CONNTEK
昆泰芯微电子

智能感知世界，传递美好生活

AN-KTH5791-002 | Application Note

# KTH5791 UART 参考示例程序说明

## KTH5791 UART Reference Example Guide

KTH5791 磁编码器芯片 UART 通信接口驱动开发指南

**适用产品：** KTH5791

**文档版本：** Rev 1.0 | **发布日期：** 2026.06.10

**昆泰芯微电子科技有限公司**

CONNTEK Microelectronics Co., Ltd.

www.conntek.com.cn

Technical Support:

support@conntek.com.cn

# Contents

# 1 概述
Overview

KTH5791 是一款**磁编码器芯片**，主要用于鼠标滚轮等旋转位置检测应用。芯片通过内置磁传感器检测磁场方向变化，经内部算法处理后，提供以下输出方式：

Table 1: KTH5791 输出方式

| 输出方式 | 说明 |
|---|---|
| UART | 支持读写寄存器、配置参数、读取角度/磁场数据 |
| AB 相 | 增量编码器信号，每齿半个跳变周期 |

本文档提供 MCU 通过 **UART 接口**与 KTH5791 芯片通信的完整参考示例程序，包含通信协议说明、寄存器表、API 函数说明以及使用示例。

## 1.1 参考程序文件说明

Table 2: 参考程序文件清单

| 文件 | 说明 |
|---|---|
| `kth5791_uart_example.h` | 头文件：寄存器地址宏定义、命令码、位域定义、数据结构、API 声明 |
| `kth5791_uart_example.c` | 源文件：完整的驱动实现和平台移植示例 |

## 1.2 芯片引脚说明

KTH5791 芯片对外提供以下功能引脚：

**通信引脚**

Table 3: 通信引脚

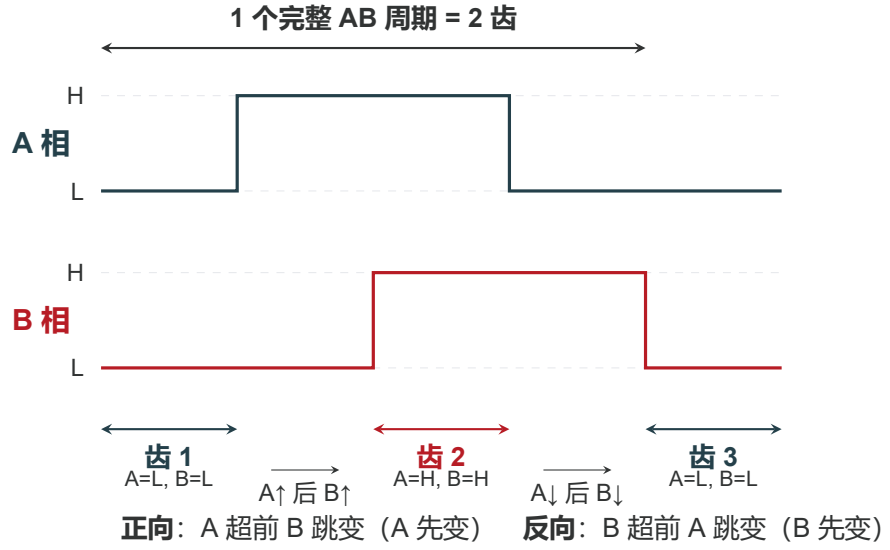
| 引脚功能 | 方向 | 说明 |
|---|---|---|
| UART TX | 输出 | 芯片 UART 发送数据 |
| UART RX | 输入 | 芯片 UART 接收数据 |

### 输出引脚

Table 4: 输出引脚

| 引脚功能 | 方向 | 说明 |
|---|---|---|
| AB 相 A 通道 | 输出 | 增量编码器 A 相输出 |
| AB 相 B 通道 | 输出 | 增量编码器 B 相输出 |

AB 相输出为增量编码器信号，A/B 两相相位差 90°。**齿是 A、B 同为高或同为低的稳态区间**；齿间 A、B 先后跳变形成过渡，跳变顺序决定旋转方向。两齿构成一个完整 AB 周期。




