UM2354
用户手册

用于连接Alexa Voice Service的ST框架，STM32Cube软件扩展

引言

本用户手册将介绍STM32Cube扩展包的内容，该扩展包可从STM32应用连接到Alexa Voice Service（AVS）。

X-CUBE-VS4A扩展包提供应用示例，将STMicroelectronics板连接到Amazon服务器，以便在STM32器件上轻松实现面向AVS的产品。

X-CUBE-VS4A扩展包由一组库和基于STM32F7系列微控制器（充当支持Alexa的器件）的应用示例组成。

X-CUBE-VS4A在32F769IDISCOVERY板上运行。它具有可直接运行的固件示例，用于演示简单智能音箱的实现。

X-CUBE-VS4A提供了下列功能：

- 板配置接口
- TCP/IP连接功能
- AVS协议封装，便于轻松实现应用
- 应用特定的服务
- 用于Alexa Voice Service的STMicroelectronics框架
- 创建面向AVS的STM32应用
- 可替换的基础音频采集
- 有限音频播放器示例

注： *X-CUBE-VS4A不包含用于音频前端增强的软件，也不包含需要与所有不同的音频服务兼容的完整媒体播放器。*

目录

图片索引

表格索引

1 概述

本用户手册将介绍X-CUBE-VS4A扩展包和STVS4A语音服务中间件。侧重点在于说明其使用，而不是解释Alexa架构或AVS帐户创建。后者的相关说明可以在Amazon网站和开发者网站上找到：

- https://alexa.amazon.com
- https://developer.amazon.com

X-CUBE-VS4A扩展包在STM32基于Arm®(a) Cortex®-M处理器的32位微控制器上运行。

*表 1*给出了相关术语和缩略语的定义，帮助您更好地理解本文档。

表1. 术语和缩略语列表

术语	定义
Alexa	Amazon的基于云的语音服务
API	应用编程接口
AVS	Alexa Voice Service
BSP	板级支持包
DHCP	动态主机配置协议
EVT	事件
HAL	硬件抽象层
IDE	集成开发环境
IP	互联网协议
JSON	JavaScript对象符号
STVS4A	用于设计基于STM32的AVS器件的意法半导体SDK
SDK	软件开发套件

.

a. Arm是Arm Limited（或其子公司）在美国和/或其他地区的注册商标。

2 关于安全的重要说明

注意： 应用开发人员必须注意安全方面，并通过实施机制来保护用于连接AVS的令牌和密钥。

X-CUBE-VS4A扩展包中提供的应用示例未实施此类机制。它只提供了一个基本实现，以便理解栈接口。

3 软件包说明

本章详述了X-CUBE-VS4A扩展包的内容和使用。

3.1 概述

X-CUBE-VS4A扩展包基于STVS4A，是支持基于STM32的Alexa Voice Service器件设计的软件开发套件。STVS4A具有服务API，用于连接AVS服务器和与服务器协商身份验证。STVS4A还为接收指令和向服务器发送事件提供支持。此外，STVS4A提供一组音频支持，包括麦克风采集和音频回放。通过添加外部元件， STVS4A支持可扩展至文字识别。

