昂科AP8000V2平台二次开发需求与方案

# 引言

本文档是针对昂科AP8000V2平台需求与方案的说明，涵盖需求说明，方案说明等内容

# 本档案适用对象

* 技术支持工程师
* 维护工程师
* 软件工程师

# 版本记录

|  |  |  |  |
| --- | --- | --- | --- |
| **文档版本** | **文档编者** | **发布日期** | **修改说明** |
| V1.0 | Colin | 20250506 | * 初稿拟定 |
| V1.1 | Colin | 20250513 | * 新增jsonrpcserver架构设计图 |

# 方案约束

# 方案风险

# 方案说明

## 采集模块框架说明



关键思路：1.避开原有链路，直接从心跳包/pt包等解析开始分流CCmdHandler::HandlePacket出一条新链路,避免过多的拷贝，信号与槽以及原代码中有延迟的线程池，直接到CCmdHandler::CustomPT中的CustomMessageHandler::instance()->OnRecvDoCustom处理采集的数据；

2.串口的收发也进行了高并发处理，高频数据采集的信号只传递数据到达信号，不进行具体的数据传递和拷贝，大量的数据分流到每个worker的队列，由信号触发worker从队列取数据到sendData发送，具体实现见SerialportManager和SerialportWorker类；3.共享内存方案(待集成)。

## RmtServer模块设计



作为一个独立的服务存在，与外部ClientApp通信，使用jsonrpc，把所有AG06上每个BPU上数据上传到ClientApp中，同时接收ClientApp的消息。测试上位机主要下发指令如下。

* StartService
* LoadProject
* SiteScanAndConnect
* DoJob
* SetAdapterEn
* DoCustom
* StopService
* GetProjectInfo

RMTServer主要上传指令如下：

* "setloadresult"
* "setjobresult"
* "SetResult"

## JsonRpcserver类设计

1. 概述

`JsonRpcServer` 类实现了一个基于 TCP 的 JSON-RPC 2.0 服务器。它被设计为应用程序的中心远程过程调用 (RPC) 服务端点，允许外部客户端通过网络与应用程序进行交互。该服务器采用长连接通讯模式，并支持多个客户端同时连接。为了确保消息的完整性和正确解析，所有 JSON-RPC 消息都封装在一个自定义的二进制协议帧中。

该类被实现为单例模式，以便在应用程序的各个模块中方便地访问和管理。

2. 主要功能

-1.JSON-RPC 2.0 协议支持\*\*: 严格遵循 JSON-RPC 2.0 规范处理请求和响应。

-2.TCP 长连接: 为每个客户端维护一个持久的 TCP 连接，允许双向通信和异步通知。

-3.自定义消息帧:

A.在 TCP 流之上使用自定义的二进制头部来封装 JSON-RPC 消息。

B.头部包含：Magic Number (4字节), 协议版本号 (2字节), JSON 负载长度 (4字节), 以及保留字段 (22字节)。

-4.多客户端支持: 能够同时处理来自多个客户端的连接和请求。

-5.单例模式: 提供全局唯一的服务器实例。

-6.可扩展的命令处理: 通过注册机制，可以方便地添加新的 RPC 方法及其处理函数。

-7.异步任务处理: 支持处理耗时操作，例如：

A.立即响应客户端请求已被接受。

B.在后台线程中执行实际任务。

C.任务完成后，通过长连接主动向客户端发送通知。

-8.设备发现与通知:

A.集成 `AngKScanManager` 进行设备扫描。

B.允许客户端订阅设备发现事件。

C.当新设备被发现或设备信息更新时，主动向订阅的客户端发送通知。

-9.日志记录: 集成 `AngkLogger` 进行详细的操作和错误日志记录。

3. 设计细节

-3.1. 单例模式

A.通过私有构造函数和静态 `Instance()` 方法实现。

B.`m\_instance` 静态成员指针持有唯一的实例。

C.提供全局访问函数 `GetGlobalJsonRPCServerApp()`。

3.2. 连接管理

A.继承自 `QTcpServer`。

B.`incomingConnection()`: 处理新的客户端连接请求，为每个连接创建一个 `QTcpSocket`。

C.`m\_clientBuffers`: 使用 `QHash<QTcpSocket\*, QByteArray>` 存储每个连接的 `QTcpSocket` 及其关联的接收缓冲区。