### 电源引脚

Table 5: 电源引脚

| 引脚功能 | 说明 |
|---|---|
| VCC | 供电电源（3.3V / 5V） |
| GND | 电源地 |

# 2 UART 通信协议
UART Communication Protocol

## 2.1 通信参数

Table 6: UART 通信参数

| 参数 | 值 |
|---|---|
| 波特率 | **921600 bps** |
| 数据位 | 8 |
| 停止位 | 1 |
| 校验位 | 无 |
| 硬件流控 | 无 |
| 包长度 | **6 字节（固定）** |
| 字节序 | 大端（高字节在前） |

## 2.2 命令包格式（MCU → KTH5791）

Table 7: 命令包格式（MCU → KTH5791，固定 6 字节）

| 位置 | 字段 | 说明 |
|---|---|---|
| Byte[0] | CMD | 命令码 |
| Byte[1] | ARG1 | 参数 1 |
| Byte[2] | ARG2 | 参数 2 |
| Byte[3] | ARG3 | 参数 3 |
| Byte[4] | CRC H | CRC 高字节 |
| Byte[5] | CRC L | CRC 低字节 |

**CRC 计算方法**：$\text{CRC} = \text{Byte}[0] + \text{Byte}[1] + \text{Byte}[2] + \text{Byte}[3]$（前 4 字节累加和，16 位无符号数）

### 2.3 响应包格式（KTH5791 → MCU）

Table 8: 响应包格式（KTH5791 → MCU，固定 6 字节）

| 位置 | 字段 | 说明 |
|---|---|---|
| Byte[0] | 0xAA | 响应头（固定值） |
| Byte[1] | Addr/Cmd | 寄存器地址或命令码 |
| Byte[2] | Data H | 数据高字节 |
| Byte[3] | Data L | 数据低字节 |
| Byte[4] | CRC H | CRC 高字节 |
| Byte[5] | CRC L | CRC 低字节 |

**磁场数据响应包**（8 字节，命令 0x5E 专用）：

Table 9: 磁场数据响应包格式（8 字节）

| 位置 | 字段 | 说明 |
|---|---|---|
| Byte[0] | 0xAA | 响应头（固定值） |
| Byte[1] | 0x29 | PlaneOrXMag 寄存器地址 |
| Byte[2] | Data1 H | 平面磁场 (在轴) / X 磁场 (离轴) 高字节 |
| Byte[3] | Data1 L | 平面磁场 (在轴) / X 磁场 (离轴) 低字节 |
| Byte[4] | Data2 H | Y 磁场高字节（离轴），在轴时值不变 |
| Byte[5] | Data2 L | Y 磁场低字节（离轴），在轴时值不变 |
| Byte[6] | CRC H | CRC 高字节 |
| Byte[7] | CRC L | CRC 低字节 |

磁场数据 CRC 算法：$\text{CRC} = \text{Byte}[0] + \text{Byte}[1] + \cdots + \text{Byte}[5]$（前 6 字节累加和）

## 2.4 命令码汇总

Table 10: 命令码列表

| 命令码 | 名称 | 参数 | 说明 |
|---|---|---|---|
| 0x55 | WriteRegister | ARG1= 地址, ARG2= 数据 H, ARG3= 数据 L | 写寄存器 |
| 0x56 | ReadRegister | ARG1= 地址, ARG2=0, ARG3=0 | 读寄存器 |
| 0x57 | CalibrateStart | 无参数 | 开始校准 |
| 0x58 | CalibrateStop | 无参数 | 停止校准 |
| 0x5A | OutputEnable | 无参数 | 输出使能 |
| 0x5B | OutputDisable | 无参数 | 输出禁用 |
| 0x5E | GetMagneticField | 无参数 | 获取磁场数据（返回 8 字节包） |

## 2.5 CRC 错误处理

当 KTH5791 收到的命令包 CRC 校验失败时，将原样返回收到的 6 字节数据，并将 CRC 字段 (Byte[4:5]) 替换为 `0xFFFF`。主程序可以通过检查响应包的 CRC 字段是否为 `0xFFFF` 来判断 CRC 错误。

**关键要点**

**协议要点:**

- 所有命令包和响应包均为固定 6 字节（磁场数据包为 8 字节）
- CRC = 前 N 字节累加和（普通包前 4 字节，磁场包前 6 字节），16 位无符号
- CRC 错误时响应包 CRC 字段为 `0xFFFF`
- 字节序为大端（高字节在前）

