南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

计算机科学与工程学院

“嵌入式系统”实验报告书

题目：嵌入式系统作业 6——I2CICM

学号：922106840127

姓名：刘宇翔

成绩

日期：　　2025 年　　4 月　　1 日

1 题目要求

1. 题目设计要求

【1】作业目标：

（1）熟悉 I2C 工作原理；

（2）能够熟练 I2C 开发编程；

（3）掌握数据滤波方法和实现，能够使用陀螺传感器进行数据读写；

（4）了解惯导传感器的姿态的解算方法。

【2】作业内容：

完成对陀螺传感器 ICM-20608-G，以 I2C 接口对传感器数据读出，实时显示三个方向的加速度和角速率，并对以上数据开展数据处理，如均值滤波，一阶/二阶低通滤波（LPF）后，解算出开发板的俯仰角和滚转角（姿态角）和航向变化角度。以上数据实时显示在 LCD 屏上。

【3】完成要求：

（1）完成对传感器数据读写；

（2）传感器原始数据量化后，加速度量：xx.xxx m/s2，角速率量：xxx.x x° /s，并给出结果写 LCD 上；

（3）对数据完成滤波处理后分别解算出姿态角和航向变化角度，实时显示 LCD；

2. 拟实现的具体功能

本实验旨在通过 I2C 接口实现对 ICM-20608-G 陀螺传感器的数据读取，掌握 I2C 通信原理及惯性传感器的数据处理方法。实验首先配置 I2C 总线，实现对加速度计和陀螺仪的原始数据读取，并进行物理量转换，将加速度数据转换为单位 m/s²，角速率转换为°/s，并实时显示在 LCD 屏幕上。

随后，利用均值滤波和一阶/二阶低通滤波（LPF）对传感器数据进行平滑处理，以减少噪声干扰。基于滤波后的数据，进一步采用姿态解算算法计算开发板的俯仰角（Pitch）、滚转角（Roll）以及航向变化角度（Yaw），并将计算结果实时输出至 LCD 屏幕，实现直观显示。

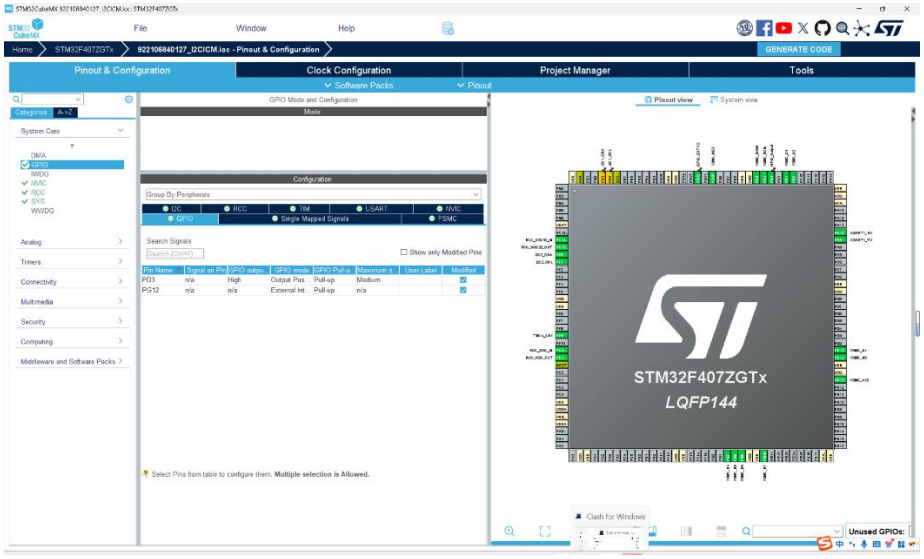
本实验不仅要求完成传感器数据采集和滤波处理，还需确保数据的实时性和准确性，以增强对惯性导航系统的理解，并为后续更复杂的姿态解算和传感器融合算法奠定基础。

