C 语言版本函数调用指南

版本V1.0

1. 概述

本指南旨在为 CH32 单片机其他业务代码提供调用“图像预处理模块”和“TCP 打包与发送模块”的 C 语言 API 说明。这两个模块被封装为独立的底层库函数，负责将原始采集数据处理并推送至上位机，同时接收上位机配置。

开发者无需关心内部的滑窗算力优化或是 TCP 连接维持、分片等细节，只需按照约定的结构体提供入参并调用相关 API 即可。

2. 核心数据结构

2.1 原始图像数据结构（`RawImageBuffer_t`）

该结构由采集模块（黑盒）在采集完成后构建并传入处理库。**注意：本库所有内部计算和过滤针对的均是 16位整数矩阵 (精确到 0.1℃的定点数)**。

```
typedef struct {
    uint16_t* pData;          // 指向二维 16位 整数矩阵的起始指针(如 275 代表 27.5℃)
    uint16_t  Width;          // 原始图像宽度
    uint16_t  Height;         // 原始图像高度
    uint32_t  FrameNumber;    // 当前帧号（或时间戳），用于溯源
} RawImageBuffer_t;
```

2.2 预处理结果结构（`PreprocessResult_t`）

预处理模块运算完毕后产出的有效数据载荷结构。

```
typedef struct {
    uint8_t*  pValidData;     // 必须是 uint8_t 类型的外部缓冲区指针，规避结构体强转导致的内存对齐陷阱
    uint32_t  DataLength;     // 有效数据的字节长度 (宽度 * 高度 * sizeof(元素))
    uint16_t  ValidWidth;     // 产出图像宽度 (对于一维，可表示点数)
    uint16_t  ValidHeight;    // 产出图像高度
    int16_t   MinTemp;        // 有效区域内的最低温度
    int16_t   MaxTemp;        // 有效区域内的最高温度
    int16_t   AvgTemp;        // 有效区域内的平均温度
    int16_t   RoiTemp;        // 触发点温度
    uint8_t   Status;         // 处理状态 (0: OK, 1: 异常)
    uint32_t  FrameNumber;    // 继承自原始图像的帧号
} PreprocessResult_t;
```

2.3 网络封装缓冲结构（`TcpTxBuffer_t`）

专为 TCP 零拷贝封包设计的外部分配缓冲区。应用层（或专门的内存池）提供足够大的连续内存空间。

```
typedef struct {
    uint8_t* pBuffer;         // 指向由应用层分配的具体内存区 (包含物理组装全空间)
    uint32_t TotalCapacity;   // 该 Buffer 总容量
    uint32_t HeadOffset;      // 【核心】预留给封包用的首部偏移量。载荷将从 pBuffer + HeadOffset 开始写入
    uint32_t ValidPayloadLen; // 在调用封装函数后，由网络库回填的最终报文总长度
} TcpTxBuffer_t;
```

2.4 系统运行状态与配置（`ConfigCommon_t`，`Config2D_t`，`Config1D_t`）

系统参数配置不再使用单一的句柄封装，而是由通信协议中定义的三个独立结构体分别管理：`ConfigCommon_t`（通用参数）、`Config2D_t`（二维专有参数）与 `Config1D_t`（一维专有参数）。TCP 模块会通过回调动态更新这三个结构体，应用层需要保存最新配置以供预处理等模块使用（详见通信协议规范 2.0）。

3. 预处理模块 API

3.1 `Preprocess_Init`

- **功能**：初始化预处理模块，分配静态计算所需的工作内存（如列累加数组）。
- **原型**：`int8_t Preprocess_Init(uint16_t maxWidth, uint16_t maxHeight);`
- **入参**：
  - `maxWidth` / `maxHeight`：系统允许的最大处理分辨率，用于预分配内存池。
- **返回值**：`0` 成功，`<0` 失败。

