5.3 水平测试板

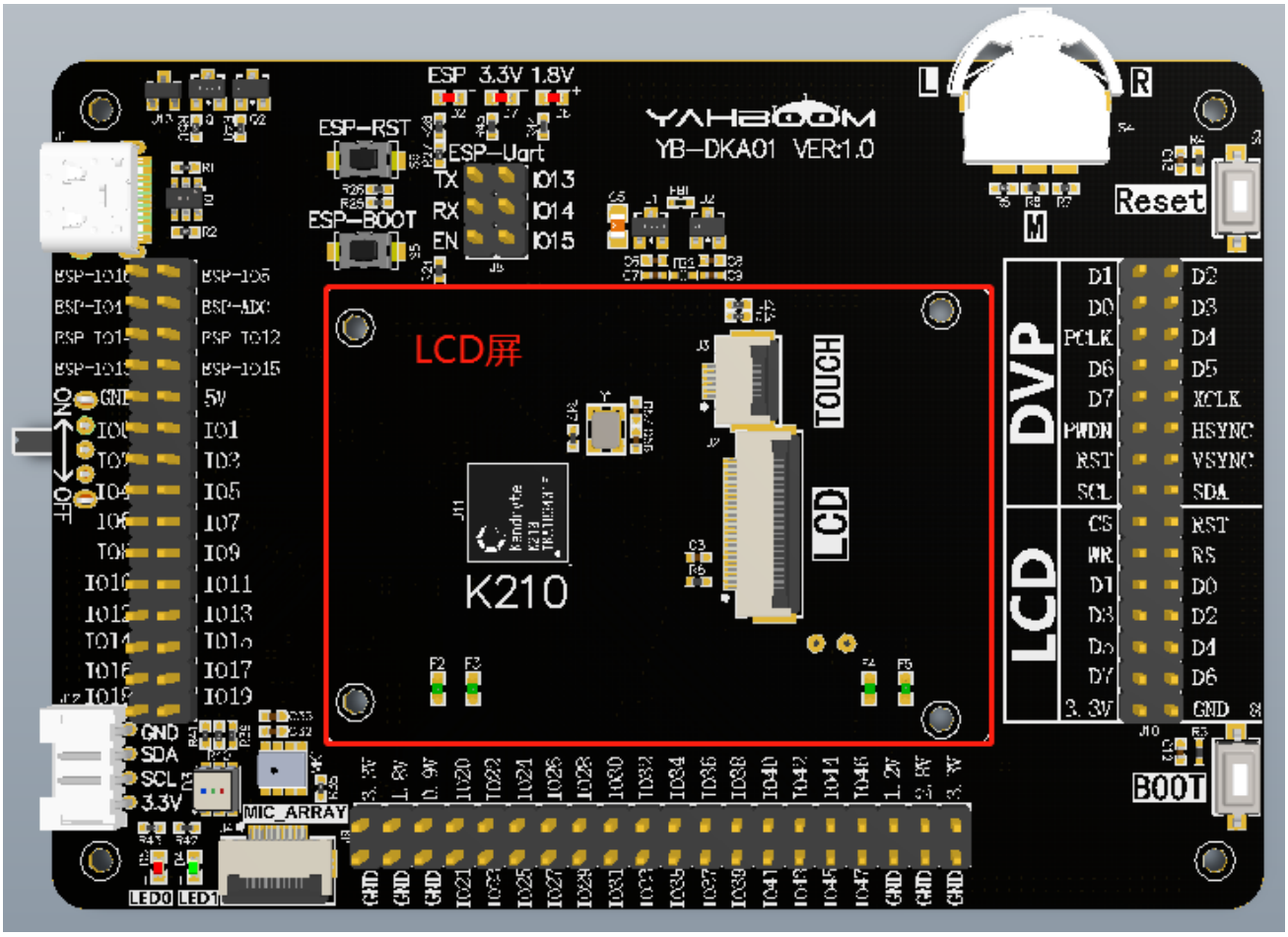
一、实验目的

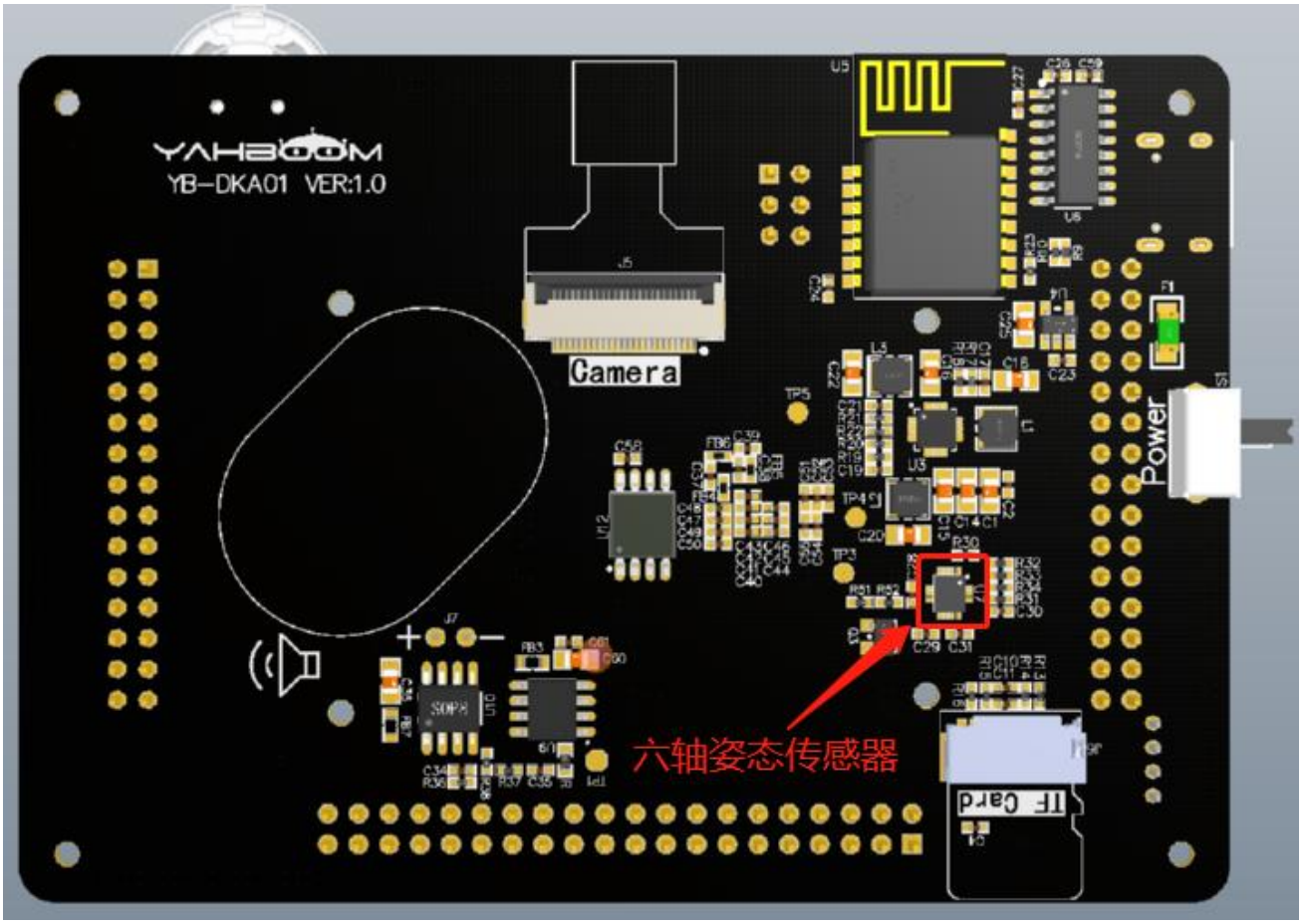
本节课主要学习K210六轴姿态传感器结合显示屏做一个水平测试板的功能。

二、实验准备

1.实验元件

LCD 显示屏、六轴姿态传感器





三、实验原理

六轴姿态传感器 ICM20607 能够提供当前开发板的姿态数据，根据 ICM20607 提供的数据，经过四元数计算方法，得到 K210 开发板的俯仰角和翻滚角，根据这两个角度数据，从而判断 K210 开发板当前的姿态。在 lvgl 图形化中显示一个机器人图标，如果 K210 开发板是平放的，则让机器人图标显示在中间，如果 K210 开发板因为摆动改变了本身的姿态，则机器人图标也会跟着移动。