D.`OnSocketDisconnected()`: 处理客户端断开连接事件，清理相关资源，并从订阅者集合中移除。

3.3. 数据接收与解析

A.`OnReadyRead()`: 当 `QTcpSocket` 接收到数据时被调用。

B.数据首先被追加到对应客户端的 `m\_clientBuffers` 中。

C.服务器循环尝试从缓冲区中解析完整的消息帧：

1. 检查是否有足够的字节构成头部 (`HEADER\_LENGTH`)。

2. 校验 Magic Number 和协议版本号。

3. 从头部读取 JSON 负载的长度。

4. 检查是否有足够的字节构成完整的 JSON 负载。

5. 提取 JSON 负载。

6. 从缓冲区中移除已处理的数据包。

D.`ProcessPacket()`: 将提取的 JSON 负载（`QByteArray`）解析为 `QJsonDocument`。

E. `ProcessJsonRpcRequest()`:

1. 验证 `QJsonDocument` 是否为有效的 JSON-RPC 2.0 请求对象（包含 "jsonrpc", "method" 字段）。

2. 提取 "method", "params", 和 "id" 字段。

3. 根据 "method" 字符串从 `m\_handlers` 查找对应的处理函数。

4. 调用找到的处理函数。

3.4. 命令处理与分发

A.`RpcHandler`: 使用 `std::function<void(QTcpSocket\*, const QJsonObject&, const QJsonValue&)>` 定义 RPC 处理函数的类型。

B.`m\_handlers`: 使用 `QHash<QString, RpcHandler>` 存储方法名到其处理函数的映射。

C.`RegisterHandlers()`: 在服务器初始化时，将所有支持的 RPC 方法名和对应的处理函数（通常是类的成员函数，通过 lambda 或 `std::bind` 包装）注册到 `m\_handlers` 中。

D.每个 `Handle<CommandName>()` 方法负责具体 RPC 命令的逻辑实现，并使用 `SendMessage()` 回复客户端。

3.5. 消息发送

A.`SendMessage()`:

1. 将 `QJsonDocument` (响应或通知) 序列化为紧凑的 JSON 字符串 (`QByteArray`)。

2. 构建包含自定义二进制头部的完整数据包。

3. 通过 `QTcpSocket::write()` 发送数据包。

B.`SendNotification()`: 辅助函数，用于构建和发送 JSON-RPC 通知（没有 "id" 字段）。

C.`SendClientDoCmd()`: 对外提供的接口，用于服务器主动向特定客户端发送特定命令的通知。

3.6. 异步操作与通知

A.对于耗时操作（如 `HandleDoJob`）：

1. 立即向客户端返回一个带有任务 ID 的接受响应。

2. 使用 `QtConcurrent::run` 将实际工作放到工作线程中执行。

3. 工作线程完成后，通过 `QMetaObject::invokeMethod` 以队列连接的方式调用主线程中的槽函数（如 `sendJobCompletionNotification`）。

4. 该槽函数负责向原始客户端发送任务完成的通知。

B.设备发现通知 (`OnDeviceFoundOrUpdated`):

1. 当 `AngKScanManager` 发出设备更新信号时，此槽被调用。

2. 从 `AngKDeviceModel` 获取设备信息。

3. 将设备信息转换为 JSON。

4. 遍历 `m\_scanSubscribers` (一个 `QSet<QTcpSocket\*>`)，向所有订阅了扫描更新的客户端发送 `DeviceDiscovered` 通知。

3.7. 错误处理（待完善）

A.在数据解析（如无效 Magic Number、不支持的协议版本、JSON 解析错误）和请求处理（如方法未找到、无效参数）的各个阶段进行错误检测。

B.发生错误时，向客户端发送符合 JSON-RPC 2.0 规范的错误响应。

C.使用 `ALOG\_WARN` 和 `ALOG\_ERROR` 记录错误信息。

D.存在问题？

4.docustom/dojob详细设计

5.存在问题:

**A.对比开源库libjson-rpc-cpp（<https://github.com/cinemast/libjson-rpc-cpp>），Alibjson-rpc-cpp的架构**:

1.接口定义先行 :