STVS4A支持Alexa Voice Service API版本v20160207。

可支持以下集成开发环境：

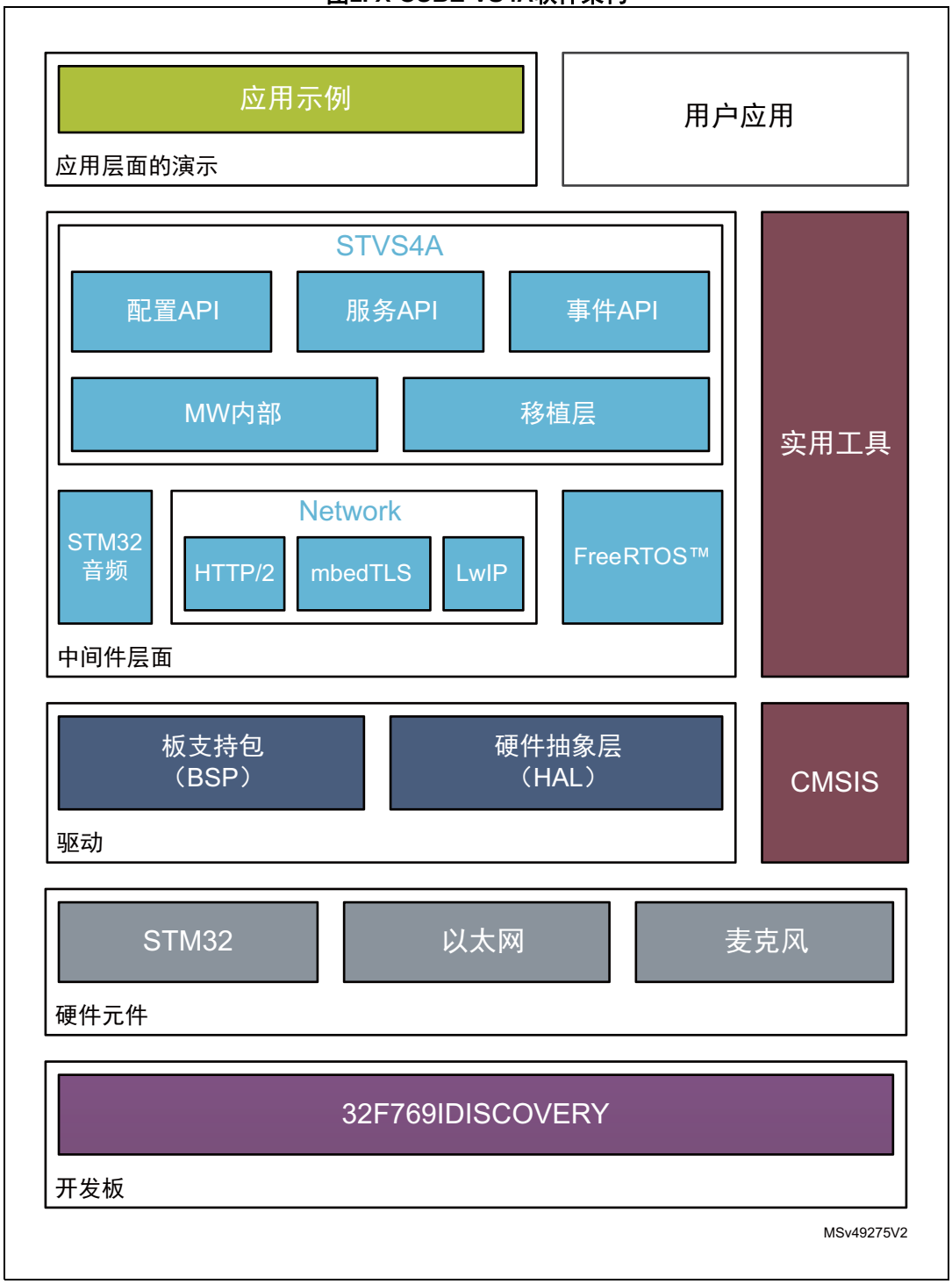
- IAR Embedded Workbench® for Arm® (EWARM)
- Keil®微控制器开发套件（MDK-ARM）
- System Workbench for STM32

注： *有关所支持IDE版本的信息，请参阅软件包根文件夹中的版本说明。*

3.2 架构

*图 1*概括了X-CUBE-VS4A软件架构。

图1. X-CUBE-VS4A软件架构



X-CUBE-VS4A是STM32Cube的扩展包，它：

- 完全兼容STM32Cube架构
- 扩展STM32Cube用于支持AVS的应用的开发
- 使用STM32微控制器的硬件抽象层STM32CubeHAL

除了*第 3.1 节：概述*中所述的功能，X-CUBE-VS4A扩展包和STVS4A中间件还具有以下功能：

- STVS4A为配置提供API，该API主要用在初始化阶段。
- STVS4A还提供一个API，用于通过应用自定义某些服务，例如访问持久存储器以加载和保存AVS令牌。
- STVS4A使用事件驱动的架构。初始化后，STVS4A主要使用通知与用户应用通信。通知是包含一个或多个参数的代码。参数是取决于事件的不透明变量。它可以理解为整数、结构指针、句柄或任何其他含义。参数的含义记录在事件文档中。
- 移植层用于自定义应用特定的环境中的STVS4A中间件。自定义确保与可能不同的MCU、网络和音频接口的连接。
- 应用和中间件组件在FreeRTOS™环境中运行。
- HTTP/2是与AVS服务器交互所使用的主要通信协议。
  X-CUBE-VS4A提供一个库，用于管理该协议。
- LwIP负责TCP/IP通信服务。由MbedTLS元件提供加密。
- STM32Audio为音频输入和输出以及MP3解码提供库。
- X-CUBE-VS4A扩展包是STM32Cube生态系统的一部分。它依赖其BSP和HAL驱动。

3.3 文件夹结构

*图 2*给出了X-CUBE-VS4A扩展包的文件夹结构。

图2. 项目文件结构

4 应用中的集成

本章中的各小节内容包括将X-CUBE-VS4A扩展包的内容集成到用户应用中的方式。

4.1 配置文件

STVS4A中间件依赖于其他元件，例如HTTP/2、LwIP、mbedTLS和FreeRTOS™：

- HTTP/2中间件依赖于其他三个组件。它以库的形式提供。
- *HTTP2_LWIP*文件夹中提供了库生成期间使用的LwIP、mbedTLS和FreeRTOS™配置文件作为参考。

