationInit();
    // #if USB_DEVICE_CONFIG_USE_TASK
    //     if (g_UsbDeviceComposite.deviceHandle)
    //     {
    //         if (xTaskCreate(USB_DeviceTask, /* pointer to the task */
    //                         "usb device task", /* task name for kernel awareness debugging */
    //                         5000L / sizeof(portSTACK_TYPE), /* task stack size */
    //                         g_UsbDeviceComposite.deviceHandle, /* optional task startup argument */
    //                         5U, /* initial priority */
    //                         &g_UsbDeviceComposite.deviceTaskHandle /* optional task handle to create */
    //                         ) != pdPASS)
    //         {
    //             usb_echo("usb device task create failed!\r\n");
    //             return;
    //         }
    //     }
    // #endif

    // while (1U)
    // {
    // }
}
```

Figure 14

3.2.4 Find hid_mouse.c, Comment function USB_DeviceHidMouseAction

Find hid_keyboard.h. Define the gesture information.

```
/*gesture*/
#define BIT(x) 1<<(x)
#define GES_UP BIT(0) //向上
#define GES_DOWM BIT(1) //向下
#define GES_LEFT BIT(2) //向左
#define GES_RIGHT BIT(3) //向右
#define GES_FORWARD BIT(4) //向前
#define GES_BACKWARD BIT(5) //向后
#define GES_CLOCKWISE BIT(6) //顺时针
#define GES_COUNT_CLOCKWISE BIT(7) //逆时针
#define GES_WAVE BIT(8) //挥动
```

Figure 15

Find hid_keyboard.c. We need to modify the function USB_DeviceHidKeyboardAction as following figure.

```
static usb_status_t USB_DeviceHidKeyboardAction(void)
{
    if(gesture_from_server)
    {
        USB_DeviceGesture(gesture_from_server);
        gesture_from_server = 0;
    }
    else
    {
        s_UsbDeviceHidKeyboard.buffer[2] = 0;
        return USB_DeviceHidSend(s_UsbDeviceComposite->hidKeyboardHandle, USB_HID_KEYBOARD_ENDPOINT_IN,
                                s_UsbDeviceHidKeyboard.buffer, USB_HID_KEYBOARD_REPORT_LENGTH);
    }
    return 0;
}
```

Figure 16

Among them, we also need to implement the following function. When the up hand gesture is detected, the previous ppt will be played. The down hand gesture will be the next PPT, the left hand gesture will exit PPT, and the forward hand gesture will play ppt

```
extern uint16_t gesture_from_server;
usb_status_t USB_DeviceGesture(uint16_t gesture)
{
    if(gesture)
    {
        switch(gesture)
        {
            case GES_FORWARD: s_UsbDeviceHidKeyboard.buffer[2] = KEY_F5;break;
            case GES_UP: s_UsbDeviceHidKeyboard.buffer[2] = KEY_LEFTARROW;break;
            case GES_DOWM:s_UsbDeviceHidKeyboard.buffer[2] = KEY_RIGHTARROW;break;
            case GES_LEFT:s_UsbDeviceHidKeyboard.buffer[2] = KEY_ESCAPE;break;
            default:break;
        }
        return USB_DeviceHidSend(s_UsbDeviceComposite->hidKeyboardHandle, USB_HID_KEYBOARD_ENDPOINT_IN,
                                s_UsbDeviceHidKeyboard.buffer, USB_HID_KEYBOARD_REPORT_LENGTH);
    }
    return 0;
}
```

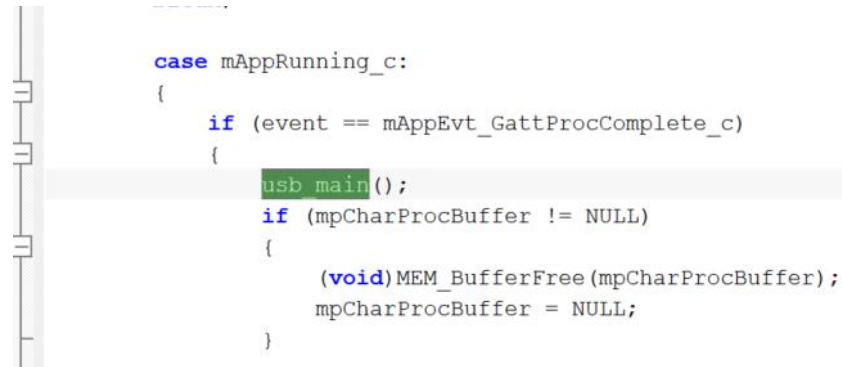
Figure 17

It also refers to an external variable gesture_from_server. The variable definition is in file temperature_collocation.c.,

3.2.5 After that, let's go to BleApp_StateMachinehandler function

in temperature_collector.c.

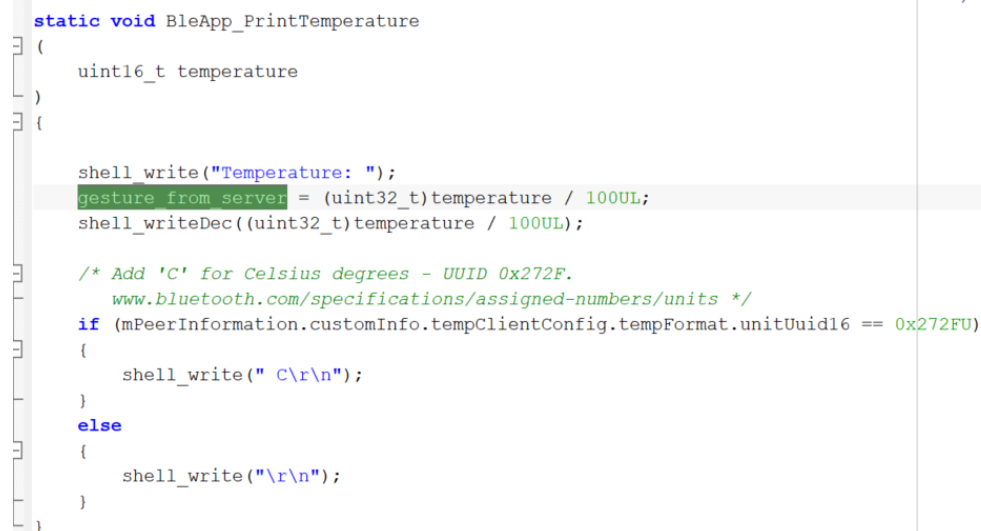
In case mAppRunning_c, we will call usb_main to initialize USB HID



```
case mAppRunning_c:
{
    if (event == mAppEvt_GattProcComplete_c)
    {
        usb_main();
        if (mpCharProcBuffer != NULL)
        {
            (void)MEM_BufferFree(mpCharProcBuffer);
            mpCharProcBuffer = NULL;
        }
    }
}
```

Figure 18

3.2.6 In BleApp_PrintTemperature, we will save the gesture data to gesture_from_server



```
static void BleApp_PrintTemperature
{
    uint16_t temperature

}

{
    shell_write("Temperature: ");
    gesture_from_server = (uint32_t)temperature / 100UL;
    shell_writeDec((uint32_t)temperature / 100UL);

    /* Add 'C' for Celsius degrees - UUID 0x272F.
       www.bluetooth.com/specifications/assigned-numbers/units */
    if (mPeerInformation.customInfo.tempClientConfig.tempFormat.unitUuid16 == 0x272FU)
    {
        shell_write(" C\r\n");
    }
    else
    {
        shell_write("\r\n");
    }
}
```

Figure 19

We finished the all steps.