使用一个单独的规约文件 (如 spec.json) 来定义 RPC 方法、参数和返回类型。

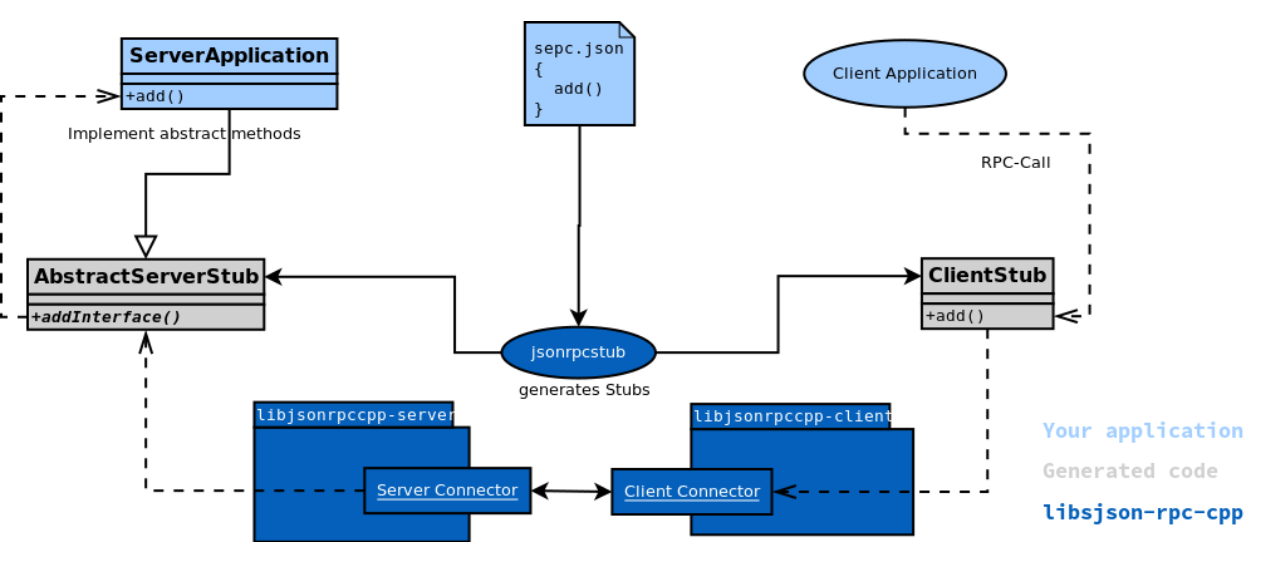
这是 API 的“单一事实来源”。

2.代码生成 :jsonrpcstub (或类似工具) 根据规约文件自动生成服务器端骨架 (AbstractServerStub) 和客户端存根 (ClientStub)。服务器应用程序 (ServerApplication) 继承生成的 AbstractServerStub 并实现具体的业务逻辑方法。客户端应用程序 (ClientApplication) 使用生成的 ClientStub 来发起类型安全的 RPC 调用。

3.关注点分离 :RPC 协议处理、网络连接 (Server Connector, Client Connector) 和序列化/反序列化等底层细节由库和生成的代码处理。开发者主要关注于在 ServerApplication 中实现业务逻辑。