注意： 用户不能以修改组件API的方式修改这些配置文件。

4.2 平台初始化

该应用负责平台初始化和所有依存关系。这包括诸如以下部分：

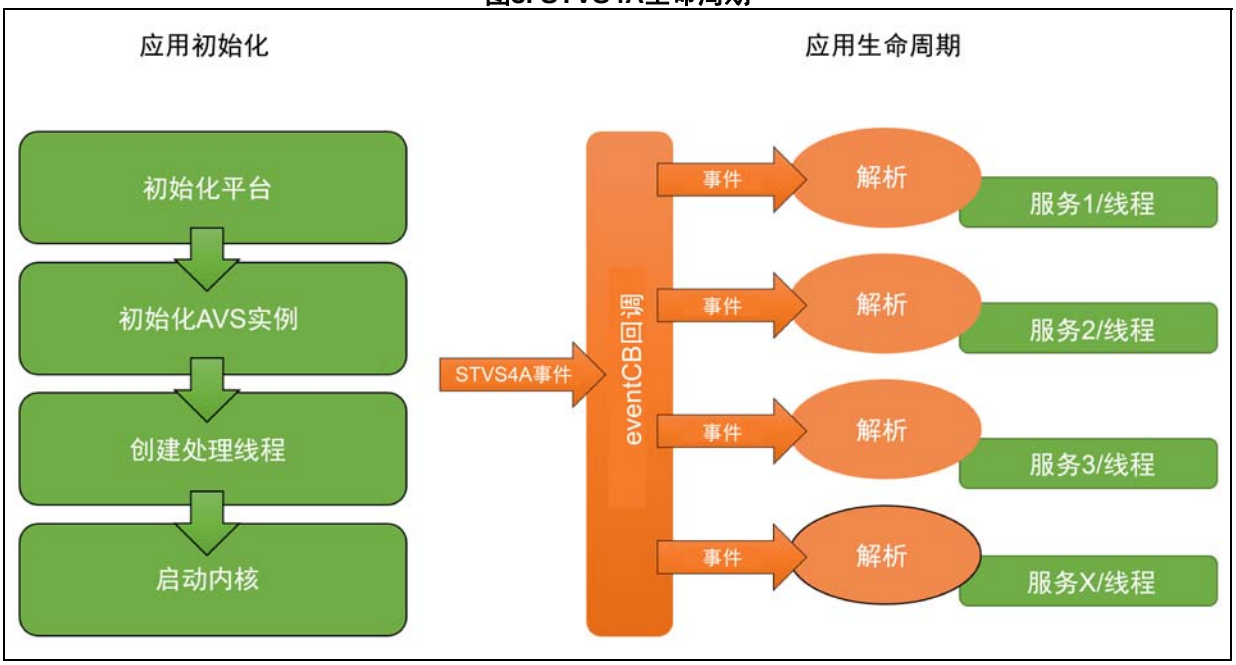
- SystemClock
- GPIO
- MPU_Config
- HAL
- Flash
- 串口

在扩展包中提供的演示应用中，*platform_init.c*代码描述了平台的基本初始化。

STVS4A没有与板初始化相关的专用活动。它假设一些必须的模块已经初始化。在平台初始化之后，STVS4A SDK也可以初始化。应用使用AVS_Create_Instance()开始STVS4A初始化。

*图 3*提供了STVS4A生命周期的概览。

图3. STVS4A生命周期



STVS4A初始化包括使用AVS帐户ID填写实例工厂和创建STVS4A实例（实例工厂是代码中定义的一个结构体，里面包含配置AVS实例的参数）。

实例工厂的大部分参数只有在自定义其标准行为时才为必填项。仅标准用例的几个参数是必填项，所有其他参数均在实例创建时填写以默认值。在设置这些参数后，STVS4A开始下一个osStartKernel()。

*表 2*所示为开始STVS4A的最少编码。

表2. 开始AVS会话的最少编码

```
platform_Init();
osInitKernel();
AVS_VERIFY(AVS_Init());
sInstanceFactory.clientId     = MY_CLIENT_ID;
sInstanceFactory.clientSecret = MY_CLIENT_SECRET;
sInstanceFactory.productId    = MY_PRODUCT_ID;
sInstanceFactory.eventCB      = appEventHandler;
AVS_VERIFY(hInstance = AVS_Create_Instance(&sInstanceFactory));
osStartKernel();
```

注意： 由于STVS4A会在应用生命周期中更新实例工厂的值，因此其必须定义为全局且不是常量，以使之总是位于RAM中。例如，在STVS4A收到IP地址后，应用会通过回调收到通知，工厂更新为有效IP地址。

