

### 5.3 Horizontal test board

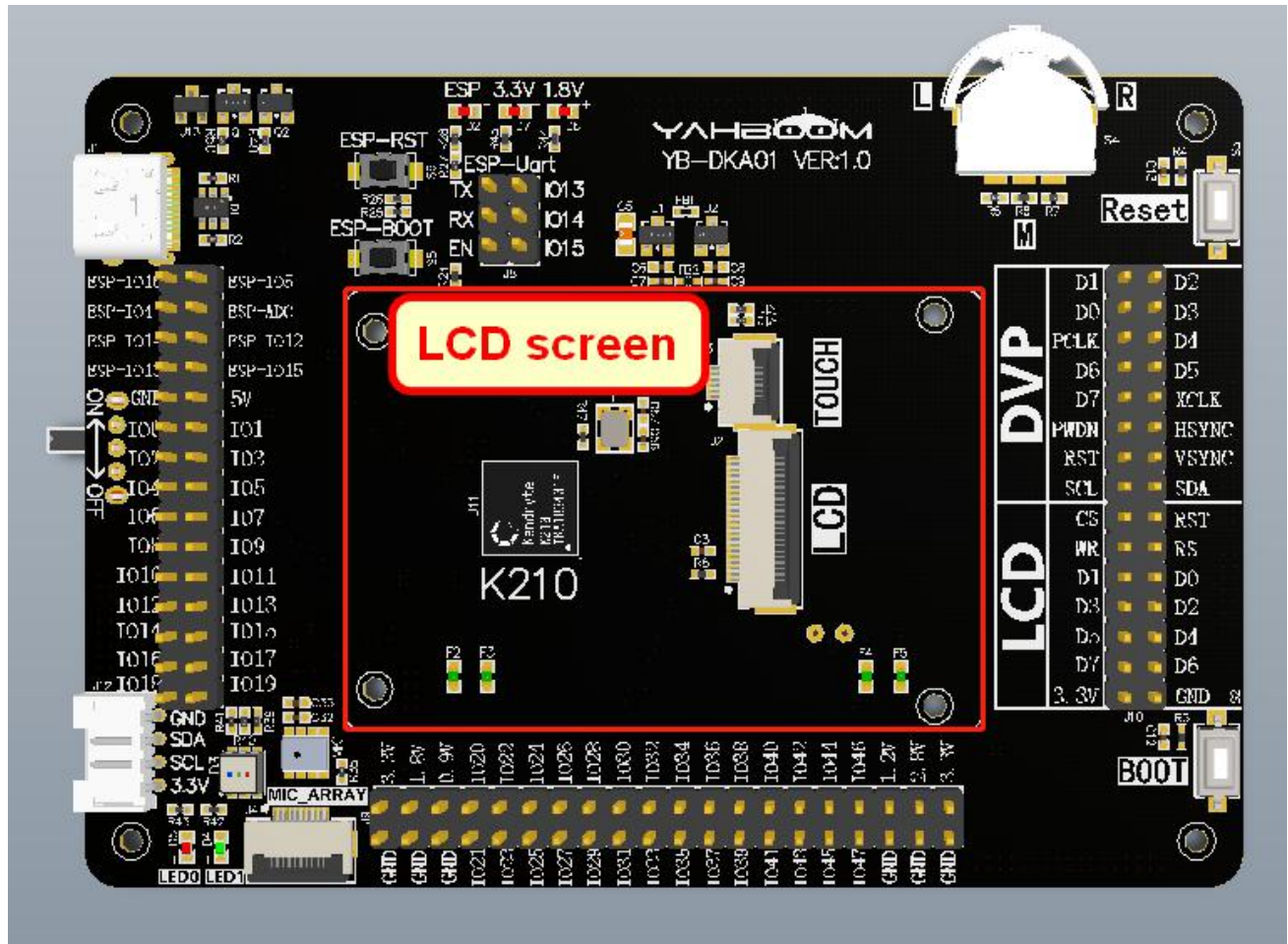
#### 1. Experiment purpose

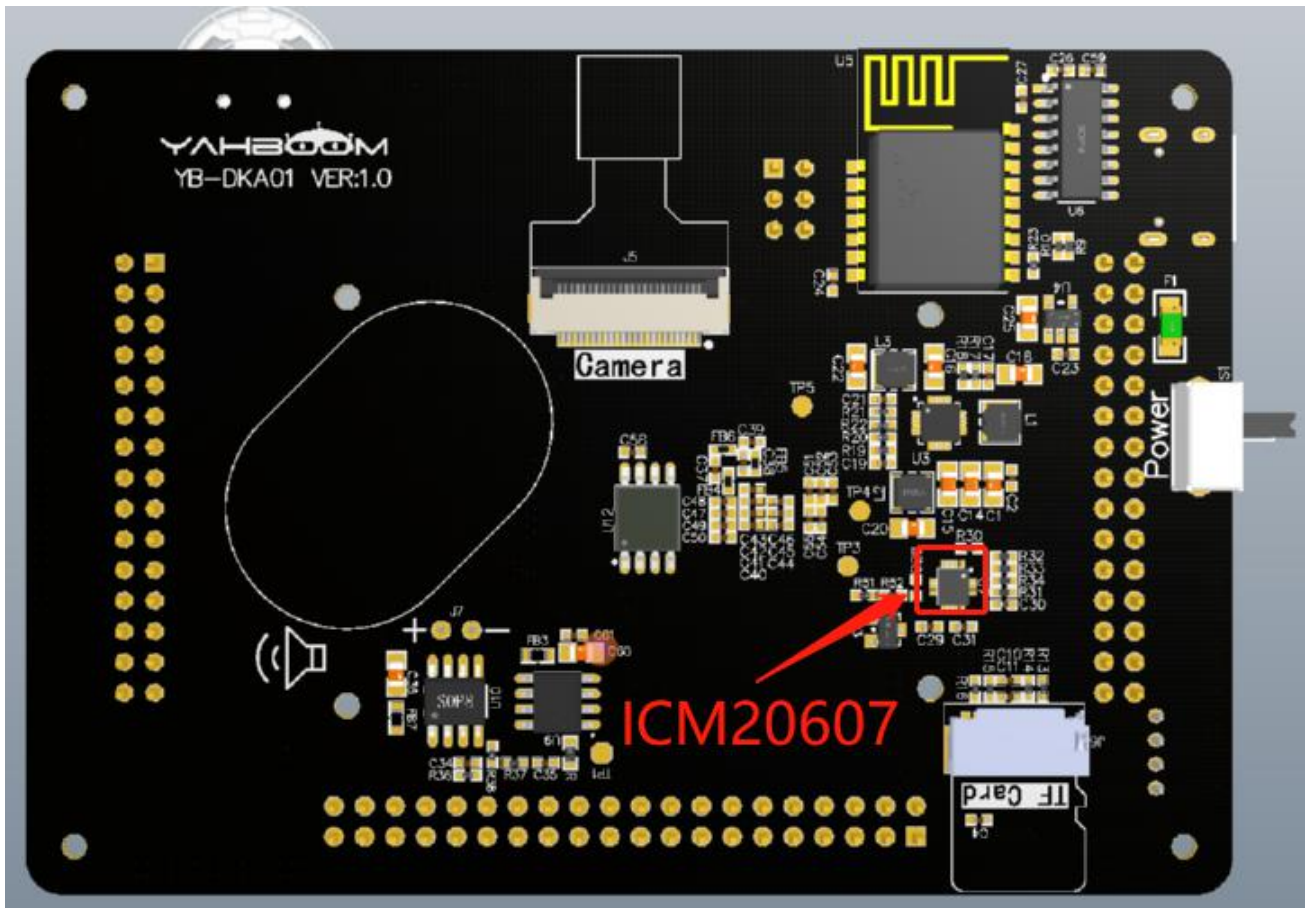
In this lesson, we mainly learn how to make the K210 six-axis attitude sensor combined with the display screen functions as a level test board.

#### 2.Experiment preparation

##### 2.1 components

LCD, Gyroscope





### 3. Experiment principle

The six-axis attitude sensor ICM20607 can provide the current attitude data of the development board. According to the data provided by ICM20607, through the quaternion calculation method, the pitch and roll angles of the K210 development board are obtained.

Based on these two angle data, the K210 development board can be judged The current posture. Display a robot icon in the lvgl graphics. If the K210 development board is flat, let the robot icon be displayed in the middle. If the K210 development board changes its posture due to the swing, the robot icon will also move.

### 4. Experiment procedure

4.1 K210's hardware pins and software functions use the FPIOA mapping relationship.

```

/*****HARDWARE-PIN*****/
//Hardware IO port, corresponding Schematic
#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)

/*****SOFTWARE-GPIO*****/
//Software GPIO port, corresponding program
#define LCD_RST_GPIONUM   (0)
#define LCD_RS_GPIONUM    (1)

#define ICM_INT_GPIONUM   (2)

/*****FUNC-GPIO*****/
//Function of GPIO port, bound to hardware IO port
#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);
}

```

4.2 Set the IO port level voltage of the LCD to 1.8V.



```
static void io_set_power(void)
{
    /*Set the IO port level voltage of the LCD to 1.8V. */
    sysctl_set_power_mode(SYSCTL_POWER_BANK6, SYSCTL_POWER_V18);
}
```

