一、设计需求

上板

无软件设计。

下板

- **主控芯片：** STM32G030。
- **数据采集：** 通过 SPI 接口驱动 AD7606-4 模数转换器，采集上板输出的四路电压信号。
- **数据处理：**
    1. 根据采集的电压值计算 S5990 各通道的输出电流。
    2. 依据 S5990 器件手册提供的公式，计算光斑在敏感面上的坐标 (X, Y)。
- **数据输出：** 通过 I2C 接口将计算得到的光斑坐标数据传输至主控板。

二、设计思路

上板

无软件设计。

下板

1. AD7606-4 SPI 驱动

- **硬件连接：**
    - **SPI 通信 (硬件 SPI1):**

STM32G030 引脚	AD7606 引脚	功能
PA1 (SPI1_SCK)	SCLK	SPI 时钟
PA6 (SPI1_MISO)	DB7/DOUTA	SPI 数据线

    - **控制信号 (GPIO):**

STM32G030 引脚	AD7606 引脚	功能
PA4 (AD7606_RESET)	RESET	芯片复位
PA5 (AD7606_BUSY)	BUSY	转换状态指示
PB0 (AD7606_CONVST)	CONVSTA & CONVSTB	转换启动信号 (双通道共用)

    - **配置引脚状态：**

AD7606 引脚	连接电平	功能说明
OS0	GND	配置为无过采样模式
OS1	GND	
OS2	GND	
BYTESEL	3V3	选择 SPI 字节传输模式
V1	-X1	模拟输入通道 (对应 S5990)
V2	-X2	
V3	-Y1	
V4	-Y2	

- **软件驱动实现：**
  - **启动转换 (`AD7606_StartConv`):**

```
void AD7606_StartConv(void)
{
    /* 在CONVST引脚产生上升沿启动转换，低电平持续时间需 >25ns */
    AD_CONVST_LOW();  // 拉低
    AD_CONVST_LOW();  // 短暂保持低电平 (连续执行约50ns)
    AD_CONVST_LOW();
    AD_CONVST_HIGH(); // 上升沿触发转换
}
```

  - **读取转换数据 (`AD7606_Scan`, `AD7606_ReadAdc`):**

```
uint8_t AD7606_Scan(void)
{
    ...
    for (i = 0; i < CH_NUM; i++) // CH_NUM = 4 (对应V1-V4)
    {
        uint8_t rx[2] = {0};
        g_obs = HAL_SPI_Receive(&hspi1, rx, 2, HAL_MAX_DELAY); // 使用SPI读取2字节(16
位)数据
        s_adc_now[i] = (rx[0] << 8) | (rx[1]); // 组合高8位和低8位
    }
    ...
}
int16_t AD7606_ReadAdc(uint8_t _ch)
{
    return s_adc_now[_ch]; // 返回指定通道的ADC原始值
}
```

  - **数据处理与坐标计算 (`AD7606_Mak`):**

```
void AD7606_Mak(void)
{
```

```
    // 1. 读取四通道ADC原始值
    for (uint8_t i = 0; i < CH_NUM; i++) {
        s_dat[i] = AD7606_ReadAdc(i);
    }
    // 2. 检查光照有效性 (基于阈值)
    uint8_t LightThreshold = I2C_Slave_GetRegister(PSD_LIGHT_THRESHOLD);
    if ((abs(s_dat[0]) < LightThreshold) &&
        (abs(s_dat[1]) < LightThreshold) &&
        (abs(s_dat[2]) < LightThreshold) &&
        (abs(s_dat[3]) < LightThreshold)) {
        // 所有通道光强均低于阈值，判定为无效光斑
        lpf_x.prev = 0;
        lpf_y.prev = 0;
        coordinate_x = 0;
        coordinate_y = 0;
        return;
    }
    // 3. 计算总电流 (近似于ADC总和)
    int32_t sum = s_dat[0] + s_dat[1] + s_dat[2] + s_dat[3];
    // 4. 计算坐标分量差值
    int32_t temp1 = s_dat[1] + s_dat[2] - (s_dat[0] + s_dat[3]); // (X2 + Y1) - (X1
+ Y2)
    int32_t temp2 = s_dat[1] + s_dat[3] - (s_dat[0] + s_dat[2]); // (X2 + Y2) - (X1
+ Y1)
    // 5. 应用S5990公式计算原始坐标 (PSD_L为敏感面尺寸参数)
    float raw_x = temp1 * PSD_L / 2.0f / sum;
    float raw_y = temp2 * PSD_L / 2.0f / sum;
    // 6. 应用低通滤波(LPF)并更新坐标
    coordinate_x = LPF_Update(&lpf_x, raw_x);
    coordinate_y = LPF_Update(&lpf_y, raw_y);
}
```

2. I2C 从机设计

- **实现逻辑：**
  - 主要代码位于 `bsp_i2c_slave.c/.h`。
  - 基于中断机制处理 I2C 通信事件。
  - 主机访问方式与常见传感器 (如 MPU6050) 类似。
- **寄存器映射 (共 256 字节)：**
  - **参数配置区 (基地址: `0x00`):**

寄存器属性	地址偏移	大小(字节)	说明
`PSD_SETTING`	0x00	1	配置区基地址
`PSD_WHO_AM_I`	0x01	1	设备标识寄存器
`PSD_LIGHT_THRESHOLD`	0x02	1	光照强度判定阈值

  - **坐标数据区 (基地址: `0x80`):**

寄存器属性	地址偏移	大小(字节)	数据类型	说明
`PSD_COORDINATE_X`	0x80	4	`float`	滤波后的 X 坐标
`PSD_COORDINATE_Y`	0x84	4	`float`	滤波后的 Y 坐标

- **寄存器操作：**
  - **主机写操作：** 用于配置设备参数（如设置 `PSD_LIGHT_THRESHOLD`）。支持通过无线通信间接配置从机，方便调试。
  - **主机读操作：** 在 `AD7606_Mak()` 完成坐标计算和滤波后，结果立即更新至坐标寄存器（`PSD_COORDINATE_X`, `PSD_COORDINATE_Y`），供主机随时读取。
  - **坐标写入实现 (`I2C_Slave_SetCoordinateRegister`):**

```
void I2C_Slave_SetCoordinateRegister(uint16_t adr, float val)
{
    uint32_t val_bits;
    memcpy(&val_bits, &val, sizeof(float)); // 将float按位复制到uint32_t
    if (adr == PSD_COORDINATE_X) {
        // 将32位浮点数的二进制表示按字节写入寄存器
        I2C_Slave_SetRegister(PSD_COORDINATE_X + 0, (uint8_t)(val_bits >> 24)); //
MSB
        I2C_Slave_SetRegister(PSD_COORDINATE_X + 1, (uint8_t)(val_bits >> 16));
        I2C_Slave_SetRegister(PSD_COORDINATE_X + 2, (uint8_t)(val_bits >> 8));
        I2C_Slave_SetRegister(PSD_COORDINATE_X + 3, (uint8_t)(val_bits));        //
LSB
    } else if (adr == PSD_COORDINATE_Y) {
        I2C_Slave_SetRegister(PSD_COORDINATE_Y + 0, (uint8_t)(val_bits >> 24));
        I2C_Slave_SetRegister(PSD_COORDINATE_Y + 1, (uint8_t)(val_bits >> 16));
        I2C_Slave_SetRegister(PSD_COORDINATE_Y + 2, (uint8_t)(val_bits >> 8));
        I2C_Slave_SetRegister(PSD_COORDINATE_Y + 3, (uint8_t)(val_bits));
    }
}
```