# 3 寄存器表
Register Map

KTH5791 寄存器为 16 位宽，分为三个地址区域。

**属性说明**:

- R = 只读
- R/W = 可读写
- Prog = 可编程（可通过 Password 寄存器保存到 MTP）

## 3.1 寄存器总览

Table 11: 寄存器总览表

| 地址 | 寄存器名称 | 简称 | 属性 | 说明 |
|---|---|---|---|---|
| 0x02 | Status | — | R | 芯片状态寄存器（**禁止写入**） |
| 0x03 | Config | — | R/W, Prog | 配置寄存器 1 |
| 0x04 | Config2 | — | R/W, Prog | 配置寄存器 2 |
| 0x05 | Password | — | R/W | 编程密码寄存器 |
| 0x06 | Config3 | — | R/W, Prog | 配置寄存器 3 |
| 0x07 | Config4 | — | R/W, Prog | 配置寄存器 4 |
| 0x21 | FirstGain | FG | R/W, Prog | 一级增益校准值 |
| 0x22 | FirstXOffset | FX | R/W, Prog | 一级 X 轴偏移校准值 |
| 0x23 | FirstYOffset | FY | R/W, Prog | 一级 Y 轴偏移校准值 |
| 0x24 | SecondGain | SG | R/W, Prog | 二级增益校准值 |
| 0x25 | SecondXOffset | SX | R/W, Prog | 二级 X 轴偏移校准值 |
| 0x26 | SecondYOffset | SY | R/W, Prog | 二级 Y 轴偏移校准值 |
| 0x27 | AngleOffset | AO | R | 角度偏移值（**禁止写入**） |
| 0x28 | AngleOut | OR | R | 角度输出值（**禁止写入**） |
| 0x29 | PlaneOrXMag | — | R | 平面磁场（在轴）/ X 轴磁场（离轴），**禁止写入** |
| 0x2A | YMag | — | R | Y 轴磁场（离轴），在轴时值不变。**禁止写入** |

## 3.2 寄存器详细位定义

### 寄存器 0x02: Status（芯片状态寄存器）

Table 12: Status 寄存器位定义

| 位 | 名称 | 说明 |
|---|---|---|
| [1] | Output | 采样输出运行中。输出使能后置 1（传感器持续采样），停止输出后清 0 |
| [2] | Calibrating | 校准运行中。开始校准后置 1，校准完成或停止后清 0 |
| [14] | MagError | 磁场异常。芯片上电后采集磁场基准值，此后磁场偏离基准值过大时置 1，此时不触发输出 |
| [15] | NoMag | 无磁。未检测到磁铁时置 1，此时不触发输出 |

**说明**：此寄存器**只读**，禁止写入。磁铁正常且磁场稳定时 [15:14] 同时清 0。

### 寄存器 0x03: Config（配置寄存器 1）

Table 13: Config 寄存器位定义

| 位 | 名称 | 说明 | 取值范围 |
|---|---|---|---|
| [6:0] | IntervalCount | 齿数-1，每个齿对应一次输出触发 | 0～127（对应 1～128 齿） |
| [7] | OutReverse | 输出方向反转 | 0= 正向，1= 反向 |
| [8] | OutputMode | 输出方式选择 | 0=UART 串口输出，1=AB 相输出 |
| [9] | IsOnAxis | 在轴/离轴模式 | 0= 离轴模式（磁铁径向安装），1= 在轴模式（磁铁轴向安装） |
| [10] | VirtualAB | 虚拟 AB 模式 | 0= 禁用，1= 使能（高速时 AB 分辨率翻倍） |
| [13:11] | WakeUpTh | 睡眠唤醒磁场阈值 | 0～7，实际阈值 = 5 × (值 +1)，范围 5～40 |
| [15:14] | PlaneSelect | 磁场平面选择 | 0=XY，1=YZ，2=XZ，3=XY2 |

**默认值**：芯片上电后从 MTP 加载，若未编程则为 0x0B17。

### 寄存器 0x04: Config2（配置寄存器 2）

Table 14: Config2 寄存器位定义

| 位 | 名称 | 说明 | 取值范围 |
|---|---|---|---|
| [6:0] | EnterSleepTime | 空闲后进入睡眠的时间 | 0～127（寄存器值 +1 秒，即 1～128 秒） |
| [7] | IsSleep | 睡眠功能使能 | 0= 禁用睡眠，1= 启用睡眠 |
| [13:8] | SleepMeasureTime | 睡眠期间测量间隔 | 0～63，1 Code=20ms |
| [15:14] | DeathRange | 死区范围 | 0～3，死区 =(值 +1)×10%× 齿间隔角 |