4.强类型与一致性:生成的存根通常能提供编译时的类型检查，减少运行时错误。

确保客户端和服务端对接口的理解一致。



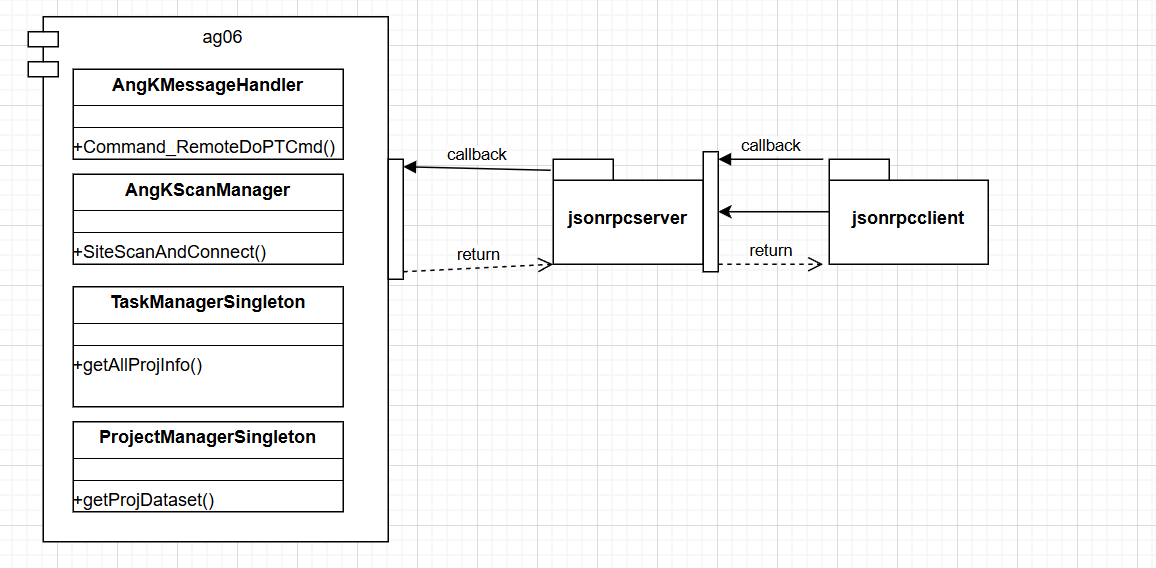
**B.仿照IceServerApp.cpp实现的架构：**

1.手动实现:服务器直接继承 QTcpServer，手动管理 TCP 连接和原始字节流。手动解析 JSON 请求 (ProcessPacket, ProcessJsonRpcRequest)，包括验证 JSON-RPC 协议版本、方法名、ID 等。使用 QHash<QString, RpcHandler> m\_handlers 将方法名 (字符串) 映射到具体的处理函数 (std::function)。

2参数处理在业务逻辑中:每个处理函数 (如 HandleDoJob, HandleDoCustom) 负责从 QJsonObject cmdData 中手动提取、验证和转换参数。这涉及大量的 cmdData.contains(), cmdData["key"].isType(), cmdData["key"].toType() 调用。

3.灵活性高但易出错:开发者对所有细节有完全控制。但也更容易因手动处理字符串、类型转换等引入错误.方法名、参数名和类型的任何不匹配都可能导致运行时问题，且不易静态检查。

4.耦合度较高:网络通信、JSON-RPC 协议解析、方法分发和参数处理紧密耦合在 JsonRpcServer 类及其处理函数中。



**C.差异性总结:**

| **特性** | **图示设计 (代码生成)** | **`JsonRpcServer` (手动实现)** |
| --- | --- | --- |
| **接口定义** | 显式规约文件 (`spec.json`) | 隐式，通过代码和处理函数签名定义 |
| **代码实现** | 自动生成存根，开发者实现业务逻辑 | 完全手动编写所有层面代码 |
| **类型安全** | 通常较高 (编译时检查) | 较低 (依赖运行时检查) |
| **关注分离** | 清晰 (RPC框架 vs. 业务逻辑) | 较低 (多方面逻辑耦合在同一模块) |
| **维护性** | 接口变更时，重新生成代码，系统性强 | 接口变更时，需手动修改多处，易遗漏 |
| **开发效率** | 初期需学习工具，但后续方法增多时效率高 | 初期简单，但方法和参数复杂后，重复工作多 |
| **错误处理** | 框架可能提供标准化错误处理 | 手动在各处实现 |
| **客户端集成** | 生成的客户端存根简化客户端开发 | 客户端需自行构造符合约定的 JSON 请求 |

**4.AI分析优缺点：**

通常情况下，图示中基于规约和代码生成的设计模式被认为是更优的，尤其对于中大型项目或需要长期维护的 API：

1.健壮性 (Robustness): 规约文件提供了清晰的契约。代码生成减少了人为错误，并有助于在编译阶段捕获接口不匹配等问题。

2.可维护性 (Maintainability): 当 API 演进时，修改规约文件并重新生成代码比手动修改多处代码更可靠、更高效。

3.可扩展性 (Scalability): 添加新方法或修改现有方法时，流程更规范，不易引入副作用。

4.团队协作 (Team Collaboration): 规约文件是客户端和服务器开发团队之间清晰的沟通桥梁。

5.开发效率 (Long-term Efficiency): 虽然初始设置可能需要一些学习成本，但长期来看，自动处理 RPC 的底层细节可以显著提高开发效率，让开发者更专注于业务逻辑。