四、实验过程

1. 首先根据上面的硬件连接引脚图，K210 的硬件引脚和软件功能使用的是

FPIOA 映射关系。

```
/*****************************HARDWARE-PIN*****************************
// 硬件IO口，与原理图对应
#define PIN_ICM_SCL             (9)
#define PIN_ICM_SDA             (10)
#define PIN_ICM_INT             (11)

#define PIN_LCD_CS              (36)
#define PIN_LCD_RST             (37)
#define PIN_LCD_RS              (38)
#define PIN_LCD_WR              (39)

/*****************************SOICMWARE-GPIO*****************************
// 软件GPIO口，与程序对应
#define LCD_RST_GPIONUM         (0)
#define LCD_RS_GPIONUM          (1)

#define ICM_INT_GPIONUM         (2)


/*****************************FUNC-GPIO*****************************
// GPIO口的功能，绑定到硬件IO口
#define FUNC_ICM_INT            (FUNC_GPIOHS0 + ICM_INT_GPIONUM)
#define FUNC_ICM_SCL            (FUNC_I2C0_SCLK)
#define FUNC_ICM_SDA            (FUNC_I2C0_SDA)

#define FUNC_LCD_CS             (FUNC_SPI0_SS3)
#define FUNC_LCD_RST            (FUNC_GPIOHS0 + LCD_RST_GPIONUM)
#define FUNC_LCD_RS             (FUNC_GPIOHS0 + LCD_RS_GPIONUM)
#define FUNC_LCD_WR             (FUNC_SPI0_SCLK)
```

```
void hardware_init(void)
{
    /* I2C ICM20607 */
    fpioa_set_function(PIN_ICM_SCL, FUNC_ICM_SCL);
    fpioa_set_function(PIN_ICM_SDA, FUNC_ICM_SDA);

    /* SPI lcd */
    fpioa_set_function(PIN_LCD_CS,  FUNC_LCD_CS);
    fpioa_set_function(PIN_LCD_RST, FUNC_LCD_RST);
    fpioa_set_function(PIN_LCD_RS,  FUNC_LCD_RS);
    fpioa_set_function(PIN_LCD_WR,  FUNC_LCD_WR);
    sysctl_set_spi0_dvp_data(1);
}
```

2. 设置 LCD 的 IO 口电平电压为 1.8V。。

```
static void io_set_power(void)
{
    /* 设置显示器电压为1.8V */
    sysctl_set_power_mode(SYSCTL_POWER_BANK6, SYSCTL_POWER_V18);
}
```

3. 初始化系统中断，使能系统全局中断。

```
/* 系统中断使能 */
plic_init();
sysctl_enable_irq();
```

4. 初始化显示器，并显示图片 1 秒。

```
/* LCD初始化，并显示一秒图片 */
lcd_init();
lcd_draw_picture_half(0, 0, 320, 240, gImage_logo);
sleep(1);
```

5. 初始化 lvgl，并启动定时器。

```
/* lvgl显示初始化，启动定时器 */
void lvgl_disp_init(void)
{
    lv_init();

    static lv_disp_buf_t disp_buf;
    static lv_color_t buf[LV_HOR_RES_MAX * 10];
    lv_disp_buf_init(&disp_buf, buf, NULL, LV_HOR_RES_MAX * 10);

    lv_disp_drv_t disp_drv;                 /*Descriptor of a displa
    lv_disp_drv_init(&disp_drv);            /*Basic initialization*/
    disp_drv.flush_cb = my_disp_flush;      /*Set your driver functi
    disp_drv.buffer = &disp_buf;            /*Assign the buffer to t
    lv_disp_drv_register(&disp_drv);        /*Finally register the d

    mTimer_init();
}
```

6. 定时器定时时间为每毫秒中断调用一次，在定时器中断函数中调用 lvgl 的任务管理函数和时钟函数。

```c
static void mTimer_init(void)
{
    timer_init(TIMER_DEVICE_0);
    timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1e6);
    timer_irq_register(TIMER_DEVICE_0, TIMER_CHANNEL_0, 0, 1, timer_irq_cb, NULL);

    timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
}
```

```c
static int timer_irq_cb(void * ctx)
{
    lv_task_handler();
    lv_tick_inc(1);
    return 0;
}
```

7.创建机器人图标。那么如何自定义创建自己喜欢的图片呢？打开 lvgl 相关资料中的图像转化工具，解压并双击打开 lvgl_image_convert.exe 程序，然后点击图片路径，将自己喜欢的图片添加并转换成.c 文件，然后放入到项目中，最后使用 LV_IMG_DECLARE()声明即可使用。注意分辨率不能太大。

```c
/* 声明机器人图标数据和图像结构体 */
LV_IMG_DECLARE(img_robot)
lv_obj_t * image_robot;

/* 创建机器人图标 */
void lvgl_creat_image(void)
{
    image_robot = lv_img_create(lv_scr_act(), NULL);
    lv_img_set_src(image_robot, &img_robot);
    lv_obj_align(image_robot, NULL, LV_ALIGN_IN_TOP_LEFT, -100, 0);
}
```