2 总体设计

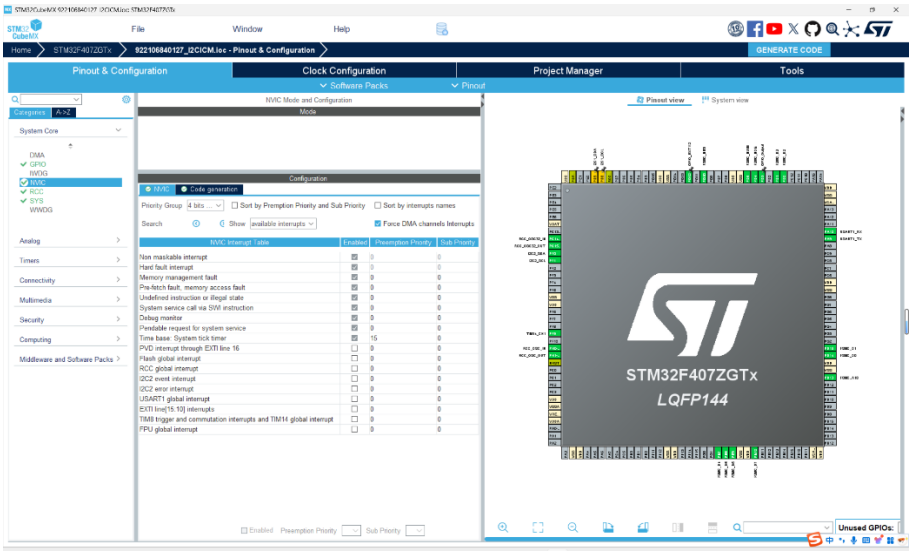
2.1 硬件设计

1. 硬件设计思路

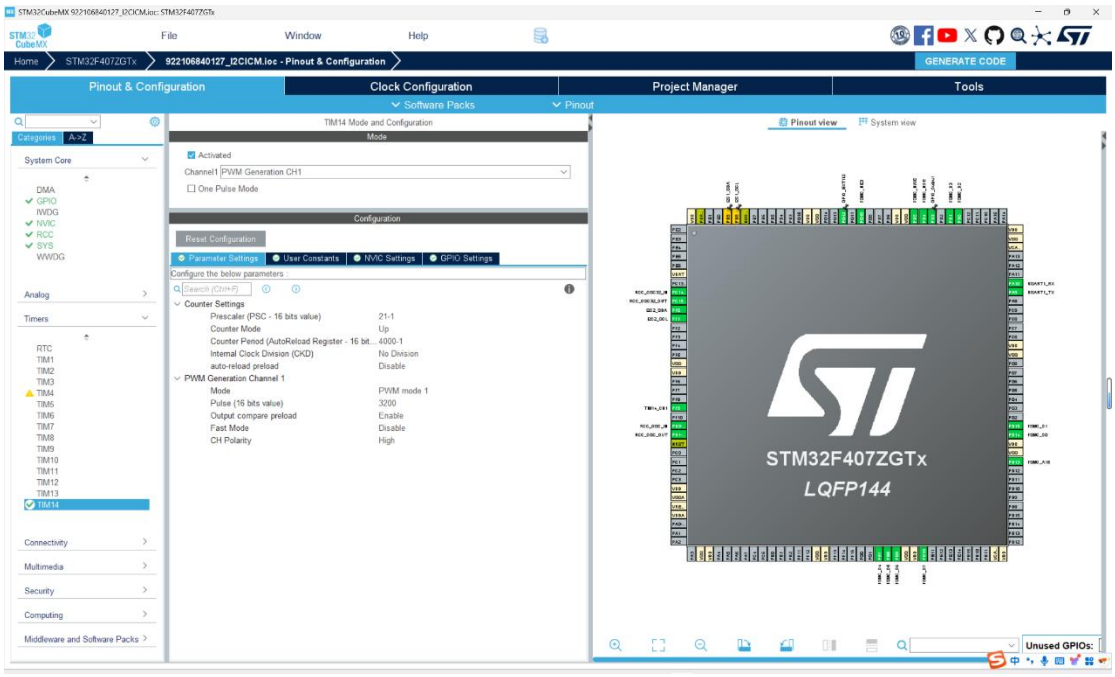
我通过查阅相关开发板原理构建硬件图和课本的参考指导确定了本次作业中会使用到的引脚与特殊设置的内容，例如 SPI,GPIO,USART 等相关设置，并对 LCD 的驱动进行了添加与相应的配置，完成整体的硬件设计，其中具体的代码工程配置如下：