JsonRpcServer 的手动实现方式对于非常简单、方法数量极少或需要对底层进行极致控制的场景可能适用。但随着方法和参数的增加，其复杂度和维护成本会迅速上升，如当前代码中 HandleDoJob 方法内大量的参数提取和验证逻辑所示，这些在代码生成方案中大部分可以自动化。因此，如果追求更高的代码质量、更低的维护成本和更好的开发体验，图示的设计模式是更推荐的选择。

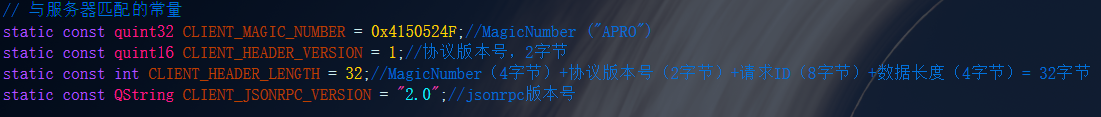
# Custom格式定义

透传，无需知道

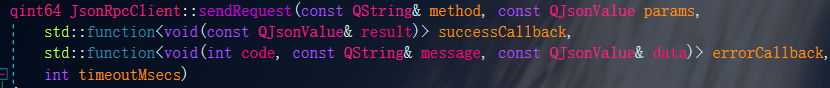
# 通讯接口定义

## 通用结构体

### 1.1常量定义：



### 1.2内容定义：



发送Send：

{

"id":1,

"jsonrpc":"2.0",

"method":"DoCustom",//方法名

"params":{

}

}

接收Receive：

{

"id":1,

"jsonrpc":"2.0",

"method":"DoCustom" //方法名

"status": "success", // 状态 success /fail

"data":{

// 存放与当前命令相关的具体数据

}

}

## 接口函数定义

### 2.1 StartService（初始化）

（启动服务，使用该函数时不会连接编程器，只会启动AG06，不需要其他处理）

通过指定Aprog路径，通过命令行直接启动应用，分为以下两种：

1. .\Aprog.exe -r 默认启动12345端口jsonrpcserver服务器
2. .\Aprog.exe -r -p 8080指定端口号启动

### 2.2 SiteScanAndConnect (扫描并初始化页面)

发送Send：

{

"id": 1,

"jsonrpc": "2.0",

"method": "SiteScanAndConnect",

"params": {}

}

接受Receive：

{

"id": 1,

"jsonrpc": "2.0",

"method": "DeviceDiscovered",

"params": {

"device": {

"chainID": 1,

"dpsFpgaVersion": "0x2002285A",

"dpsFwVersion": "V2.2.0006A",

"firmwareVersion": "2.02.031",

"firmwareVersionDate": "20250111",

"fpgaLocation": "Normal",

"fpgaVersion": "0x2030121A",

"hopNum": 0,

"ip": "192.168.71.23",

"isLastHop": false,

"linkNum": -1,

"mac": "00:3A:22:00:01:02",

"mainBoardInfo": {

"hardwareOEM": "ACVIEW",

"hardwareSN": "A06U24120700012",

"hardwareUID": "CCE3DF00A00C0672",

"hardwareVersion": "0100-010000-010000-010000"

},

"muAppVersion": "2.02.029",

"muAppVersionDate": "20240111",

"muLocation": "Normal",

"port": "8080",

"siteAlias": "Site01"

},

"ipHop": "192.168.71.23:0"

}

}

### 2.3 LoadProject(任务文件);

发送Send：

{

"id": 2,

"jsonrpc": "2.0",

"method": "LoadProject",

"params": {

"path": "D:/Projects/FactoryTask/",

"taskFileName": "production\_schedule.actask"

}

}

接收Receive：

{

"id": 2,

"jsonrpc": "2.0",

"method": "LoadProject",

"data": {

"errorCode": -32000,

"message": "File not found",

"details": "Path: D:/InvalidDir/ | File: missing.json"

}

}

### 2.4 SetAdapterEn（初始化site和socket状态）

发送Send：

{

"id": 1,

"jsonrpc": "2.0",

"method": "SiteSocketInit",

"params": {

"Site": [

{

"row": 1,

"col": 1,

"sockets": [

{"row": 1, "col": 1, "enabled": true},

{"row": 1, "col": 2, "enabled": false},

...

(最多8行3列配置)

]

},

...