3.2 `Preprocess_Execute`

- **功能**：对单帧二维矩阵进行裁剪并导出。系统会基于温度过滤启动滑动窗口去寻找热源，锁定目标区域（ROI）后，**会将这段区域内未被修改过的真实原始像素原样导出**，写入外部提供的 Buffer 中。
- **原型**：`int8_t Preprocess_Execute(const RawImageBuffer_t* input, TcpTxBuffer_t* out_buffer, PreprocessResult_t* output_meta);`
- **入参**：
  - `input`：采集模块提供的原始数据句柄。
  - `out_buffer`：应用层预先分配好的待发送缓冲区。库将直接通过 `out_buffer->pBuffer + out_buffer->HeadOffset` 零拷贝写入原始图像测温字节。
- **出参**：
  - `output_meta`：运算完成的规范化统计结果（长宽、最大、最小温度、平均温度等信息，统计依据同样为未失真的原始像素）。
- **返回值**：`0` 表示处理成功并锁定 ROI，`<0` 表示内存越界或其他致命错误。

3.3 `Preprocess_CheckInternalTrigger2D`

- **功能**：根据上位机设定的“触发 ROI 区域”、“温度触发阈值”及“判定条件(最高温/平均温)”，对传入的单帧原始图像进行内部热源触发判定。
- **原型**：`int8_t Preprocess_CheckInternalTrigger2D(const RawImageBuffer_t* input);`
- **入参**：
  - `input`：当前需要评判的原始图像。
- **返回值**：`1` 表示触发条件满足（画面中设定的 ROI 区域发现了足够高温的目标），`0` 表示未触发，`<0` 参数错误。
- **使用场景**：在内部触发机制（`TriggerMode = 0`）下，结合相机的连续 DMA 或轮询输出使用。

3.4 `Preprocess_Settings_Change`

- **功能**：安全地将通过 TCP 接收到的最新业务工作参数更新至预处理库内部。支持影子机制加锁更新策略，确保不会破坏正在进行处理的流水线帧。
- **原型**：`int8_t Preprocess_Settings_Change(const Config2D_t* newConfig2D, const Config1D_t* newConfig1D, const ConfigCommon_t* newCommon);`
- **入参**：
  - `newConfig2D` / `newConfig1D`：从上位机新下发的专用参数结构体（如 `TargetWidth`，`TriggerTemperatureThreshold`）。
  - `newCommon`：从上位机新下发的通用配置结构体。
- **返回值**：`0` 成功，`<0` 失败。

4. TCP 打包与发送模块 API

4.1 `TcpLogic_Init`

- **功能**：初始化整个应用层 TCP 管理任务，包括底层 Socket 绑定、接收任务建立以及缓冲池初始化。
- **原型**：`int8_t TcpLogic_Init(const uint8_t* deviceUUID, const uint8_t* authToken);`
- **入参**：
  - `deviceUUID`：16 字节的设备物理识别码（如 MAC 或 UID）。
  - `authToken`：身份验证令牌。
- **返回值**：`0` 成功，`<0` 失败。

4.2 `TcpLogic_Start`

- **功能**：非阻塞启动 TCP 服务管理引擎。此后库将在后台自动进行 5511 (控制流) 与 5512 (数据流) 的连接、握手(Handshake)、重连和心跳维持。
- **原型**： `void TcpLogic_Start(void);`

4.3 `TcpLogic_BuildAndSendTemperatureFrame`

- **功能**：将之前由预处理写入 `TcpTxBuffer_t` 内的数据（连同 `PreprocessResult_t` 结构体中的统计数据）封装。函数不需要搬移大块矩阵数据，直接利用移位操作和 `HeadOffset` 空间从前向后组装报文头（包含 TLV、Magic等），最后压入 5512 发送缓冲。
- **原型**：
  `int8_t TcpLogic_BuildAndSendTemperatureFrame(TcpTxBuffer_t* io_buffer, const PreprocessResult_t* processMeta, uint8_t frameType, uint8_t is2D);`
- **入参**：
  - `io_buffer`：包含已被预处理模块填充过载荷的 Buffer 包裹器。本函数执行完后，其中的 `ValidPayloadLen` 将被更新。
  - `processMeta`：包含帧号与温区极值统计。
  - `frameType`：帧类型 (0x00 LIVE, 0x01 TRIGGER, 0x02 MASKED)。
  - `is2D`： `1` 为二维数组， `0` 为一维。
- **返回值**： `0` 已组装完毕并压入队列， `<0` 失败。