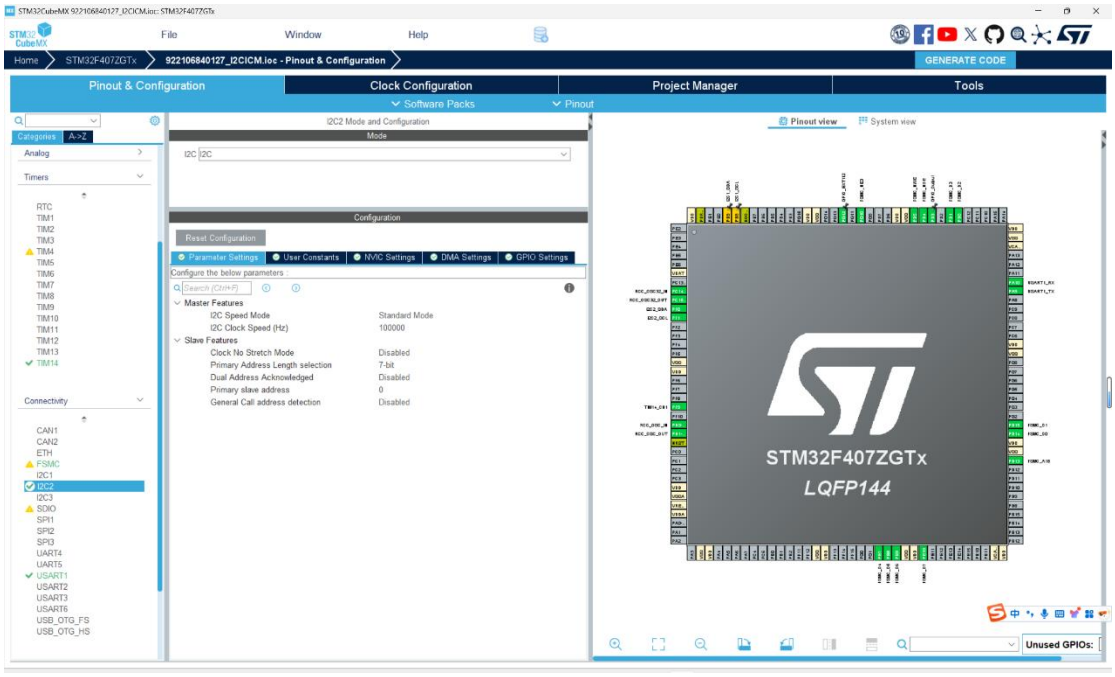
GPIO 端口设置



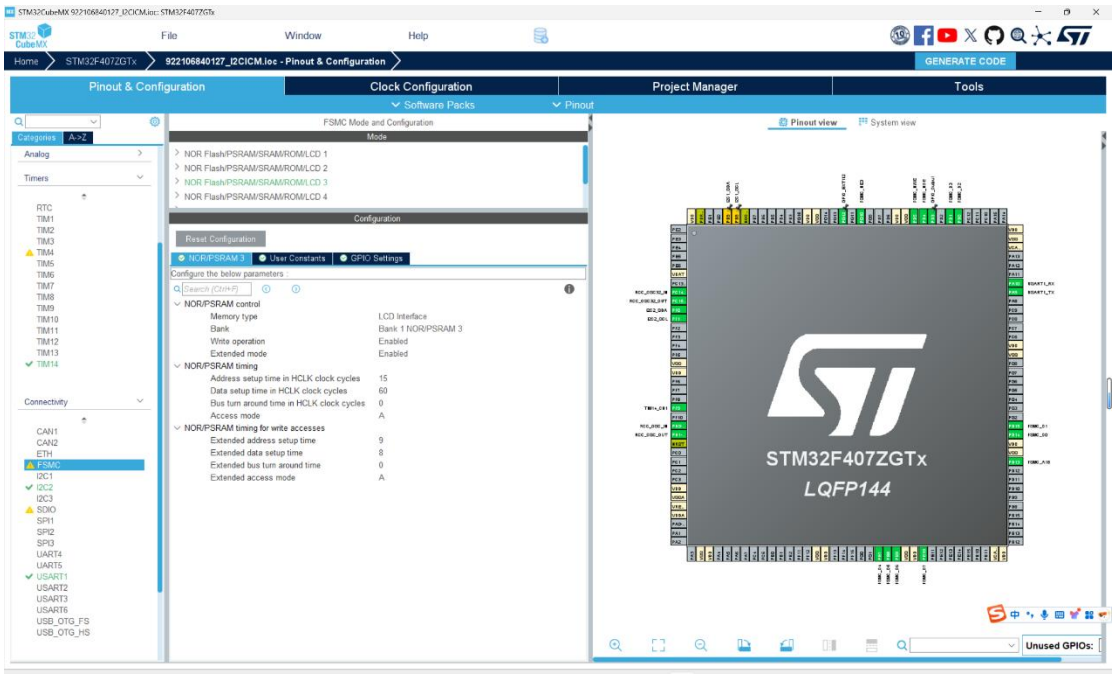
NVIC 设置



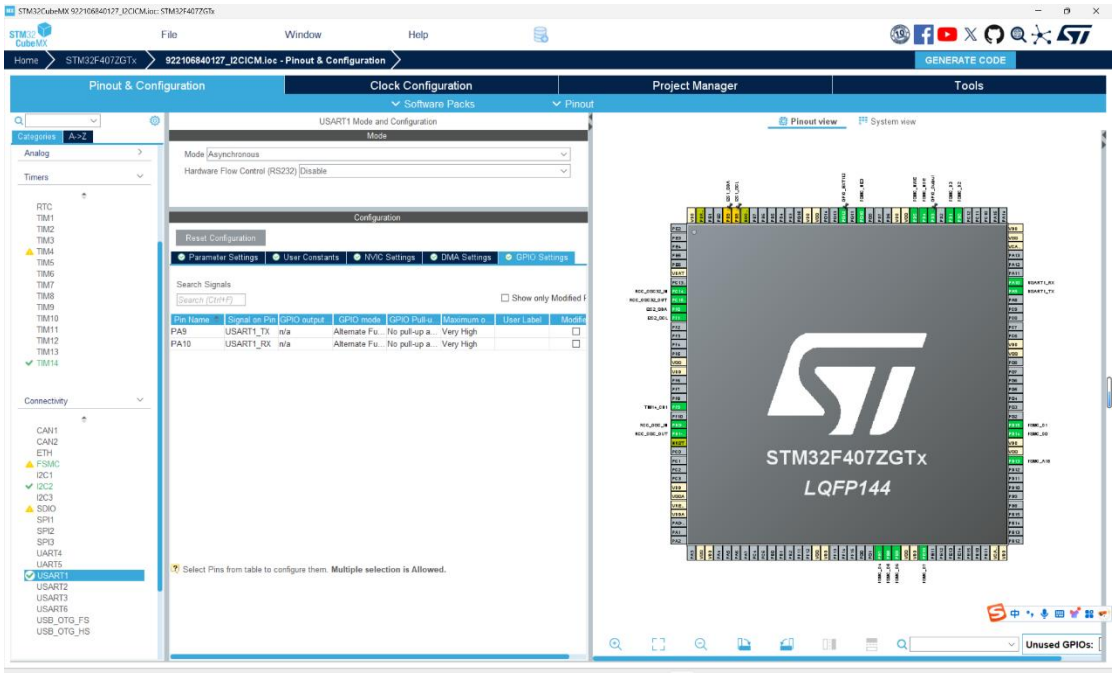
TIM14 设置



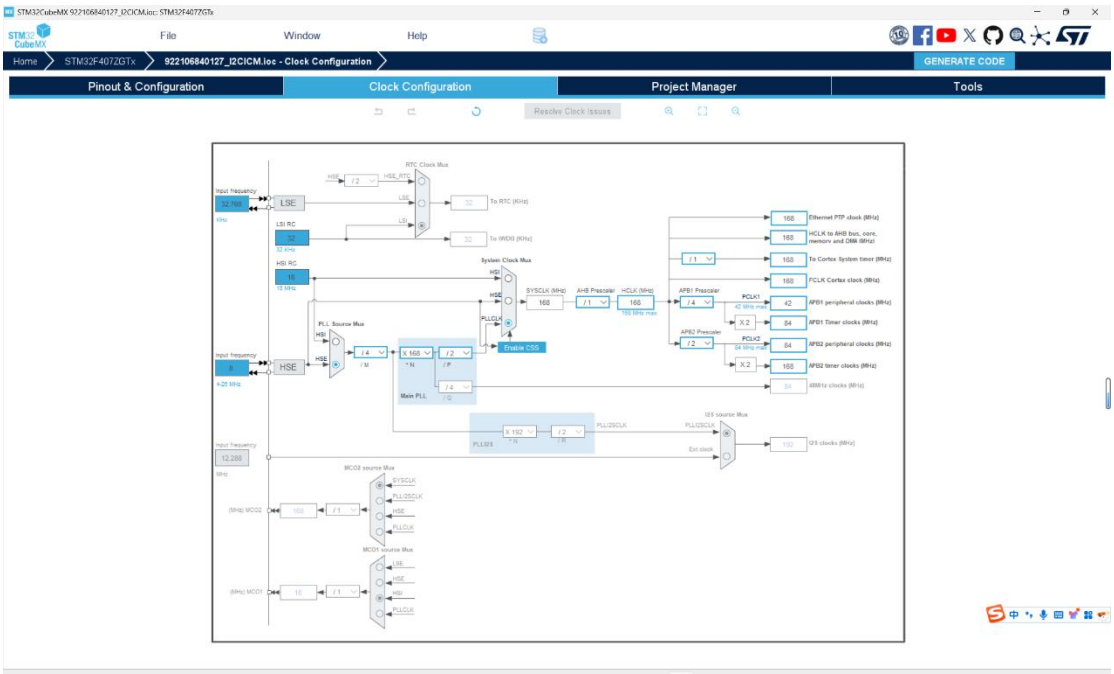
I2C2 设置



FSMC 设置



USART 设置



时钟配置表

此外，一些普遍的工程配置在本报告中不再展示（例如 RCC 时钟，编译工具链选择 MDK-ARM V5.32 等）。 以上配置图即为我作为 STM32CUBEMX 进行的配置设置，设置后点击 “Generate Code”生成初始化代码，继续进行接下来的代码软件设计部分。

2.2 软件设计

1. 软件设计概述

（1）概述

本项目基于 STM32 微控制器，使用 I2C 接口读取 ICM-20608-G 惯性传感器的数据，并进行信号处理以计算设备的姿态信息。系统采用多级滤波算法对数据进行优化，并在 LCD 屏幕上实时显示处理后的传感器数据。

（2）体系结构

系统采用分层设计，主要包括以下模块：

- **硬件抽象层（HAL）**：基于 STM32 HAL 库，封装 I2C、USART、GPIO 等底层驱动。
- **传感器驱动层**：实现 ICM-20608-G 的初始化、数据读取及中断处理。