(最多10行5列配置)

]

}

}

接收Receive：

{

"id": 1,

"jsonrpc": "2.0",

"method": "SiteSocketInit",

"data": {

"Site": [

{

"row": 1,

"col": 1,

"sockets": [

{"row": 1, "col": 1, "enabled": true},

{"row": 1, "col": 2, "enabled": false},

...

(最多8行3列配置)

]

},

...

(最多10行5列配置)

]

}

}

### 2.5 GetAdapterEn(获取当前使能状态)

### 2.6 DoJob

命令说明：请求服务器执行相应的编程命令，编程命令的允许操作依赖于创建的工程文件有，AngKProjDataset类管理，此类目前直接由gui页面持有为其成员变量，通过指针传递的方式在各个gui之间传递,不方便二次开发，强烈要求改为单例类。该命令为异步执行，请求成功之后函数会立即返回。执行结果服务器端通过 SetJobResult 通知客户端。需要根据返回的工程文件构建docmdSeqJson，按如下格式发送出来

发送Send：

{

"id": 2,

"jsonrpc": "2.0",

"method": "DoJob",

"params": {

"docmdSeqJson": {

"CmdID":"1806",

"CmdRun":"Read",

"CmdSequences":[{"ID":"806","Name":"Read"}],

"CmdSequencesGroupCnt":1

}

"strIP":"1806",

"nHopNum":0, 目前固定值0

"CmdFlag":0, 目前固定值0

"CmdID":1047 区分dojob还是docustom

"SKTEn":0xFFFF 全部使能，当前只能支持全部使能

"BPUID":8 目前固定值

}

}

接收Receive：

{

"id": 2,

"jsonrpc": "2.0",

"method": "ClientDoCmd",

"params": {

"cmd": "setDoJobResult",

"data": {

"strIP":"1806",

"nHopNum":0,

"BPUID":1,

"SKTIdx":0,

"result":Passed,

}

}

}

### 2.7 DoCustom（实现测试项）

命令说明：同上，透传

结构如下：

Send结构

{

"id": <唯一请求ID>,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"testStep": "<测试步骤名称>",

"commands": <具体指令参数>

}

}

Receive结构

{

"id": <必须与发送ID一致>,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"testStep": "<对应测试步骤>",

"results": <测试结果数据>

}

}

#### 测试步骤1 电学测试 electricalTest

发送Send：

{

"id": 1,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"testStep": "ElectricalTest",

"subSteps": [

{

"subStepType": "PinContinuityTest",

"config": {

"pinCounts": 7,

"lowerLimit": 0.5,

"upperLimit": 1.5

}

},

{

"subStepType": "PowerOnCurrentTest",

"config": {

"voltageLowerLimit": 3.1,

"voltageUpperLimit": 3.5,

"currentLowerLimit": 3,

"currentUpperLimit": 3

}

}

]

}

}

接收Receive：

{

"id": 1,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"testStep": "ElectricalTest",

"subSteps": [

{

"subStepType": "PinContinuityTest",

"results": {

"pinStatus": {

"pin1": {

"status": "OK",

"measuredValue": 1.2

},

"pin2": {

"status": "FAIL",

"measuredValue": 0.3

},

"pin3": {

"status": "OK",

"measuredValue": 1.2

},

"pin4": {

"status": "FAIL",

"measuredValue": 0.3

},"pin5": {

"status": "OK",

"measuredValue": 1.2

},

"pin6": {

"status": "FAIL",

"measuredValue": 0.3

},"pin7": {

"status": "OK",

"measuredValue": 1.2

}

}

}

},

{

"subStepType": "PowerOnCurrentTest",

"results": {

"status": "OK",

"voltageMeasuredValue": 1.2,

"currentMeasuredValue": 1.2

}

}

]

}

}

#### 测试步骤2 FrequencySweep

发送Send:

{

"id": 2,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"testStep": "FrequencySweep",

"config": {

"startFreq": 1,

"endFreq": 100,

"stepSize": 1,

"iterations": 10

}

}

}

接收Receive:

{

"id": 2,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"frequencyResponse": [

{"freq": 1000, "amplitude": -3.2, "phase": 45},

{"freq": 5000, "amplitude": -12.5, "phase": 85}

],

"status": "COMPLETE"