**死区说明：**死区以每个齿中心（卡位点）为基准，向两侧各展开 deathRange × 齿间隔角。当角度落在齿中心两侧的死区内、且仅移动 1 格时，不触发输出，用于滤除卡位点附近的微小抖动。

以齿数 =24（齿间隔角 =15°）、死区 =1（10%）为例：


各死区档位（齿数 =24 时）：

- 死区 =1：齿中心两侧各 1.5°（±10% 齿间隔角）不触发
- 死区 =2：齿中心两侧各 3.0°（±20% 齿间隔角）不触发
- 死区 =3：齿中心两侧各 4.5°（±30% 齿间隔角）不触发
- 死区 =4：齿中心两侧各 6.0°（±40% 齿间隔角）不触发

**默认值**：芯片上电后从 MTP 加载，若未编程则为 0x9F83。

### 寄存器 0x05: Password（编程密码寄存器）

Table 15: Password 寄存器位定义

| 位 | 名称 | 说明 |
|---|---|---|
| [14:0] | password | 编程密码值（正确密码为 0x1234） |
| [15] | program | 编程触发位：此位写 1 且 [14:0] 密码正确时，自动将 Config～Config4 和校准参数保存到内部 MTP |

**使用方式**：

1. 先配置好所有需要保存的寄存器（Config、Config2～4、校准参数等）
2. 向 Password 寄存器写入（`0x8000 | 0x1234`）即 `0x9234`
3. 芯片自动执行编程，完成后自动恢复
4. 建议编程后等待 20ms 确保完成

### 寄存器 0x06: Config3（配置寄存器 3）

Table 16: Config3 寄存器位定义

| 位 | 名称 | 说明 | 取值范围 |
|---|---|---|---|
| [1:0] | SampleFreq | 磁传感器采样频率档位 | 0～3（对应不同的内部采样速率） |
| [9:2] | FilterTime | 反转过滤窗口时间（高速 → 低速时，窗口内反转信号被过滤） | 0～255 |
| [15:10] | HighSpeedJudge | 高速判定 RPM 阈值 | 0～63（寄存器值 +1）×100 RPM，即 100～6400 RPM |

**RPM 阈值说明**：当转速超过此阈值时，芯片切换到高速模式，采用不同的输出策略以保证响应速度。

**默认值**：芯片上电后从 MTP 加载，若未编程则为 0x1654。

### 寄存器 0x07: Config4（配置寄存器 4）

Table 17: Config4 寄存器位定义

| 位 | 名称 | 说明 |
|---|---|---|
| [0] | FilterEn | 反转过滤使能（0= 禁用，1= 启用）。启用后，芯片从高速降至低速时根据 FilterTime 开启过滤窗口，窗口内反转信号被过滤 |
| [1] | WakeABDelay | 唤醒 AB 延迟使能（从睡眠唤醒时 AB 输出延迟） |
| [15:2] | — | 保留 |

**默认值**：芯片上电后从 MTP 加载，若未编程则为 0x0000。

### 寄存器 0x21～0x26: 校准参数寄存器

Table 18: 校准参数寄存器

| 地址 | 名称 | 位宽 | 说明 |
|---|---|---|---|
| 0x21 | FirstGain (FG) | [15:0] | 一级增益校准值 |
| 0x22 | FirstXOffset (FX) | [15:0] | 一级 X 轴偏移校准值 |
| 0x23 | FirstYOffset (FY) | [15:0] | 一级 Y 轴偏移校准值 |
| 0x24 | SecondGain (SG) | [15:0] | 二级增益校准值 |
| 0x25 | SecondXOffset (SX) | [15:0] | 二级 X 轴偏移校准值 |
| 0x26 | SecondYOffset (SY) | [15:0] | 二级 Y 轴偏移校准值 |

**说明**：这些校准值由芯片校准流程自动生成，用户一般无需手动修改。通过 CalibrateStart 命令启动校准，芯片会自动计算并填充这些值。

### 寄存器 0x27～0x2A: 角度输出与磁场数据寄存器（只读）

Table 19: 角度输出与磁场数据寄存器总览

| 地址 | 名称 | 属性 | 说明 |
|---|---|---|---|
| 0x27 | AngleOffset (AO) | R | 角度偏移值（0～65535），最终输出角度 = 原始角度 + 此偏移。**禁止写入** |
| 0x28 | AngleOut (OR) | R | 当前角度输出值（0～65535 对应 0°～360°）。角度计算公式：角度$(^\circ) = \frac{\text{AngleOut}}{65536} \times 360$ |
| 0x29 | PlaneOrXMag | R | 在轴模式 (IsOnAxis=1)：输出平面合成磁场；离轴模式 (IsOnAxis=0)：输出 X 轴磁场分量。**禁止写入** |
| 0x2A | YMag | R | 在轴模式 (IsOnAxis=1)：值不变化；离轴模式 (IsOnAxis=0)：输出 Y 轴磁场分量。**禁止写入** |