8.图标创建完成就要根据 ICM20607 传感器获取开发板的姿态角，根据姿态角来移动图像的位置。

```c
int16_t lvgl_x, lvgl_y;
while (1)
{
    /* 读取角度 */
    get_icm_attitude();

    /* 转化数据 */
    lvgl_x = lvgl_get_x(g_attitude.roll);
    lvgl_y = lvgl_get_y(g_attitude.pitch);

    /* 修改机器人图标的位置 */
    lvgl_move_image(lvgl_x, lvgl_y);
    msleep(1);
}
```

9. 这里使用的是四元数计算姿态角的方式，首先读取陀螺仪和加速度计的原始数据，再通过计算得到四元数。

```c
/* 读取ICM姿态角 */
void get_icm_attitude(void)
{
    icm_get_gyro();
    icm_get_acc();

    g_icm20607.accX = icm_acc_x;
    g_icm20607.accY = icm_acc_y;
    g_icm20607.accZ = icm_acc_z;
    g_icm20607.gyroX = icm_gyro_x;
    g_icm20607.gyroY = icm_gyro_y;
    g_icm20607.gyroZ = icm_gyro_z;

    get_attitude_angle(&g_icm20607, &g_attitude, DT); // 四元素算法
}
```

10. 利用归一化四元数和积分的方式计算得到四元数。

```
/* 四元素获取  dt: 5MS左右 */
void get_attitude_angle(icm_data_t *p_icm, attitude_t *p_angle, float dt)
{
    vector_t Gravity, Acc, Gyro, AccGravity;
    static vector_t GyroIntegError = {0};
    static float KpDef = 0.8f;
    static float KiDef = 0.0003f;
    float q0_t, q1_t, q2_t, q3_t;
    float NormQuat;
    float HalfTime = dt * 0.5f;

    Gravity.x = 2 * (NumQ.q1 * NumQ.q3 - NumQ.q0 * NumQ.q2);
    Gravity.y = 2 * (NumQ.q0 * NumQ.q1 + NumQ.q2 * NumQ.q3);
    Gravity.z = 1 - 2 * (NumQ.q1 * NumQ.q1 + NumQ.q2 * NumQ.q2);
    // 加速度归一化，
    NormQuat = q_rsqrt(squa(p_icm->accX)+ squa(p_icm->accY) +squa(p_icm->accZ));

    //归一后可化为单位向量下方向分量
    Acc.x = p_icm->accX * NormQuat;
    Acc.y = p_icm->accY * NormQuat;
    Acc.z = p_icm->accZ * NormQuat;

    //向量叉乘得出的值，叉乘后可以得到旋转矩阵的重力分量在新的加速度分量上的偏差
    AccGravity.x = (Acc.y * Gravity.z - Acc.z * Gravity.y);
    AccGravity.y = (Acc.z * Gravity.x - Acc.x * Gravity.z);
    AccGravity.z = (Acc.x * Gravity.y - Acc.y * Gravity.x);

    GyroIntegError.x += AccGravity.x * KiDef;
    GyroIntegError.y += AccGravity.y * KiDef;
    GyroIntegError.z += AccGravity.z * KiDef;
```

```
    //角速度融合加速度比例补偿值，与上面三句共同形成了PI补偿，得到矫正后的角速度值
    Gyro.x = p_icm->gyroX * Gyro_Gr + KpDef * AccGravity.x + GyroIntegError.x; //弧度制
    Gyro.y = p_icm->gyroY * Gyro_Gr + KpDef * AccGravity.y + GyroIntegError.y;
    Gyro.z = p_icm->gyroZ * Gyro_Gr + KpDef * AccGravity.z + GyroIntegError.z;
    // 一阶龙格库塔法，更新四元数
    //矫正后的角速度值积分，得到两次姿态解算中四元数一个实部Q0，三个虚部Q1~3的值的变化
    q0_t = (-NumQ.q1 * Gyro.x - NumQ.q2 * Gyro.y - NumQ.q3 * Gyro.z) * HalfTime;
    q1_t = (NumQ.q0 * Gyro.x - NumQ.q3 * Gyro.y + NumQ.q2 * Gyro.z) * HalfTime;
    q2_t = (NumQ.q3 * Gyro.x + NumQ.q0 * Gyro.y - NumQ.q1 * Gyro.z) * HalfTime;
    q3_t = (-NumQ.q2 * Gyro.x + NumQ.q1 * Gyro.y + NumQ.q0 * Gyro.z) * HalfTime;

    //积分后的值累加到上次的四元数中，即新的四元数
    NumQ.q0 += q0_t;
    NumQ.q1 += q1_t;
    NumQ.q2 += q2_t;
    NumQ.q3 += q3_t;

    // 重新四元数归一化，得到单位向量下
    NormQuat = q_rsqrt(squa(NumQ.q0) + squa(NumQ.q1) + squa(NumQ.q2) + squa(NumQ.q3));
    NumQ.q0 *= NormQuat;
    NumQ.q1 *= NormQuat;
    NumQ.q2 *= NormQuat;
    NumQ.q3 *= NormQuat;

    /* 计算姿态角 */
    get_angle(p_angle);
}
```