}

}

}

#### 测试步骤3标度标定

发送Send：

{

"id": 3,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"testStep": "ScalingCalibration",

"config": {

"timeInterval": 100,

"samples": 1000,

"axis": "X"

}

}

}

接受Receive：

{

"id": 3,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"calibrationData": {

"rawADC": [1024, 2048, 3072],

"statistics": {

"maxValue": 3072,

"minValue": 1024,

"averageValue": 2048,

"standardDeviation": 724.0

},

"processedValue": 1.5,

"status": "OK"

}

}

#### 测试步骤4 功能安全标定

发送Send：

{

"id": 4,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"testStep": "FunctionalSafety",

"recipe": [

{

"instruction": "ENABLE\_SAFE\_MODE",

"interval": 50,

"readBack": true

},

{

"instruction": "VERIFY\_CHECKSUM",

"interval": 30,

"readBack": false

}

]

}

}

接收Receive：

{

"id": 4,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"errorDetection": {

"crcCheck": true,

"timeout": 200

}

}

}

#### 测试步骤5 烧写OTP

发送Send：

{

"id": 5,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"testStep": "OTPProgramming",

"condition": {

"voltage": 8.5,

"tolerance": 0.1

},

"recipe": "OTP\_WRITE\_PATTERN"

}

}

接收Receive：

{

"id": 5,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"errorCode": -32002,

"errorDetails": {

"voltage": 8.3,

"message": "Voltage out of tolerance"

}

}

}

#### 测试步骤6 PSI5测试

发送Send：

{

"id": 6.1,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "SelfTest",

"command": "READ\_DIAGNOSTICS"

}

},

},

{

"id": 6.2,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "NoiseMeasurement",

"config": {

"samples": 500,

"deviceType": "SENSOR\_A",

"threshold": {

"absMax": 50,

"absMin": 0

}

}

},

{

"id": 6.3,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "ManchesterCurrent",

"config": {

"baseCurrent": {

"min": 4,

"max": 6,

"value": 5

},

"deltaCurrent": {

"min": 22,

"max": 30,

"value": 25

}

}

},

{

{

"id": 6.4,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "BitTimeMeasurement",

"config": {

"threshold": 480,

"unit": "LSB"

}

}

},

{

"send": {

"id": 6.5,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "TDelayMeasurement"

}

},

{

{

"id": 6.6,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "SyncPulseVDD",

"config": {

"voltage": 8.5,

"tolerance": 0.1

}

}

},

{

{

"id": 6.7,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "AsyncCommunication",

"command": "START\_ASYNC\_MODE"

}

},

{

{

"id": 6.8,

"jsonrpc": "2.0",

"method": "DoCustom",

"params": {

"subStep": "DualTimeSlot",

"config": {

"timeSlot1": {

"expected": 100,

"unit": "μs"

},

"timeSlot2": {

"expected": 200,

"unit": "μs"

}

}

}

}

}

接收Receive：

{

"id": 6.1,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"diagnosticCode": "01E7",

"error": {

"code": -32004,

"message": "Invalid diagnostic code",

"allowedValues": ["01E7"]

}

}

}

{

"id": 6.2,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"noiseStats": {

"max": 55,

"min": -1,

"avg": 22.3,

"stdDev": 6.7

},

"errorCode": {

"code": -32005,

"message": "Absolute value exceeds threshold",

"rawValue": 55

}

}

}

}，

{

"id": 6.3,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"currentParams": {

"baseCurrent": 5.1,

"deltaCurrent": 24.8

},

"compliance": {

"base": "PASS",

"delta": "PASS"

}

}

}

},

{

"id": 6.4,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"bitTime": {

"measured": 485,

"status": "FAIL",

"tolerance": {

"min": -480,

"max": 480

}

}

}

}

},

{

"id": 6.5,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"tDelay": {

"measured": 120,

"unit": "ns",

"status": "PASS"

}

}

}

},

{

"id": 6.6,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"voltage": {

"measured": 8.3,

"status": "FAIL",

"allowedRange": [8.4, 8.6]

}

}

}

},

{

"id": 6.7,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"communicationStatus": "ESTABLISHED",

"errorCount": 0

}

}

},

"receive": {

"id": 6.8,

"jsonrpc": "2.0",

"method": "DoCustom",

"data": {

"timeSlot1": {

"measured": 105,

"status": "PASS"