**注意：**地址 0x27～0x2A 均为**只读**寄存器，禁止写入。尝试写入这些地址将被芯片忽略。角度和磁场数据仅在输出使能（OutputEnable）后才会被持续更新。

# 4 端口移植指南
Port Migration Guide

参考示例程序采用分层设计，用户只需在自己的 MCU 平台上实现以下 **4 个**移植函数：

**设计提示**

**设计提示：**移植工作只需实现 4 个平台相关的 UART 和延时函数，所有 KTH5791 协议和寄存器操作逻辑已在参考程序中完整实现，无需修改。

## 4.1 需要实现的移植函数

```c
/**
 * @brief 初始化 UART 外设
 * @note  配置：921600bps，8数据位，1停止位，无校验，无流控
 */
void Kth5791_UART_Init(void);

/**
 * @brief 通过 UART 发送数据
 * @param pData 数据指针
 * @param size  数据长度（固定 6 字节）
 */
void Kth5791_UART_Send(uint8_t *pData, uint16_t size);

/**
 * @brief 通过 UART 接收数据
 * @param pData      接收缓冲区指针
 * @param size       期望接收长度（固定 6 字节）
 * @param timeout_ms 超时时间（ms），建议 100ms
 * @return 实际接收到的字节数
 */
uint16_t Kth5791_UART_Recv(uint8_t *pData, uint16_t size,
                           uint32_t timeout_ms);

/**
 * @brief 毫秒级延时
 * @param ms 延时毫秒数
 */
void Kth5791_DelayMs(uint32_t ms);
```

### 4.2 STM32 HAL 库移植示例

```c
/* UART 句柄（全局）*/
UART_HandleTypeDef huart1;

void Kth5791_UART_Init(void)
{
    huart1.Instance          = USART1;
    huart1.Init.BaudRate     = 921600;
    huart1.Init.WordLength   = UART_WORDLENGTH_8B;
    huart1.Init.StopBits     = UART_STOPBITS_1;
    huart1.Init.Parity       = UART_PARITY_NONE;
    huart1.Init.Mode         = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    HAL_UART_Init(&huart1);
}

void Kth5791_UART_Send(uint8_t *pData, uint16_t size)
{
    HAL_UART_Transmit(&huart1, pData, size, 100);
}

uint16_t Kth5791_UART_Recv(uint8_t *pData, uint16_t size,
                           uint32_t timeout_ms)
{
    if (HAL_UART_Receive(&huart1, pData, size, timeout_ms) == HAL_OK)
        return size;
    else
        return 0;
}

void Kth5791_DelayMs(uint32_t ms)
{
    HAL_Delay(ms);
}
```

### 4.3 通用平台（基于 SysTick）移植示例

```c
static volatile uint32_t g_tick_ms = 0;

/* 在定时器/SysTick 中断服务函数中调用 */
void SysTick_Handler(void)
{
    g_tick_ms++;
```

```
}

void Kth5791_DelayMs(uint32_t ms)
{
    uint32_t start = g_tick_ms;
    while ((g_tick_ms - start) < ms);
}

void Kth5791_UART_Init(void)
{
    /* TODO: 根据具体 MCU 实现:
     * 1. 使能 UART 时钟
     * 2. 配置 GPIO (TX=推挽输出/复用, RX=浮空输入/复用)
     * 3. 配置 UART 寄存器: 921600, 8N1
     * 4. 使能 UART 模块
     */
}

void Kth5791_UART_Send(uint8_t *pData, uint16_t size)
{
    for (uint16_t i = 0; i < size; i++)
    {
        while (!(USART1->SR & USART_SR_TXE));  /* 等待发送缓冲区空 */
        USART1->DR = pData[i];
    }
    while (!(USART1->SR & USART_SR_TC));       /* 等待发送完成 */
}

uint16_t Kth5791_UART_Recv(uint8_t *pData, uint16_t size,
                           uint32_t timeout_ms)
{
    uint32_t start = g_tick_ms;
    uint16_t count = 0;
    while (count < size)
    {
        if (USART1->SR & USART_SR_RXNE)
            pData[count++] = (uint8_t)USART1->DR;
        if (timeout_ms > 0 && (g_tick_ms - start) >= timeout_ms)
            break;
    }
    return count;
}
```