4.4 `TcpLogic_GetLatestConfig`

- **功能**：主动查询并返回 TCP 库缓存的、由上位机最近一次下发的完整配置参数结构体。适用于应用层需要在非回调上下文中（如初始化后首次同步或故障恢复后重新拉取）获取当前生效配置的场景。
- **原型**： `int8_t TcpLogic_GetLatestConfig(ConfigCommon_t* out_common, Config2D_t* out_cfg2d, Config1D_t* out_cfg1d);`
- **出参**：
  - `out_common`：由调用者提供的通用配置结构体指针，库将最新缓存的通用配置拷贝到此处。
  - `out_cfg2d`：由调用者提供的二维专有配置结构体指针。
  - `out_cfg1d`：由调用者提供的一维专有配置结构体指针。
- **返回值**： `0` 成功（配置有效）， `-1` 尚未从上位机收到过任何配置， `<0` 其他错误。

4.5 接收与配置更新回调注册

TCP 库在后台线程处理收到的指令（如参数更改或控制信号）。主业务通过注册回调函数来处理这些上位机下发的事件，确保底层安全。

```
// 定义回调函数类型
typedef void (*ConfigUpdateCallback_t)(const ConfigCommon_t* common, const Config2D_t* cfg2d, const Config1D_t* cfg1d);
typedef void (*DetectionResultCallback_t)(uint32_t frameNumber, uint8_t resultStatus);

// 【注意】：此回调专门用于测试上位机主动请求帧。实际业务由硬件触发或DMA循环完成，业务代码不应依赖或实现此回调。
typedef void (*TempFrameRequestCallback_t)(uint8_t is2dRequest);

// 注册回调 API
void TcpLogic_RegisterConfigCallback(ConfigUpdateCallback_t cb);
void TcpLogic_RegisterDetectionCallback(DetectionResultCallback_t cb);
void TcpLogic_RegisterTempFrameRequestCallback(TempFrameRequestCallback_t cb);
```

应用示例：参数热更新
当注册了 `ConfigUpdateCallback_t` ，TCP 发生控制流的参数接收时，后台库完成参数解析及 CRC 校验后，触发此回调。用户可在回调中利用软中断或影子积存器机制将新参数赋予当前正在使用的 `SystemConfig_t` 。

5. 核心 API 设计准则与开发规范

为兼顾底层性能考量与可靠性网络封装，本库严格遵循以下原则：

1. **“传入指针 + 长度” 与预留偏移封包 (Zero-Copy Offset) 方案**
   物理内存分配由应用层（依托其内存池机制）负责，传递给库的操作句柄为 `out_buffer` 。预处理在填充矩阵数据时，会越过 `HeadOffset` （这个偏移量等于后续 TCP 组合包所需的 Header 及 TLV 指示器的大小）。后续 TCP 网络库封装时，**仅需要往前填充封包信息，无需对上百 KB 的矩阵数据做任何 `memcpy` 内存搬移动作**。

2. **防范访问陷阱 (Strict uint8_t Vectoring)**
   禁止将网口收发缓冲区中的指针强转为 `uint32_t` 或是具体通讯结构体使用。库内部所有的数据移动与地址递增处理严格使用 `uint8_t *` 处理网络数据流，从根本上阻绝了因不同编译器或单片机对齐法则不一导致的 HardFault 或越界访问。
3. **内置安全大小端转化 (Bitwise Disassembly/Assembly)**
   这是针对 16位 整数矩阵数据的核心保障。无论是将采集到的定点温度（ `uint16_t` / `int16_t` ）拆分成网络字节流（封包），还是从字节流解析成单片机状态（解包），都彻底抛弃了直接强转或结构体对齐强塞的方式。库内部严格规定使用单字节移位操作（如 `val = (buf[0]) | (buf[1]<<8);` ），完美解决“网络端与主机小端”之间的安全切换，并在提取矩阵数据时保证 2-Byte Little-Endian 输出。

6. 底层数据收发机制与平台移植架构 (Port 层解耦)