- **信号处理层**：包括均值滤波（Mean Filter）和低通滤波（LPF），用于数据

去噪和平滑。

- **姿态计算层**：利用加速度计数据计算俯仰角（Pitch）和横滚角（Roll），并使用角速度积分计算偏航角（Yaw）。
- **UI 显示层**：负责数据的 LCD 屏幕显示及串口输出。

（3）关键技术实现

【1】传感器初始化与数据读取

系统通过 I2C 接口初始化 ICM-20608-G，包括时钟源设置、量程配置、低通滤波器（DLPF）配置等。数据读取采用**轮询**方式获取加速度计和陀螺仪的原始测量值，并进行数据转换。

【2】信号滤波

- **均值滤波**：维护一个固定大小的滑动窗口，计算其均值，以降低随机噪声。
- **低通滤波（LPF）**：基于指数平滑算法，抑制高频噪声，提高数据平稳性。

【3】姿态角计算

- **俯仰角（Pitch）与横滚角（Roll）**：基于加速度计数据计算倾斜角度。
- **偏航角（Yaw）**：通过对 Z 轴角速度积分得到偏航角变化量。

【4】数据可视化

数据通过 LCD 屏幕进行实时显示，包括传感器的加速度、角速度及姿态信息。此外，系统使用 UART 串口输出数据，以便进一步分析和调试。

（4）主要功能

- 读取 ICM-20608-G 的加速度和角速度数据。
- 进行均值滤波和低通滤波。
- 计算 Pitch、Roll、Yaw 角。
- 通过 LCD 实时显示数据。
- 通过 UART 输出数据，便于调试。