使用*表 2*所示的简单初始化，Alexa能够在通过函数AVS_Set_State(hInstance, AVS_STATE_START_CAPTURE)更改STVS4A状态后进行听和说。但是，最终产品意味着更多交互。应用需要管理事件，以便支持更复杂的用例并能够与服务器和本地元件完全交互。

4.3 STVS4A事件

STVS4A事件（EVT）是来自内部状态机或Alexa服务器的通知。为了接收事件，应用需设置AVS_Instance_Factory中的eventCB字段，如*表 3*中的代码示例所示。

表3. AVS应用事件处理程序

```
AVS_Result appEventHandler(AVS_Handle handle, uint32_t pCookie, AVS_Event_t evt,
                           uint32_t  pparam)
{
  return AVS_OK;
}
...
static AVS_Instance_Factory sInstanceFactory;
  memset(&sInstanceFactory, 0, sizeof(sInstanceFactory));
  sInstanceFactory.clientId     = MY_CLIENT_ID;
  sInstanceFactory.clientSecret = MY_CLIENT_SECRET;
  sInstanceFactory.eventCB      = appEventHandler;
  AVS_VERIFY(hInstance          = AVS_Create_Instance(&sInstanceFactory));
  osStartKernel();
```

STVS4A使用EVT通过eventCB通知应用。EVT行为取决于EVT环境。返回值很重要，因为在某些环境中，它驱动引擎响应：

- AVS_OK返回值意味着一切正常且STVS4A执行其正常的处理工作。
- AVS_EVT_HANDLED返回值意味着应用已捕获事件且AVS不必继续解析事件。

以下面的事件处理用例为例：

- 当STVS4A收到来自Alexa的指令时，它首先发送EVT_DIRECTIVE，参数是JSON句柄。应用可以解析并直接管理JSON。STVS4A还解析JSON，并能发送关于该指令的更详细的EVT。例如，如果JSON是报警事件。STVS4A发送EVT_DIRECTIVE_ALERT，并且只就ALERT指令通知应用。如果应用捕获EVT_DIRECTIVE且其处理程序返回AVS_EVT_HANDLED，STVS4A将停止解析且不会收到EVT_DIRECTIVE_ALERT。

EVT十分敏感。由于它们是同步的，因此不可能在回调中长时间阻止其消息。所有EVT从STVS4A内核直接调度给用户应用。如果应用需要处理EVT中的繁重作业，它必须将作业分配给任务或创建FreeRTOS™队列。否则，STVS4A状态管理将崩溃，整体行为将受到影响。

大多数EVT是可重入的，这意味着可以从EVT中发送EVT。而有些EVT是不可重入的，以避免系统使用本地资源或事件生成其他EVT，从而使应用陷入死锁状态。不可重入的EVT从EVT_NO_REENTRANT_START开始，到EVT_NO_REENTRANT_END结束。

STVS4A套件提供多种服务，这些服务以示例的形式提供，展示了EVT管理。

4.4 STVS4A持久对象

AVS架构管理持久对象，以便保存环境，避免受到复位的影响。令牌和TLS根证书正是如此，它们是STVS4A生命周期中可能更新的对象。在进行终端用户身份验证后，从AVS服务器获取令牌并定期刷新令牌。根证书从开发者的参考API网站获取，Amazon随时可以将其撤销。STVS4A提供一种机制，用于通过应用服务保存这些对象。例如，这样可以避免在每次复位后运行整个身份验证流程。应用必须使用安全的机制，将给定信息保存在Flash存储器或其他持久存储器中。为此，将在AVS_Factory中暴露单个回调persistentCB。

persistentCB回调管理不同对象的持久存储。在各种不同对象中，至少必须持久存储令牌和根证书。由应用负责安全地管理这些信息。当应用必须保存对象时， STVS4A通过AVS_PERSIST_SET参数调用persistentCB，而当STVS4A想要检索对象时，则通过参数AVS_PERSIST_GET或AVS_PERSIST_GET_POINTER调用。应用负责以合适的方式管理这些事件，以便使用正确的对象服务API。对象采用数组的形式，其大小适合进行加载或存储。

在套件中提供的演示应用中，服务*service_persistent_storage.c*是对使用Flash存储器管理持久对象的直接描述。提供的示例以非安全方式保存在Flash存储器中。

表4. 持久存储应用服务回调

```
platform_Init();
osInitKernel();
AVS_VERIFY(AVS_Init());
  sInstanceFactory.clientId      = MY_CLIENT_ID;
  sInstanceFactory.clientSecret  = MY_CLIENT_SECRET_ID;
  sInstanceFactory.productId     = MY_CLIENT_PRODUCT_ID;
  sInstanceFactory.eventCB       = appEventHandler;
  sInstanceFactory.persistentCB = appPersistCB;
AVS_VERIFY(hInstance = AVS_Create_Instance(&sInstanceFactory));
osStartKernel();
```