（预处理及TCP封包逻辑）绝对不会直接操作硬件寄存器或特定的 Socket API。

取而代之的是一个位于 `qdx_port.h` 的**硬件抽象层（HAL，Port层）**。

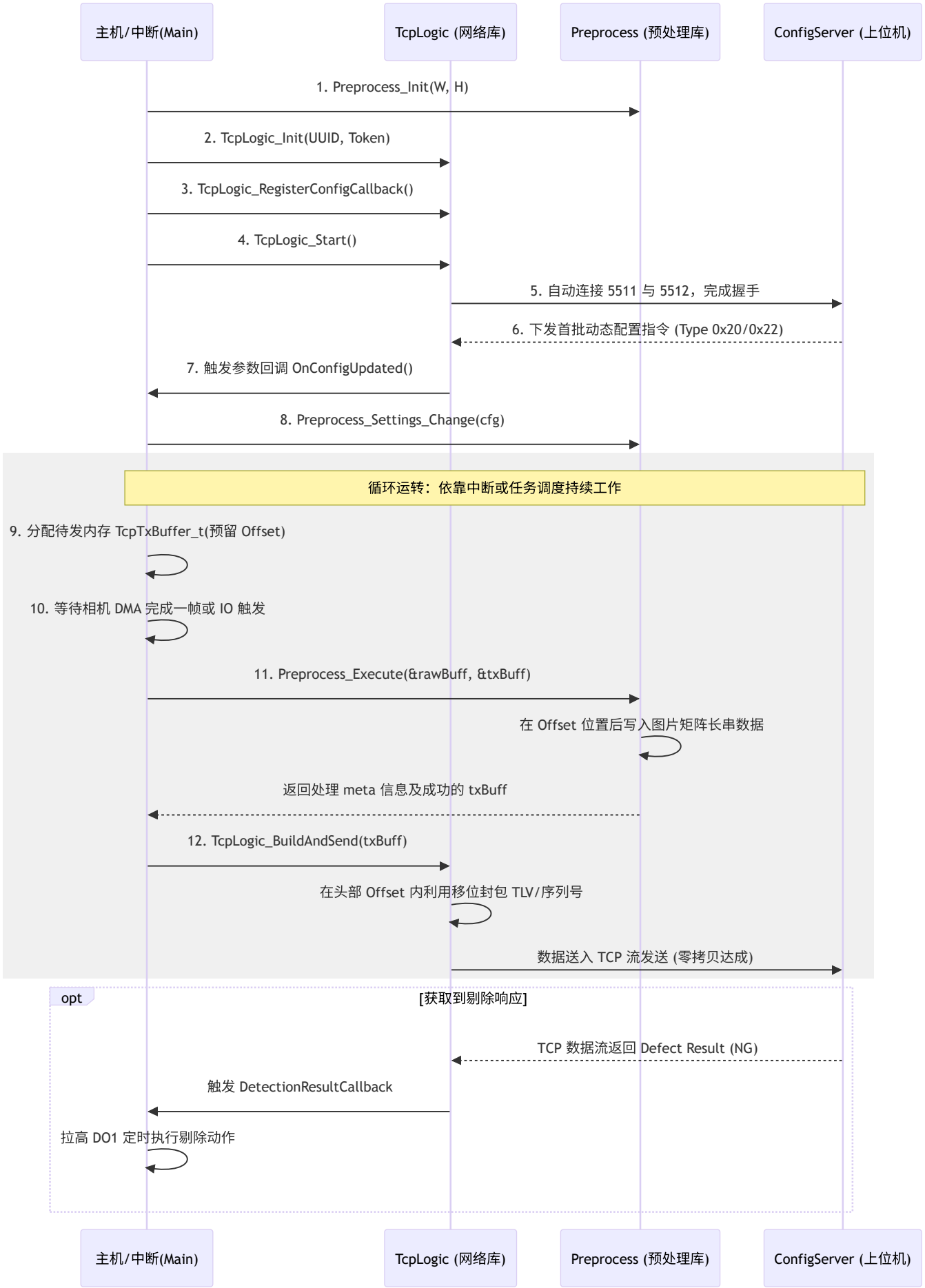
1. **发送：硬件发送缓冲区写入地址 (`qdx_port_tcp_send`)**
   当应用层调用类似 `TcpLogic_BuildAndSendTemperatureFrame` 时，传递的是在外部通过内存池预先分派好、并在前面加上了协议头的业务缓冲数组（即 `io_buffer->pBuffer` ）。网络库内部并不会“写入网卡相关的寄存器”，而是调用 `qdx_port_tcp_send` 。这保证了**只需将封装好的完整、连续的 RAM 地址指针和长度丢给驱动层**。内部的 WCH NET 或 LwIP 会从给定的这个内存地址将其发往 DMA 或以太网 MAC。
2. **接收：硬件接收缓冲区 (`qdx_port_tcp_recv`)**
   对应地，系统会启动后台线程监听 TCP 数据流。它每次都会从 `qdx_port_tcp_recv` 尝试获取数据，由底层协议栈驱动将接收到的真正网络比特流存入库提供的接收缓冲中，随后库再去解析配置命令。