11. 在通过四元数融合角度，最后得到俯仰角和横滚角。

```
/* 四元素融合角度 */
static void get_angle(attitude_t *p_angle)
{
    vecxZ = 2 * NumQ.q0 * NumQ.q2 - 2 * NumQ.q1 * NumQ.q3; /*矩阵(3,1)项*/
    vecyZ = 2 * NumQ.q2 * NumQ.q3 + 2 * NumQ.q0 * NumQ.q1; /*矩阵(3,2)项*/
    veczZ = NumQ.q0 * NumQ.q0 - NumQ.q1 * NumQ.q1 - NumQ.q2 * NumQ.q2 + NumQ.q3 * NumQ.q3;

    p_angle->pitch = asin(vecxZ) * RtA;             //俯仰角
    p_angle->roll = atan2f(vecyZ, veczZ) * RtA;     //横滚角
}
```

12. 把横滚角的数据转化成机器人图像的 X 坐标。这里的转化功能主要是把角度-90 到 90 度的变化范围，转成屏幕的横向尺寸大小，注意要减掉机器人图像的宽度。

```
/* 把翻滚角转化成图像的x坐标 */
int16_t lvgl_get_x(float a)
{
    int16_t temp = (int16_t)a;
    temp = temp * (-1);
    temp = temp > 0 ? (temp - 180) : (temp + 180);
    temp = convert_value(temp, -90, 90, 0, 320-IMG_ROBOT_WIDTH);

    return temp;
}
```

```
/* 映射关系转化 */
int16_t convert_value(int16_t x, int16_t in_min, int16_t in_max, int16_t out_min, int16_t out_max)
{
    int16_t res = (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
    if (res <= out_min) res = out_min;
    else if (res >= out_max) res = out_max;
    return res;
}
```

14. 把俯仰角的数据转成图像的 Y 坐标。

```
/* 把俯仰角转化成图像的Y坐标 */
int16_t lvgl_get_y(float a)
{
    int16_t temp = (int16_t)a;
    // temp = temp * (-1);
    temp = convert_value(temp, -90, 90, 0, 240-IMG_ROBOT_HIGH);

    return temp;
}
```

15. 最后是根据 XY 的值来改变机器人图像的位置。

```
/* 修改机器人图标的位置 */
void lvgl_move_image(int16_t x, int16_t y)
{
    lv_obj_set_pos(image_robot, (lv_coord_t)x, (lv_coord_t)y);
}
```

16.编译调试，烧录运行

把本课程资料中的 icm20607 复制到 SDK 中的 src 目录下，然后进入 build 目录，运行以下命令编译。

cmake .. -DPROJ=icm20607 -G "MinGW Makefiles"

make

```
Generating .bin file ...
[100%] Built target icm20607
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

编译完成后，在 build 文件夹下会生成 icm20607.bin 文件。

使用 type-C 数据线连接电脑与 K210 开发板，打开 kflash，选择对应的设备，再将程序固件烧录到 K210 开发板上。

五、实验现象

烧录完成固件后，系统会弹出一个终端界面，如果没有弹出终端界面的可以打开串口助手显示调试内容。

终端会打印 ICM20607 传感器的 ID 号，如果是烧录完成后，需要手动按一下复位按键才可以正常识别到 ID。

```
C:\Users\Administrator\AppData\Local\Temp\tmpC06C.tmp
ID error! WHO_AM_I=0x00
Please press the reset key to reboot
WHO_AM_I=0x05
```

开机屏幕会先显示 1 秒图片，然后变成整个屏幕白色，并且显示一个机器人图像，等待 3 秒机器人稳定在显示器中间的时候，左右摆动开发板，机器人图像会左右移动，前后摆动开发板，机器人图像会左右上下移动。重新放回水平桌面上则机器人返回中间。

六、实验总结

1.从 icm20607 传感器读出的数据不能直接拿来计算角度，需要经过计算成四元数后才能计算出角度。

2.机器人图标显示的位置是以左上角为开始点，所以计算位置的时候需要把机器人图标的长宽尺寸减掉，否则在最右边的图标会显示不全。

3.烧录完固件后需要按一下复位键，并且等待传感器稳定后再操作。