4.5 服务实现

服务是一种Alexa应用。通常情况下，服务将AVS指令连接到器件，而器件使用事件将状态报告给AVS。接口包含标准接口和自定义接口，*第 4.5.1 节*至*第 4.5.6 节*对此进行了描述。

4.5.1 简单服务实现

无论如何，服务需使用eventCB实现事件解析器。实现采用switch/case的形式，捕获并处理您感兴趣的所有事件。在随套件提供的演示应用中，服务*service_serial_trace.c*描述了此类简单交互。

4.5.2 线程式服务实现

如果服务需要处理信息或更改STVS4A状态，应用需要创建一个或多个FreeRTOS™任务。这些任务能够查询或更改STVS4A状态，处理信息，并将新状态通知Alexa。

在随套件提供的演示应用中，服务*service_wakeup.c*是对改变STVS4A状态的基本线程的直接描述。在该服务中，任务等待按键或按钮操作以将STVS4A状态从*空闲*更改为*捕获*，并等待对话结束。

4.5.3 包含AVS指令/事件实现的服务

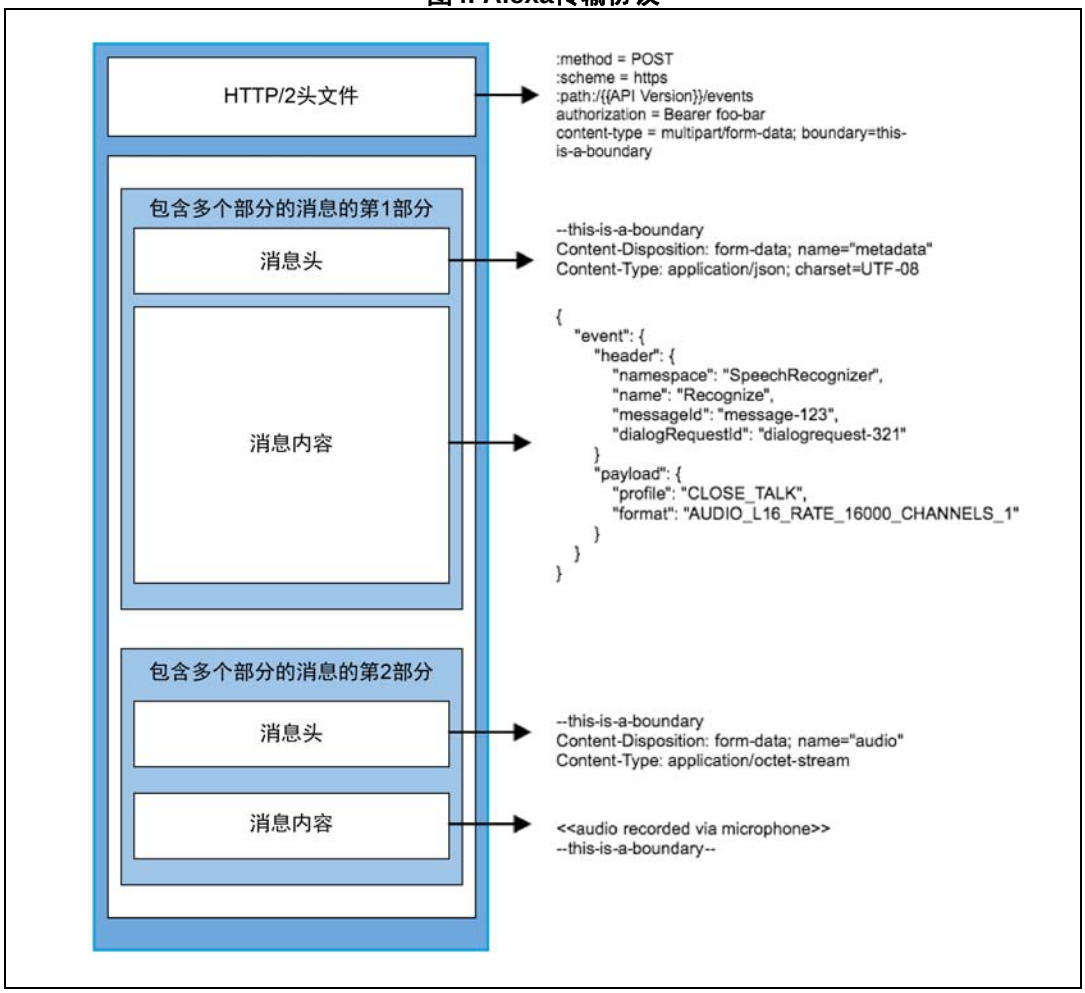
复杂用例除了与STVS4A交互，还需要与Alexa交互。此类用例更复杂，需要侦听AVS指令，并将服务状态通知服务器。