# 5 API 函数参考
API Function Reference

## 5.1 基础通信函数

Table 20: 基础通信函数

| 函数 | 说明 |
|---|---|
| `Kth5791_CalcCRC()` | 计算 6 字节包的 CRC（前 4 字节累加和） |
| `Kth5791_CalcMagCRC()` | 计算 8 字节磁场包的 CRC（前 6 字节累加和） |
| `Kth5791_SendCommand()` | 构建并发送命令包（不等待响应） |
| `Kth5791_SendAndRecv()` | 发送命令并阻塞等待响应 |

## 5.2 寄存器读写

Table 21: 寄存器读写函数

| 函数 | 说明 | 返回 |
|---|---|---|
| `Kth5791_ReadRegister(addr, pValue)` | 读取单个寄存器 | 0= 成功 |
| `Kth5791_WriteRegister(addr, value)` | 写入单个寄存器 | 0= 成功 |
| `Kth5791_ModifyRegister(addr, mask, value)` | 读-修改-写指定寄存器位域 | 0= 成功 |

## 5.3 数据读取

Table 22: 数据读取函数

| 函数 | 说明 |
|---|---|
| `Kth5791_Init()` | 初始化：验证通信连接 |
| `Kth5791_ReadAngle()` | 读取当前角度值（0~65535）。**必须在输出使能后调用** |
| `Kth5791_ReadMagneticField()` | 读取 X/Y 磁场数据。**必须在输出使能后调用，否则寄存器值未更新** |

**重要：**角度（0x28）和磁场（0x29/0x2A）寄存器仅在输出使能（OutputEnable）后才会被持续更新。未使能输出时，这些寄存器保持的是旧值或初始值。

## 5.4 芯片配置

Table 23: 芯片配置函数

| 函数 | 参数 | 说明 |
|---|---|---|
| `Kth5791_SetToothCount(n)` | n: 1～128 | 配置齿数（每圈输出触发数） |
| `Kth5791_SetAxisMode(isOnAxis)` | 0= 离轴, 1= 在轴 | 配置在轴/离轴模式 |
| `Kth5791_SetPlaneSelect(plane)` | 0=XY, 1=YZ, 2=XZ, 3=XY2 | 配置磁场平面 |
| `Kth5791_SetOutputReverse(rev)` | 0= 正向, 1= 反向 | 配置输出方向 |
| `Kth5791_SetVirtualAB(enable)` | 0= 禁用, 1= 使能 | 虚拟 AB 模式（高速时分辨率翻倍） |
| `Kth5791_SetWakeUpThreshold(level)` | 0～7 | 睡眠唤醒磁场阈值（阈值 =5×(level+1)） |
| `Kth5791_SetSleepConfig(...)` | — | 配置睡眠参数 |
| `Kth5791_SetDeathRange(range)` | 1～4 | 配置死区范围，死区 =range×10%× 齿间隔角 |
| `Kth5791_SetSampleFilterConfig(...)` | — | 配置采样频率和反转过滤参数 |
| `Kth5791_SetFilterConfig(...)` | — | 配置反转过滤功能（高速 → 低速时过滤反转信号） |