7. 常规调用流程图 (伪代码模式)

```
// 1. 初始化
Preprocess_Init(MAX_W, MAX_H);
TcpLogic_Init(MyUUID, MyToken);
TcpLogic_RegisterConfigCallback(OnConfigUpdated);
TcpLogic_RegisterDetectionCallback(OnHardwareReject);

// 2. 启动网络引擎线程 (RTOS环境下)
TcpLogic_Start();

// 3. 图像中断 / DMA 轮询回调
void OnCameraDataReady(uint16_t* matrix, uint16_t w, uint16_t h) {
    RawImageBuffer_t rawBuff = {matrix, w, h, ++frameCnt};

    // 【核心】判断这帧图像里是否有物体达到了设定的触发温度
    if (Preprocess_CheckInternalTrigger2D(&rawBuff) == 1) {

        // 发现高温目标！分配一块发送专用的零拷贝缓冲
        TcpTxBuffer_t txBuff = MemoryPool_GetTxBuffer();
        PreprocessResult_t resMeta = {0};

        // 交由库执行滑动窗口剪裁，将最核心的高温区域内原始像素直接填入缓冲的预留偏移后
        if (0 == Preprocess_Execute(&rawBuff, &txBuff, &resMeta)) {
            // TCP封包：在头部预留好的 HeadOffset 内执行无损组包并发出
            TcpLogic_BuildAndSendTemperatureFrame(&txBuff, &resMeta, 0x01, 1);
        } else {
            MemoryPool_FreeTxBuffer(&txBuff);
        }
    }
}

// 4. 当参数更新回调触发时
void OnConfigUpdated(const ConfigCommon_t* common, const Config2D_t* cfg2d, const Config1D_t* cfg1d) {
    // 将获得配置输入给预处理模块，利用互斥锁安全地刷新工作参数（如 TriggerRoi）
    Preprocess_Settings_Change(cfg2d, cfg1d, common);
}
```

6. 函数调用时序图 (Control Flow)

通过下方时序图，可清晰展示从设备启动，到外部通信介入，再到硬件持续触发采集的数据流动与函数调用顺序。


主机/中断(Main)
TcpLogic (网络库)
Preprocess (预处理库)
ConfigServer (上位机)
1. Preprocess_Init(W, H)
2. TcpLogic_Init(UUID, Token)
3. TcpLogic_RegisterConfigCallback()
4. TcpLogic_Start()
5. 自动连接 5511 与 5512，完成握手
6. 下发首批动态配置指令 (Type 0x20/0x22)
7. 触发参数回调 OnConfigUpdated()
8. Preprocess_Settings_Change(cfg)
循环运转：依靠中断或任务调度持续工作
9. 分配待发内存 TcpTxBuffer_t(预留 Offset)
10. 等待相机 DMA 完成一帧或 IO 触发
11. Preprocess_Execute(&rawBuff, &txBuff)
在 Offset 位置后写入图片矩阵长串数据
返回处理 meta 信息及成功的 txBuff
12. TcpLogic_BuildAndSend(txBuff)
在头部 Offset 内利用移位封包 TLV/序列号
数据送入 TCP 流发送 (零拷贝达成)
opt
[获取到剔除响应]
TCP 数据流返回 Defect Result (NG)
触发 DetectionResultCallback
拉高 DO1 定时执行剔除动作
主机/中断(Main)
TcpLogic (网络库)
Preprocess (预处理库)
ConfigServer (上位机)