对于来自AVS的指令，使用EVT和eventCB回调进行通知。JSON句柄参数传递至应用。该参数是根（json_t *），可以进行解析以便提取有用的信息。

4.5.4 发送简单AVS事件

大多数情况下，Alexa会在指令发出后收到响应。响应采用AVS事件的形式。为此， STVS4A提供函数AVS_Send_JSon()。AVS_Send_JSon使用*图 4*所示的Alexa传输协议将单个事件发送至Alexa。

图4. Alexa传输协议



1. 此图复制自AVS网站。

AVS_Send_JSon()发送简单事件。这意味着*Multipart Message Pt2*被省略，在JSON主体之后立即关闭流。*表 5*所示为向AVS发送简单事件。

表5. 向AVS发送简单事件

```
const char *pJson = create_my_event();
   if(AVS_Send_JSon(handle,pJson) != AVS_OK)
   {
     AVS_TRACE_ERROR("Send Json Event");
   }
   free((void *)pJson);
```

4.5.5 AVS自定义事件流

如果应用需要创建自定义事件或流，STVS4A将为此提供一组特定的函数，如*表 6*所示。

表6. 允许创建自定义事件或流的函数

```
void *          AVS_Get_Http2_Instance(AVS_Handle hInstance);
AVS_HStream *   AVS_Open_Stream(AVS_Handle hHandle);
AVS_Result      AVS_Add_Body_Stream(AVS_Handle hInstance, AVS_HStream hStream,
                                    const char *pJson);
AVS_Result      AVS_Read_Stream(AVS_Handle hInstance, AVS_HStream hStream,
                                void *pBuffer, uint32_t szInSByte, uint32_t *retSize);
AVS_Result      AVS_Write_Stream(AVS_Handle hInstance, AVS_HStream hStream,
                                  const void *pBuffer, size_t lengthInBytes);
AVS_Result      AVS_Stop_Stream(AVS_Handle hInstance, AVS_HStream hStream);
AVS_Result      AVS_Close_Stream_(AVS_Handle hInstance, AVS_HStream hStream);
AVS_Result      AVS_Process_Json_Stream(AVS_Handle hInstance, AVS_HStream hStream);
const char_t *  AVS_Get_Reponse_Type_Stream(AVS_Handle hInstance,
                                            AVS_HStream hStream);
```

创建自定义事件的典型序列采取伪代码的形式，如*表 7*所示。

表7. 事件创建伪代码

```
/* main HTTP/2 instance */
void *hClient=AVS_Get_Http2_Instance(hInstance);
/* Create the stream & add HTTP2/2 header */
AVS_HStream *hStream = AVS_Open_Stream(hInstance,hClient);
/* Create the Multipart PL1 and send the message content */
AVS_Add_Body_Stream(hInstance,hStream,create_my_event());
/* Prepare multiPart-PL2 */
/* Customize push/pull payload */
AVS_Read_Stream(hInstance,hStream,...);
AVS_Write_Stream/Pull(hInstance,hStream,...);
....
/* End multiPart-PL2 */
AVS_Stop_Stream(hInstance,hStream);
```

4.5.6 管理同步事件状态

某些事件或服务需要向Alexa发送同步状态。同步状态是所有服务环境的报告。STVS4A提供函数AVS_Post_Sychro()，强制引擎尽快发布状态。可以随时从STVS4A内核请求同步状态。同步状态必须反映所有服务的准确环境。

此外，STVS4A提供一个机制，将自定义环境添加到同步状态。在发送同步状态之前和创建JSON头文件之后，STVS4A发送EVT_SEND_SYNCHRO_STATE消息，以根（json_t*）为参数。每个服务都会收到通知，可以自由地更新contex JSON数组，并在通过STVS4A内核向Alexa发送同步事件之前添加其自己的事件。

如果必须通过函数AVS_Send_JSon()向Alexa发送事件，则必须为每条消息添加环境状态。为此，在将(json_t *)转换为字符串之前使用函数AVS_Json_Add_Context()。

在随套件提供的演示应用中，服务*service_alarm.c*展示了EVT_SEND_SYNCHRO_STATE管理示例。

4.6 JSON和JANSSON

实现服务的主要作业是解析和创建AVS JSON事件。为此，STVS4A使用JANSSON Open Source API管理JSON字符串。

如需关于JANSSON API的更多信息，请访问http://www.digip.org/jansson/。

该API易于使用且内存占用量低。在向应用发送指令事件时， STVS4A总是给出根句柄JANSSON。然后，比较容易从该句柄获取JSON字符串，如*表 8*所示。

表8. JANSSON字符串提取

```
AVS_Result service_alarm_event_cb(AVS_Handle handle, uint32_t pCookie, AVS_Event_t
evt, uint32_t  pparam)
{
If (evt==EVT_DIRECTIVE_ALERT)
    {
    // the json is the parameter
    json_t *root = (json_t *)pparam;
    const char *pJson = json_dumps(root, 0);
    AVS_TRACE_INFO(" %s ", pJson) ;
    }
...
}
```