## 5.5 操作命令

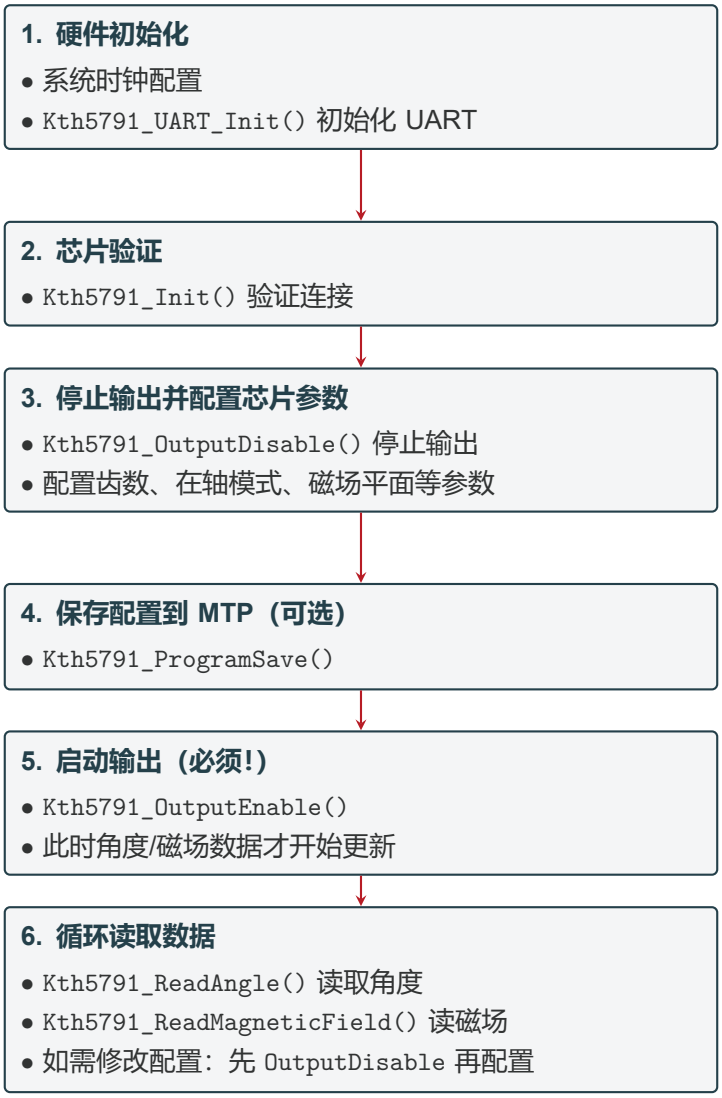
Table 24: 操作命令函数

| 函数 | 说明 |
|---|---|
| `Kth5791_OutputEnable()` | 启动输出，开始持续采样。**读取角度/磁场前必须先调用** |
| `Kth5791_OutputDisable()` | 停止输出。修改配置参数前建议先停止输出 |
| `Kth5791_CalibrateStart()` | 开始自动校准。**必须在输出停止状态下调用** |
| `Kth5791_CalibrateStop()` | 停止校准 |
| `Kth5791_ProgramSave()` | 将配置保存到芯片 MTP（需正确密码） |

**注意：**校准期间不能写入寄存器，写操作会被忽略。

# 6 典型使用流程
Typical Usage Flow

## 6.1 基本使用流程




## 6.2 完整代码示例

```c
#include "kth5791_uart_example.h"
#include <stdio.h>

int main(void)
{
    int ret;
    uint16_t angle, xMag, yMag;
    uint8_t toothCount;

```

```
    /* 1. 初始化硬件 */
    Kth5791_UART_Init();

    /* 2. 验证通信 */
    ret = Kth5791_Init();
    if (ret != 0) {
        printf("KTH5791 init failed: %d\r\n", ret);
        while (1);
    }
    printf("KTH5791 OK\r\n");

    /* 3. 配置芯片（芯片上电默认处于输出状态，需先停止输出） */
    Kth5791_OutputDisable();
    Kth5791_SetToothCount(24);              // 24格鼠标滚轮
    Kth5791_SetAxisMode(1);                 // 在轴模式
    Kth5791_SetPlaneSelect(PLANE_SELECT_XY);
    Kth5791_SetOutputReverse(0);            // 正向
    Kth5791_SetDeathRange(2);               // 死区=2，即±20%齿间隔角
    Kth5791_SetSampleFilterConfig(0, 10, 500);
    Kth5791_SetFilterConfig(0, 0);          // 反转过滤（通常无需启用）

    /* 4. 验证配置 */
    Kth5791_GetToothCount(&toothCount);
    printf("ToothCount=%d\r\n", toothCount);

    /* 5. 保存配置（写入后断电不丢失） */
    Kth5791_ProgramSave();
    printf("Config saved.\r\n");

    /* 6. 启动输出（必须！数据从此时开始更新） */
    Kth5791_OutputEnable();

    /* 7. 主循环 */
    while (1)
    {
        Kth5791_ReadAngle(&angle);
        printf("Angle=%u (%.2f deg)\r\n", angle,
               (float)angle * 360.0f / 65536.0f);
        Kth5791_DelayMs(50);
    }
}
```