（5）小结

本系统通过 I2C 接口完成了惯性传感器数据的采集、滤波与处理，利用数学计算得到设备姿态信息，并通过 LCD 直观显示。该设计适用于嵌入式运动检测、机器人导航等应用场景。

2.软件流程分解

下面给出 I2CICM 项目的软件流程分解说明，其整体流程可分为以下几个阶段：

A. 初始阶段

开始 → 系统初始化

程序启动后首先进入系统初始化阶段，主要完成 MCU 及各外设的初始化工作。此阶段包括：

• 系统时钟配置，通过 SystemClock_Config 确保 MCU 工作在稳定频率下；

• GPIO、FSMC、定时器（TIM14）和 USART 串口初始化，为按键、LCD 显示、PWM 输出及串口调试奠定基础；

• I2C 接口（I2C2）的初始化，确保与 ICM-20608-G 传感器的通信正常；

• LCD 显示驱动的初始化，通过 drv_lcd_init 及 lcd_clear 确保显示模块处于正常工作状态；

• 调用 ICM20608_Init 函数，对陀螺仪/加速度计进行初始化配置，包括时钟源选择、量程设置、低通滤波器（DLPF）配置等。

B. 数据采集与预处理阶段

进入主循环后，系统不断进行数据采集与预处理。主要流程如下：

• 传感器数据读取：调用 ICM20608_ReadRawData 函数，通过 I2C 接口读取加速度和陀螺仪的原始数据（14 字节数据）；

• 数据转换：将原始加速度数据转换为物理量（单位：m/s^2），采用公式 ax = accel[0] * 9.81 / ACCEL_SENSITIVITY 等，将数据量化；

• 数据滤波处理：对转换后的加速度数据采用两级滤波——首先使用均值滤波（MeanFilter_Update）进行滑动窗口求平均，再利用低通滤波（LPF_Update）对均值滤波结果进行指数平滑，获得稳定的 ax_filtered、ay_filtered、az_filtered；

• 角速度数据同样经过低通滤波处理，分别处理 X、Y、Z 轴角速度数据，确保信号稳定。

C. 姿态解算阶段

在数据预处理的基础上，系统进行姿态角计算，流程如下：

• 调用 CalculateAngles 函数，根据滤波后的加速度数据计算俯仰角（Pitch）和横

滚角（Roll），采用反三角函数求解；

• 对低通滤波后的陀螺仪 Z 轴角速度数据进行积分计算，依据采样间隔（dt=10ms）累计更新偏航角（Yaw）。

D. 数据显示与调试阶段

完成数据采集、滤波与姿态计算后，系统将结果实时展示和调试输出：

• 调用 LCD_Display 函数，将加速度（ax_filtered、ay_filtered、az_filtered）、角速度（低通滤波后的 gx、gy、gz）及计算得到的姿态角（Pitch、Roll、Yaw）格式化显示在 LCD 屏上；

• 通过 UART 串口输出部分调试信息（如 Yaw 值），便于系统性能监控和问题排查。

E. 循环调度与系统保护

系统在主循环中，每次完成一轮数据处理后，通过 HAL_Delay 延时（50ms），确保数据采集周期稳定；同时，若在数据读取或其他操作中发生错误，将调用 Error_Handler 函数，进入保护模式以避免系统异常扩散。

总体流程为：系统初始化 → 传感器及外设配置 → 主循环中进行数据采集、滤波与姿态计算 → LCD 及串口实时显示与调试输出 → 延时调度后循环重复。此流程确保了从硬件初始化、数据获取、信号处理到结果显示的各个环节有序、高效运行，实现 I2CICM 项目预定功能。

3. 函数设计与功能实现

以下对代码中几个关键函数的设计与功能实现进行说明：

（1） **ICM20608_Init**

该函数用于初始化 ICM-20608 传感器。在设计上，函数首先配置传感器的电源管理寄存器，选择合适的时钟源（PLL 模式），并确保设备唤醒；接着配置陀螺仪和加速度计的量程，分别设定为±250°/s 和±2g；此外，还设置数字低通滤波器（DLPF）参数，以降低高频噪声影响。通过对多个寄存器（如 ICM20608_PWR_MGMT1_REG 、 ICM20608_GYRO_CONFIG_REG 、 ICM20608_ACCEL_CONFIG1_REG 等）的连续写操作，实现对传感器的全面配置，确保后续数据采集的准确性和稳定性。

（2） **ICM20608_ReadRawData**

此函数负责通过 I2C 接口从传感器中读取连续的 14 字节数据，包含加速度计和陀螺仪的原始数据。设计时，函数首先通过 HAL_I2C_Mem_Read 调用读取数据到缓存区，并对返回状态进行检测，若出现错误则输出错误信息。随后，将数据解析成加速度和角速度的三个方向数值（高低字节合成整型数据），便于后续的物理量转换和滤波处理。该函数的设计重点在于高效、稳定的数据采集以及错误检测机制。

（3） **MeanFilter_Update**

用于对输入数据进行均值滤波。设计思路为使用固定大小的滑动窗口，将最新的采样数据存入环形缓冲区，并更新索引。函数内遍历整个缓冲区计算平均值，输出即为滤波后的结果。该函数简单有效地降低了数据中的随机噪声，适合初级信号平滑处理。