创建JSON事件是服务中的常见任务。*表 9*中的代码段提供了这样的创建示例。

表9. JSON事件创建示例

```
//{
//"event": {
//    "header": {
//        "namespace": "Alerts",
//        "name": "SetAlertSucceeded",
//        "messageId": "{{STRING}}"
//    },
//    "payload": {
//        "token": "{{STRING}}"
//    }
//}
// send an alarm event to AVS
static    void service_alarm_event(AVS_Handle handle,const char *pToken, const char
*pName)
{
   uint32_t err=0;
   json_t *root  = json_object();
   json_t *event = json_object();
   json_t *header = json_object();
   json_t *payload = json_object();
   static char_t msgid[32];
   sprintf(msgid, FORMAT_MESSAGE_ID, messageIdCounter++);
   err |= json_object_set_new(header, "namespace", json_string("Alerts"));
   err |= json_object_set_new(header, "name", json_string(pName));
   err |= json_object_set_new(header, "messageId", json_string(msgid));
   err |= json_object_set_new(payload, "token", json_string(pToken));
   // links
   err |= json_object_set_new(root, "event", event);
   err |= json_object_set_new(event, "header", header);
   err |= json_object_set_new(event, "payload", payload);

   /* Add the context state to the event */
   AVS_VERIFY(AVS_Json_Add_Context(handle, root));

   const char *pJson = json_dumps(root, 0);
   json_decref(root);
   if(AVS_Send_JSon(handle,pJson) != AVS_OK)
   {
     AVS_TRACE_ERROR("Send Json Event");
   }
   free((void *)pJson);
}
```

解析JSON事件也是一项易于执行的任务，例如在从Alexa接收到指令时。*表 10*中的代码段对此类解析进行了描述。

表10. JSON事件解析

```
 //"directive": {
    //    "header": {
    //        "namespace": "Alerts",
    //        "name": "SetAlert",
    //        "messageId": "{{STRING}}",
    //        "dialogRequestId": "{{STRING}}"
    //    },
    //    "payload": {
    //        "token": "{{STRING}}",
    //        "type": "{{STRING}}",
    //        "scheduledTime": "{{STRING}}",
    //        "assets": [
    //            {
    //                "assetId": "{{STRING}}",
    //                "url": "{{STRING}}"
    //            },
    //            {
    //                "assetId": "{{STRING}}",
    //                "url": "{{STRING}}"
    //            },
    //        ],
    //        "assetPlayOrder": [ {{LIST}} ],
    //        "backgroundAlertAsset": "{{STRING}}",
    //        "loopCount": {{LONG}},
    //        "loopPauseInMilliseconds": {{LONG}}
    //    }
    //}
    // the directive EVT_DIRECTIVE_ALERT manages timer and alarms
    // we can overload this event to add and delete items

  case EVT_DIRECTIVE_ALERT:
    {
      // the json is the parameter
      json_t *root = (json_t *)pparam;
      // parse it to extract information
      json_t *directive = json_object_get(root, "directive");
      json_t *payload    = json_object_get(directive,"payload");
      json_t *header    = json_object_get(directive,"header");
      json_t *scheduledTime = json_object_get(payload,"scheduledTime");
      json_t *token = json_object_get(payload,"token");
      json_t *type = json_object_get(payload,"type");

      json_t *name    = json_object_get(header,"name");
      const char_t *pToken = json_string_value(token);
      const char_t *pTime = json_string_value(scheduledTime);
      const char_t *pType = json_string_value(type);
...
```

4.7 调试JSON

STVS4A提供一种机制，用于转存JSON事件和指令。STVS4A可以在串行控制台上转存JSON脚本。该选项默认为禁用。使用以下代码段使能该选项：

```
AVS_Set_Debug_Level(    AVS_TRACE_LVL_DEFAULT |
                        AVS_TRACE_LVL_JSON |
                        AVS_TRACE_LVL_JSON_FORMATED);
```

JSON脚本的可视化有两种选择，Alexa总是以压缩模式发送或接收脚本：

- 使用AVS_TRACE_LVL_JSON，JSON脚本按其发送或接收时的原样显示
- 使用AVS_TRACE_LVL_JSON_FORMATED，STVS4A将格式化JSON脚本以使之可读