### 6.3 校准流程

离轴结构或磁铁位置变更后需要进行校准，在轴结构无需校准。**校准必须在输出停止状态下进行**，校准由芯片自动完成：

1. 调用 `CalibrateStart` 启动校准，芯片开始持续采样
2. 旋转磁铁，芯片自动采集校准数据
3. 校准完成后芯片自动将参数写入对应寄存器（0x21~0x26），并退出校准（`Status[2]` 清 0）
4. 如中途需放弃，可调用 `CalibrateStop` 强制退出

```
/* 如果当前输出已使能，先停止 */
Kth5791_OutputDisable();

/* 校准流程 */
Kth5791_CalibrateStart();                  // 启动校准（期间不能写入寄存器）
/* 旋转磁铁，芯片自动采集数据并完成校准 */
while (1) {
    uint16_t status;
    Kth5791_ReadRegister(KTH5791_REG_STATUS, &status);
    if (!(status & STATUS_CALIBRATING_MASK))  // 校准自动完成
        break;
    Kth5791_DelayMs(100);
}
Kth5791_ProgramSave();                     // 保存校准参数到 MTP
Kth5791_OutputEnable();                    // 重新启动输出
```

> **注意：**校准期间写寄存器操作会被芯片忽略。校准自动完成后无需调用 `CalibrateStop`。

# 7 注意事项
Notes

1. **波特率要求**：KTH5791 的 UART 波特率固定为 **921600 bps**，部分 MCU 的主频可能无法精确产生此波特率。建议选择能整除 921600 的主频（如 72MHz、144MHz 等），或确保波特率误差在 ±2% 以内。
2. **包长度固定**：KTH5791 使用 **6 字节固定包长**协议，利用 UART 空闲中断（IDLE）检测帧结束。发送时必须连续发送 6 个字节，中间不能有超过 1 字节时间的间隔。
3. **写保护**：地址 **0x27**～**0x2A**（AngleOffset、AngleOut、PlaneOrXMag、YMag）**禁止写入。**尝试写入这些地址将被芯片忽略。
4. **输出使能前置条件**：**必须先调用 OutputEnable 启动输出，才能读取到有效的角度和磁场数据。**未使能输出时，0x28/0x29/0x2A 寄存器值为旧数据或无效值。
5. **校准期间写保护**：校准开始后，所有寄存器写入操作会被芯片忽略，直到校准停止。如需修改配置，请在校准前后进行。
6. **配置修改时机**：在输出使能状态下修改配置参数，建议先调用 OutputDisable 停止输出，修改完成后再重新 OutputEnable。
7. **编程密码**：向 MTP 保存配置时，必须向 Password 寄存器（0x05）写入正确密码 `0x1234` 并将最高位（bit15）置 1。密码错误时编程操作被忽略。
8. **校准要求**：
   - 校准期间需要旋转磁铁以采集完整的磁场圆周数据
   - 建议在最终产品安装状态下进行校准
   - 校准完成后务必保存到 MTP，否则断电丢失
   - 校准必须在输出停止状态下进行
9. **通信超时**：建议为每次读写操作设置 100ms 超时。在 921600 bps 下，6 字节传输时间约 65$\mu$s，100ms 足够。

# 8 附录：快速调试检查清单
Appendix: Quick Debug Checklist

Table 25: 调试检查清单

| 检查项 | 说明 |
|---|---|
| 硬件接线 | TX→RX, RX→TX, GND→GND 是否正确 |
| 供电电压 | KTH5791 供电是否正常（3.3V 或 5V） |
| 波特率 | MCU 端波特率是否为 921600 |
| 数据格式 | 8N1 (8 数据位, 无校验, 1 停止位) |
| 通信验证 | 尝试读取 Config 寄存器（0x03），确认返回正常 |
| CRC 校验 | 确认 CRC 计算方法为前 4 字节累加和 |
| 包长度 | 确保每次发送/接收都是 6 字节 |

CONNTEK
昆泰芯微电子

## Technical Support

如需技术支持或有任何疑问，请通过以下方式联系我们：

For technical support or any questions, please contact us through the following channels:


**技术支持邮箱:** support@conntek.com.cn

**销售咨询邮箱:** sales@conntek.com.cn

**公司网站:** www.conntek.com.cn



**昆泰芯微电子科技有限公司**

CONNTEK Microelectronics Co., Ltd.