（4） **LPF_Update**

实现低通滤波处理，其设计基于一阶指数平滑算法。函数通过给定的滤波系数 alpha，结合当前输入数据与上一次滤波输出，计算出新的滤波值，并更新内部状态。此方法能够有效抑制高频干扰，进一步提升数据平稳性。设计上，LPF 函数既独立又易于复用，可针对不同数据采用不同的滤波参数。

（5） **CalculateAngles**

此函数主要用于基于加速度计的滤波数据计算设备的俯仰角（Pitch）和横滚角（Roll）。其设计利用反正切函数和平方根运算，根据加速度在各轴的分量求出角度，并转换为角度制（°）。这种方法适用于静态或低动态场景下的姿态估计，为后续融合陀螺仪数据提供基础。

（6） **LCD_Display**

用于将传感器数据和姿态计算结果实时显示在 LCD 屏上。设计中，函数通过格式化字符串，将加速度、角速度和计算出的 Pitch、Roll、Yaw 等数据分段显示在屏幕不同位置。该函数不仅实现了数据可视化，同时也支持调试输出，使用户能直观地监控系统运行状态。

总体来看，这些关键函数各司其职：从传感器的初始化、数据采集到信号滤波与姿态解算，再到数据的实时显示，共同构成了 I2CICM 项目中数据处理和信息反馈的核心模块。

4. μvision 详细代码

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "fsmc.h"

#include "stdio.h"
//#include "stdlib.h"
#include "./BSP/LCD/drv_lcd.h"
#include "./BSP/LCD/rttlogo.h"

#include "math.h"



#define ICM20608_ADDR 0x68   // ??AD0??,???0x68
#define ICM20608_CONFIG_REG 0x1A              // configuration:fifo, ext sync and
dlpf
#define ICM20608_GYRO_CONFIG_REG 0x1B      // gyroscope configuration
#define ICM20608_ACCEL_CONFIG1_REG 0x1C // accelerometer configuration
#define ICM20608_ACCEL_CONFIG2_REG 0x1D // accelerometer configuration
#define ICM20608_INT_ENABLE_REG 0x38        // interrupt enable
#define ICM20608_ACCEL_MEAS 0x3B             // accelerometer measurements
#define ICM20608_GYRO_MEAS 0x43              // gyroscope measurements
#define ICM20608_PWR_MGMT1_REG 0x6B         // power management 1
#define ICM20608_PWR_MGMT2_REG 0x6C         // power management 2
```

```
#define FILTER_SIZE 5
#define M_PI 3.1415927f
// ???????????? ±2g,1g = 9.81 m/s²,??? 16384
#define ACCEL_SENSITIVITY 16384.0f
// ???????(??:?/?),?????????
#define GYRO_SENSITIVITY 1.0f



int16_t accel[3];
int16_t gyro[3];

typedef struct {
    float buffer[FILTER_SIZE];
    uint8_t index;
} MeanFilter;

typedef struct {
    float prev_output;
    float alpha; // a = dt / (dt + 1/(2pf_cutoff))
} LPF;

float ax_filtered,ay_filtered,az_filtered;
LPF lpf_ax,lpf_ay,lpf_az;

        // ??????
MeanFilter accel_filter_x, accel_filter_y, accel_filter_z;
LPF lpf_gyro_z = { .alpha = 0.1f }; // ????a
LPF lpf_gyro_x = { .alpha = 0.1f }; // ????a
LPF lpf_gyro_y = { .alpha = 0.1f }; // ????a
```

```
void ICM20608_Init(void) {
    // ??????,?????(PLL)
    uint8_t data = 0x01; // CLKSEL=1, SLEEP=0
    HAL_I2C_Mem_Write(&hi2c2, (ICM20608_ADDR << 1), ICM20608_PWR_MGMT1_REG, 1, &data, 1, 100);

    // ??????? ±250dps
    data = 0x00; // FS_SEL=0
    HAL_I2C_Mem_Write(&hi2c2, (ICM20608_ADDR << 1), ICM20608_GYRO_CONFIG_REG, 1, &data, 1, 100);

    // ???????? ±2g
    data = 0x00; // AFS_SEL=0
    HAL_I2C_Mem_Write(&hi2c2, (ICM20608_ADDR << 1), ICM20608_ACCEL_CONFIG1_REG, 1, &data, 1, 100);

    // ?????DLPF?? 92Hz
    data = 0x06; // DLPF_CFG=6
    HAL_I2C_Mem_Write(&hi2c2, (ICM20608_ADDR << 1), ICM20608_CONFIG_REG, 1, &data, 1, 100);

    // ??????DLPF?? 99Hz
    data = 0x02; // DLPF_CFG=2
    HAL_I2C_Mem_Write(&hi2c2, (ICM20608_ADDR << 1), ICM20608_ACCEL_CONFIG2_REG, 1, &data, 1, 100);
}
void ICM20608_EnableDataReadyInterrupt(void) {
    uint8_t data = 0x01; // DATA_RDY_EN=1
    HAL_I2C_Mem_Write(&hi2c2, (ICM20608_ADDR << 1),
```