4.3 Initialize the system interrupt and enable the system global interrupt.

```
/* System interrupt enable */
plic_init();
sysctl_enable_irq();
```

4.4 The LCD initializes and displays a picture for one second

```
/* The LCD initializes and displays a picture for one second */
lcd_init();
lcd_draw_picture_half(0, 0, 320, 240, gImage_logo);
sleep(1);
```

4.5 Initialize lvgl and start the timer.

```
/* lvgl display initialization, start timer */
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 display driver*/
    lv_disp_drv_init(&disp_drv);      /*Basic initialization*/
    disp_drv.flush_cb = my_disp_flush; /*Set your driver function*/
    disp_drv.buffer = &disp_buf;     /*Assign the buffer to the display*/
    lv_disp_drv_register(&disp_drv);  /*Finally register the driver*/

    mTimer_init();
}
```

4.6 The timer timing time is called once every millisecond interrupt, and the task management function and clock function of lvgl are called in the timer interrupt function.

```
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);
}

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

4.7 Create robot icon.

```
/* Declare robot icon data and image structure*/
LV_IMG_DECLARE(img_robot)
lv_obj_t * image_robot;

/* Create robot icon*/
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);
}
```

4.8 Obtain the attitude angle of the development board according to the ICM20607 sensor, and move the position of the image according to the attitude angle.

```
int16_t lvgl_x, lvgl_y;
while (1)
{
    /* Reading angle */
    get_icm_attitude();

    /* Conversion data*/
    lvgl_x = lvgl_get_x(g_attitude.roll);
    lvgl_y = lvgl_get_y(g_attitude.pitch);

    /* Modify the position of the robot icon*/
    lvgl_move_image(lvgl_x, lvgl_y);
    msleep(1);
}
```

4.9 We use the quaternion method to calculate the attitude angle. First, read the original data of

the gyroscope and accelerometer, and then calculate the quaternion.

```
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);
}
```

4.10 The quaternion is calculated by normalized quaternion and integral.

```

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);
    // Normalized acceleration,
    NormQuat = q_rsqrt(squa(p_icm->accX) + squa(p_icm->accY) + squa(p_icm->accZ));

    //After normalization, it can be reduced to the downward direction component of the unit vector
    Acc.x = p_icm->accX * NormQuat;
    Acc.y = p_icm->accY * NormQuat;
    Acc.z = p_icm->accZ * NormQuat;

    //The value obtained by the cross product of the vector. After the cross product, the deviation of
    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;

    //Angular velocity fusion acceleration proportional compensation value, and the above three sentences
    Gyro.x = p_icm->gyroX * Gyro_Gr + KpDef * AccGravity.x + GyroIntegError.x; //弧度制, 此处补偿的是角速
    Gyro.y = p_icm->gyroY * Gyro_Gr + KpDef * AccGravity.y + GyroIntegError.y;
    Gyro.z = p_icm->gyroZ * Gyro_Gr + KpDef * AccGravity.z + GyroIntegError.z;

    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;

    NumQ.q0 += q0_t;
    NumQ.q1 += q1_t;
    NumQ.q2 += q2_t;
    NumQ.q3 += q3_t;

    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;

    get_angle(p_angle);
}

```

4.11 After the quaternion fusion angle, the pitch angle and roll angle are finally obtained.



```

/* Four elements integration angle */
static void get_angle(attitude_t *p_angle)
{
    vecxZ = 2 * NumQ.q0 * NumQ.q2 - 2 * NumQ.q1 * NumQ.q3; /*Matrix (3,1) item*/
    vecyZ = 2 * NumQ.q2 * NumQ.q3 + 2 * NumQ.q0 * NumQ.q1; /*Matrix (3,2) item*/
    veczZ = NumQ.q0 * NumQ.q0 - NumQ.q1 * NumQ.q1 - NumQ.q2 * NumQ.q2 + NumQ.q3 * NumQ.q3; /*Matrix (3,3) item*/

    p_angle->pitch = asin(vecxZ) * RtA;          //Pitch angle
    p_angle->roll = atan2f(vecyZ, veczZ) * RtA;   //Roll angle
}

```

4.12 Convert the roll angle data into the X coordinate of the robot image.

```

/*Convert the roll angle to the X coordinate of the image */
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;
}

```

```

/*Mapping relationship conversion */
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;
}

```

4.13 Convert the pitch angle data to the Y coordinate of the image.

```

/*Convert the pitch angle to the Y coordinate of the image*/
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;
}

```

4.14 Change the position of the robot image according to the value of XY.