},

"timeSlot2": {

"measured": 195,

"status": "PASS"

}

}

### 2.8 GetProjectInfo（获取项目具体信息）

命令说明：当 LoadProject 成功之后，也就是客户端接收到 SetLoadResult 并得知加载工程成功之后，可以通过发送该命令请求服务器返回工程文件中的适配板等相关信息。客户端可以将这个信息用于显示，比较等。

varCmdData:无pvarRespData：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 占用字节数 | 含义 | 说明 |
| Checksum | 以\0 结尾的字符串 | 工程文件中客户烧录档案对应的校验值 | 16 进制字符串，前面的 0 会被忽略不显示在字符串中。 |
| DevName | 以\0 结尾的字符串 | 烧录的芯片名称 |  |
| AdpName | 以\0 结尾的字符串 | 对应的适配板的名称 |  |
| ICManu | 以\0 结尾的字符串 | 芯片厂商名称 |  |
| Type | 以\0 结尾的字符串 | 芯片类型 |  |
| Package | 以\0 结尾的字符串 | 芯片封装名称 |  |

在内存中的回应数据字节序示例如下：

其中 7F7A560 表明 Checksum 的值为 0x7F7A560.

字符串的编码格式为多字节字符集。

返回值：成功返回 0，失败返回其他值。返回成功之后，客户端通过 pvarRespData

得到相应的信息。

### 2.9 日志接口（暂未定义）

### 2.10 事件接口（暂未定义）

### 2.11 GetJobResult（暂未定义）

文档中无参考

### 2.12 GetProgrammerInfo

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@brief GetProgrammerInfo 功能： 获取烧录器的信息

@param[in] Buffer: 烧录信息存放位置,返回格式如下[{"SiteSN":"00001","Type":"PT100","SlotNum:"8", {"index":"0","SN":"4000000000123456","limited":"30000","current":"9000"}, {"index":"1","SN":"4000000000123457","limited":"30000","current":"9000"}, {"index":"2","SN":"4000000000123458","limited":"30000","current":"9000"}, {"index":"3","SN":"4000000000123459","limited":"30000","current":"9000"}, {"index":"4","SN":"400000000012345A","limited":"30000","current":"9000"}, {"index":"5","SN":"400000000012345F ","limited":"30000","current":"9000"}, {"index":"6","SN":"40000000001234DE", "limited":"30000","current":"9000"}, {"index":"7","SN":"40000000001234DF", "limited":"30000","current":"9000"}} {"SiteSN":"00002","Type":"PT100","SlotNum:"8", {"index":"0","SN":"4000000000123500", "limited":"30000","current":"5000"}, {"index":"1","SN":"4000000000123501", "limited":"30000","current":"5000"}, {"index":"2","SN":"4000000000123502", "limited":"30000","current":"5000"}, {"index":"3","SN":"4000000000123503", "limited":"30000","current":"5000"}, {"index":"4","SN":"4000000000123504", "limited":"30000","current":"5000"}, {"index":"5","SN":"4000000000123505", "limited":"30000","current":"5000"}, {"index":"6","SN":"4000000000123506", "limited":"30000","current":"5000"}, {"index":"7","SN":"4000000000123507", "limited":"30000","current":"5000"}} ] 备注：

ProgrammerSN: 烧录器序列号成员， 后面需要跟具体的值用":"隔开， 比如"00001",表示该烧

录器的序列号为

SlotNum ： 烧录器上的烧录端口数， 后面需要跟具体的值用":"隔开， 比如"4"， 表示该烧录器有个端口

index : 烧录器端口号， 后面需要跟具体的值用":"隔开， 比如"3"，表示该烧录器的第三个端口，端口号从开始到 SlotNum-1.

SN: 该端口号上的座子的序列号， 后面需要跟具体的值用":"隔开， 比如"4000000000123501"。limited: 该端口号上的座子使用次数限制， 后面需要跟具体的值用":"隔开， 比如"30000"， 表示该座子最多下压次

current : 该端口号上的座子已使用次数， 后面需要跟具体的值用":"隔开， 比如"9000"， 表示

该座子已经使用次@param[in] Size: Buffer 大小@return

TRUE 执行成功

FALSE 执行失败\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# 其他说明

（暂无）

# 防呆说明

列出如何防呆的建议

# SVN说明

# 测试建议