```
ICM20608_INT_ENABLE_REG, 1, &data, 1, 100);
}
void ICM20608_ReadRawData(int16_t *accel, int16_t *gyro) {
    uint8_t buffer[14];
    HAL_I2C_Mem_Read(&hi2c2, (ICM20608_ADDR << 1), ICM20608_ACCEL_MEAS, 0x00000001U, buffer, 14, 100);
        HAL_StatusTypeDef status = HAL_I2C_Mem_Read(&hi2c2, (ICM20608_ADDR << 1), ICM20608_ACCEL_MEAS, I2C_MEMADD_SIZE_8BIT, buffer, 14, 100);

        if (status != HAL_OK) {
          printf("I2C Read Error: %d\r\n", status);
          return;
    }

    // ???????????(?????)
    accel[0] = (buffer[0] << 8) | buffer[1];    // X?
    accel[1] = (buffer[2] << 8) | buffer[3];    // Y?
    accel[2] = (buffer[4] << 8) | buffer[5];    // Z?
    //printf("Accel X: %d, Y: %d, Z: %d\r\n", accel[0], accel[1], accel[2]);

    // ???????
    gyro[0] = (buffer[8]  << 8) | buffer[9];     // X?
    gyro[1] = (buffer[10] << 8) | buffer[11];     // Y?
    gyro[2] = (buffer[12] << 8) | buffer[13];     // Z?
}


// ?????????????
void CalculateAngles(float ax, float ay, float az, float *pitch, float *roll) {
```

```
    *pitch = atan2f(ay, sqrtf(ax * ax + az * az)) * 180.0f / M_PI;
    *roll = atan2f(-ax, sqrtf(ay * ay + az * az)) * 180.0f / M_PI;
}


float MeanFilter_Update(MeanFilter *filter, float new_val) {
    filter->buffer[filter->index] = new_val;
    filter->index = (filter->index + 1) % FILTER_SIZE;
    float sum = 0;
    for (int i = 0; i < FILTER_SIZE; i++) sum += filter->buffer[i];
    return sum / FILTER_SIZE;
}



float LPF_Update(LPF *filter, float input) {
    filter->prev_output = filter->alpha * input + (1 - filter->alpha) *
filter->prev_output;
    return filter->prev_output;
}

void LCD_Display(float pitch, float roll, float yaw, float ax, float ay, float az, float gx,
float gy, float gz) {
    char buffer[32];

    // ??
   // lcd_clear(WHITE);

    // ?????
    sprintf(buffer, "Ax: %.3f ", ax);
```

```
        lcd_show_string(10, 0,24,buffer);
        sprintf(buffer, "Ay: %.3f ", ay);
       lcd_show_string(10, 25,24,buffer);
        sprintf(buffer, "Az: %.3f ", az);
       lcd_show_string(10, 50,24,buffer);

        // ?????
        sprintf(buffer, "Gx: %.1f ", gx);
        lcd_show_string(10, 75,24,buffer);
       sprintf(buffer, "Gy: %.1f ", gy);
        lcd_show_string(10, 100,24,buffer);
       sprintf(buffer, "Gz: %.1f ", gz);
        lcd_show_string(10, 125,24,buffer);


        // ?????
        sprintf(buffer, "Pitch: %.1f", pitch);
        lcd_show_string(10, 150,24,buffer);
        sprintf(buffer, "Roll: %.1f", roll);
        lcd_show_string(10, 170,24,buffer);
        sprintf(buffer, "Yaw: %.1f", yaw);
        lcd_show_string(10, 190,24,buffer);

            lcd_show_string(10,210,24,"922106840127_I2CICM");
}



/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```
// fprintf()???
int fputc(int ch, FILE *f)
{
      HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 1000);
      return ch;
}



int main(void)
{
   HAL_Init();
   SystemClock_Config();
   MX_GPIO_Init();
   MX_FSMC_Init();
   MX_TIM14_Init();
     HAL_TIM_Base_Start_IT(&htim14);
     HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);
     drv_lcd_init();
   MX_USART1_UART_Init();
   MX_I2C2_Init();
     ICM20608_Init();


     lcd_clear(WHITE);



          // ?????????????0,???????
   for (int i = 0; i < FILTER_SIZE; i++) {
         accel_filter_x.buffer[i] = 0.0f;
```

```
        accel_filter_y.buffer[i] = 0.0f;
        accel_filter_z.buffer[i] = 0.0f;
}

accel_filter_x.index = 0;
accel_filter_y.index = 0;
accel_filter_z.index = 0;

          // ???????????????
lpf_ax.alpha = 0.1f; lpf_ax.prev_output = 0.0f;
lpf_ay.alpha = 0.1f; lpf_ay.prev_output = 0.0f;
lpf_az.alpha = 0.1f; lpf_az.prev_output = 0.0f;

   // ????????
float pitch, roll, yaw = 0.0f;



while (1)
{
        // ?????????
   ICM20608_ReadRawData(accel, gyro);

        // ??????
   // ??????????????,???????? ±2g,? 1g = 9.81 m/s²
   float ax = accel[0] * 9.81f / ACCEL_SENSITIVITY;
   float ay = accel[1] * 9.81f / ACCEL_SENSITIVITY;
   float az = accel[2] * 9.81f / ACCEL_SENSITIVITY;