```
/*Modify the position of the robot icon */
void lvgl_move_image(int16_t x, int16_t y)
{
    lv_obj_set_pos(image_robot, (lv_coord_t)x, (lv_coord_t)y);
}
```

#### 4.15 Compile and debug, burn and run

Copy the icm20607 to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

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

**make**

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

After the compilation is complete, the **icm20607.bin** file will be generated in the build folder.

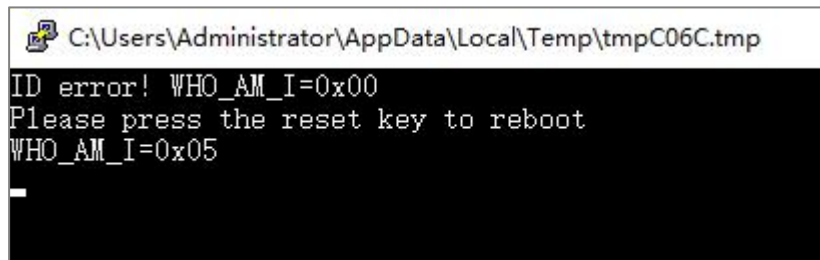
We need to use the type-C data cable to connect the computer and the K210 development board.

Open kflash, select the corresponding device, and then burn the **icm20607.bin** file to the K210 development board.

### 5. Experimental phenomenon

After the firmware is write, a terminal interface will pop up. **If the terminal interface does not pop up, we can open the serial port assistant to display the debugging content.**

When we press the reset button on K210 board, the terminal interface will print the ID number of the ICM20607 sensor.

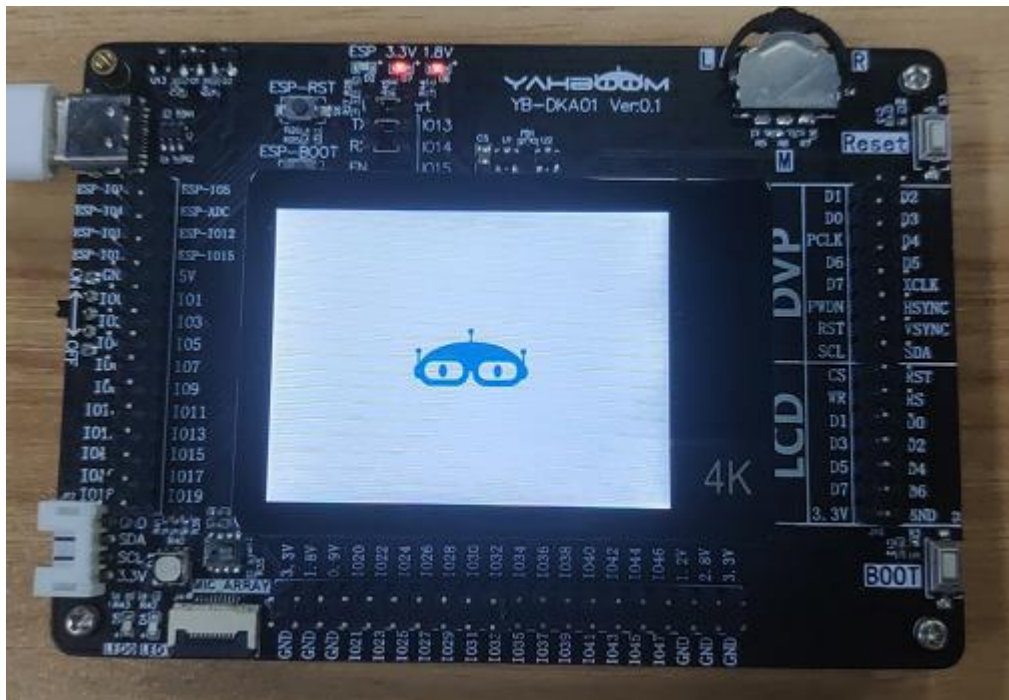


```
C:\Users\Administrator\AppData\Local\Temp\tmpC06C.tmp
ID error! WHO_AM_I=0x00
Please press the reset key to reboot
WHO_AM_I=0x05
█
```

We can see that the LCD screen will display a picture for 1 second, then the entire screen will become white, and a robot image will be displayed. We put the K210 development board horizontally on the desktop, and after waiting for 3 seconds, the robot will stabilize in the middle of the LCD.

When we tilt the development board in different directions, the robot image will move with it.

When we place the development board horizontally on the desktop again, the robot image will return to the middle position.



## 6. Experiment summary

6.1 The data read from the icm20607 sensor cannot be used to calculate the angle directly, it needs to be calculated into a quaternion to calculate the angle.

6.2 The display position of the robot icon is the upper left corner as the starting point. When calculating the position, we need to subtract the length and width of the robot icon, otherwise the icon on the far right will be incomplete.

6.3 After burning the firmware, we need to press the reset button of K210 board and wait for the sensor to stabilize. Then, start operate it.