   // ????
```

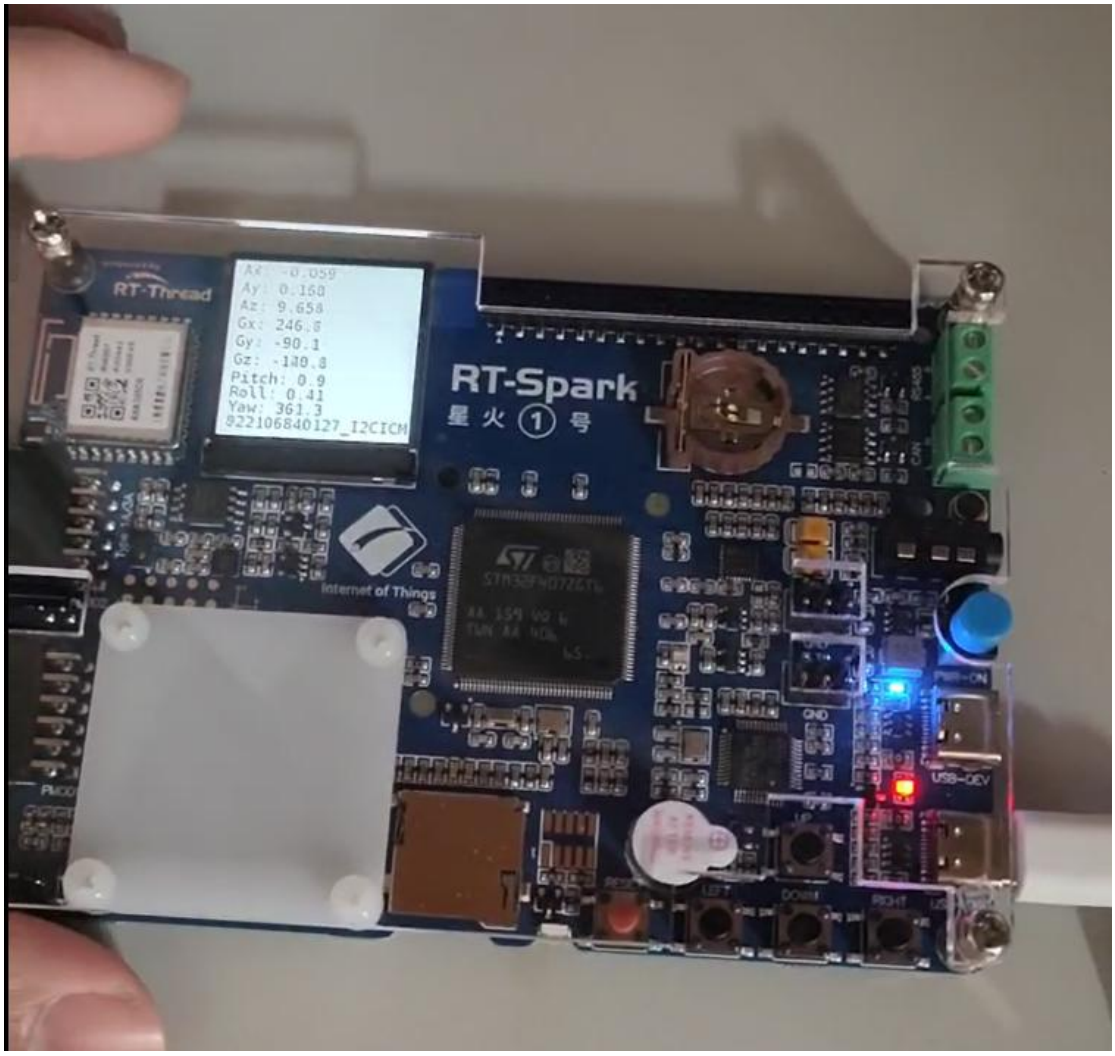
```
        // ?????????:?????,?????
    ax_filtered = LPF_Update(&lpf_ax, MeanFilter_Update(&accel_filter_x, ax));
    ay_filtered = LPF_Update(&lpf_ay, MeanFilter_Update(&accel_filter_y, ay));
    az_filtered = LPF_Update(&lpf_az, MeanFilter_Update(&accel_filter_z, az));



         // ?????
        CalculateAngles(ax_filtered, ay_filtered, az_filtered, &pitch, &roll);



        // ??????????????(??:?/?)
    float gx = gyro[0] * GYRO_SENSITIVITY;
    float gy = gyro[1] * GYRO_SENSITIVITY;
    float gz = gyro[2] * GYRO_SENSITIVITY;

        // ????Z???????????
    float gz_filtered = LPF_Update(&lpf_gyro_z, gz);
        float gx_filtered = LPF_Update(&lpf_gyro_x, gx);
        float gy_filtered = LPF_Update(&lpf_gyro_y, gy);

      // ???????:?????Z????(?????? dt = 10ms)
      yaw += gz_filtered * 0.01f;

           printf("Yaw: %.2f\r\n", yaw);    // ??????

           // ???LCD
       LCD_Display(pitch, roll, yaw, ax_filtered, ay_filtered, az_filtered, gx_filtered,
gy_filtered, gz_filtered);

           HAL_Delay(50); // ?????
```

```
    }
}

/**
    * @brief System Clock Configuration
    * @retval None
    */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_S
CALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 168;
```

```
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType                                    =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if  (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,  FLASH_LATENCY_5)  !=
HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
```

```
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

3 实验结果分析与总结



通过实验结果的 LCD 可以看出，我已经完成了本实验的基本要求，并且添加了个人学号信息标识，具体实验结果可以看视频。