注： *使用AVS_TRACE_LVL_JSON_FORMATED时需要时间进行打印和格式化，并且如果同时有太多trace，可能会发生干扰。*

5 应用程序示例

本章介绍类似智能音箱的应用。

注： *该应用不实现智能音箱的整套功能。*

5.1 应用描述

用户可以使用该应用与Amazon的Alexa对话。

用户可以要求Alexa唱歌，听新闻，检查天气预报，控制智能家居设备等。

5.2 Alexa Voice Service帐户

可以使用两种帐户：

- 开发者需使用Amazon帐户才能在Alexa服务器上创建其设备。请访问 https://developer.amazon.com了解详情。
  提供的应用示例以设备标识符和*secrets*为例。
- 该应用的最终用户使用Amazon帐户登录（请参考*第 5.3节：网络设置和身份验证*）并使用应用。

注： *在开发阶段，开发者和最终用户帐户可以是相同的。*

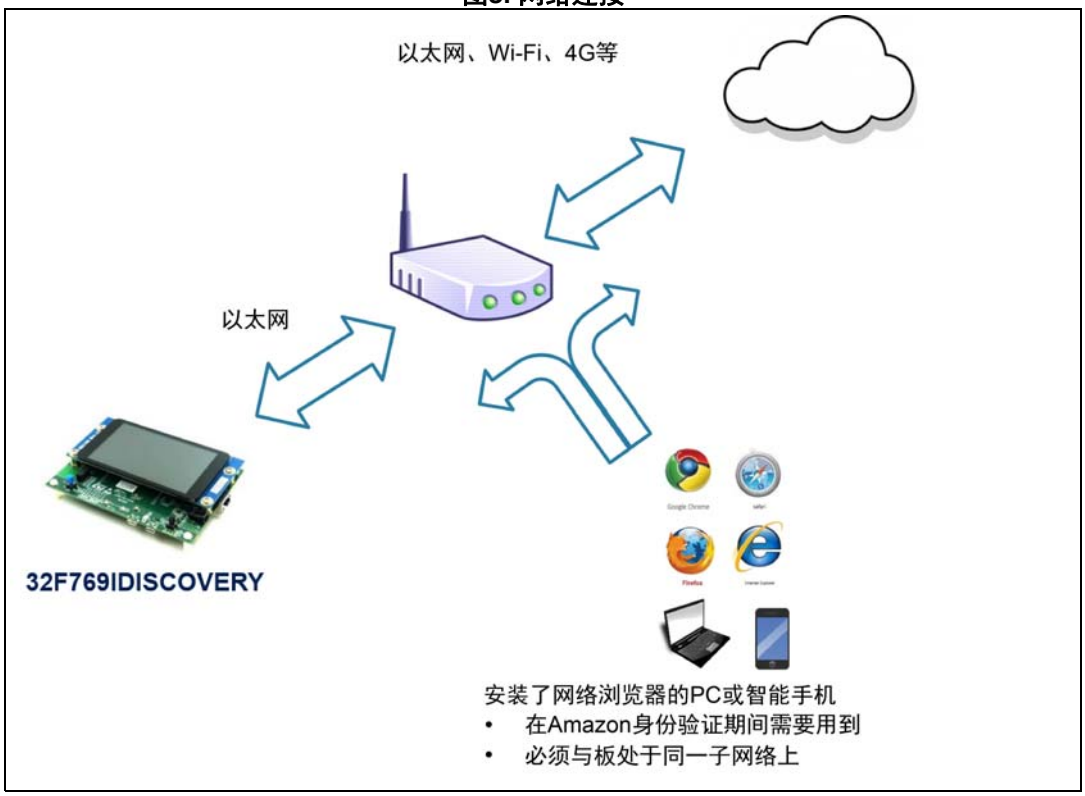
5.3 网络设置和身份验证

为了运行该应用，必须将板插入可以直接访问互联网（无代理）的路由器。

路由器必须配置为DHCP服务器。该应用需要一个DHCP服务器，用来设置其IP地址。

一台PC或智能手机，在身份验证阶段使用。*图 5*所示为网络和身份验证环境。

图5. 网络连接



身份验证阶段的序列如下：

1. 从PC或智能手机连接到STM32板内部网络服务器
   – http://xxx.xxx.xxx.xxx，其中xxx.xxx.xxx.xxx是板IP地址。初始化阶段结束后，板IP地址显示在用户界面上
2. 在Navigator重定向至Amazon登录页面时，提交有效的Amazon凭证
3. 然后，Navigator尝试从以*http://stvs4a/grant_me?code=*开头的长URL加载内容。用板的IP地址替换URL中的*stvs4a*字符串并发送请求
4. 连接后，浏览器显示一条消息，告知已成功获得令牌，并显示令牌的第一个字符
5. 此时，板保存连接到AVS服务器所需的信息，即使在下一次复位后。此信息不是Amazon凭证

注意： 上述例子不考虑安全问题，假设在可信环境中运行。对于最终产品，用户必须开发并落实与产品环境相匹配的安全机制。

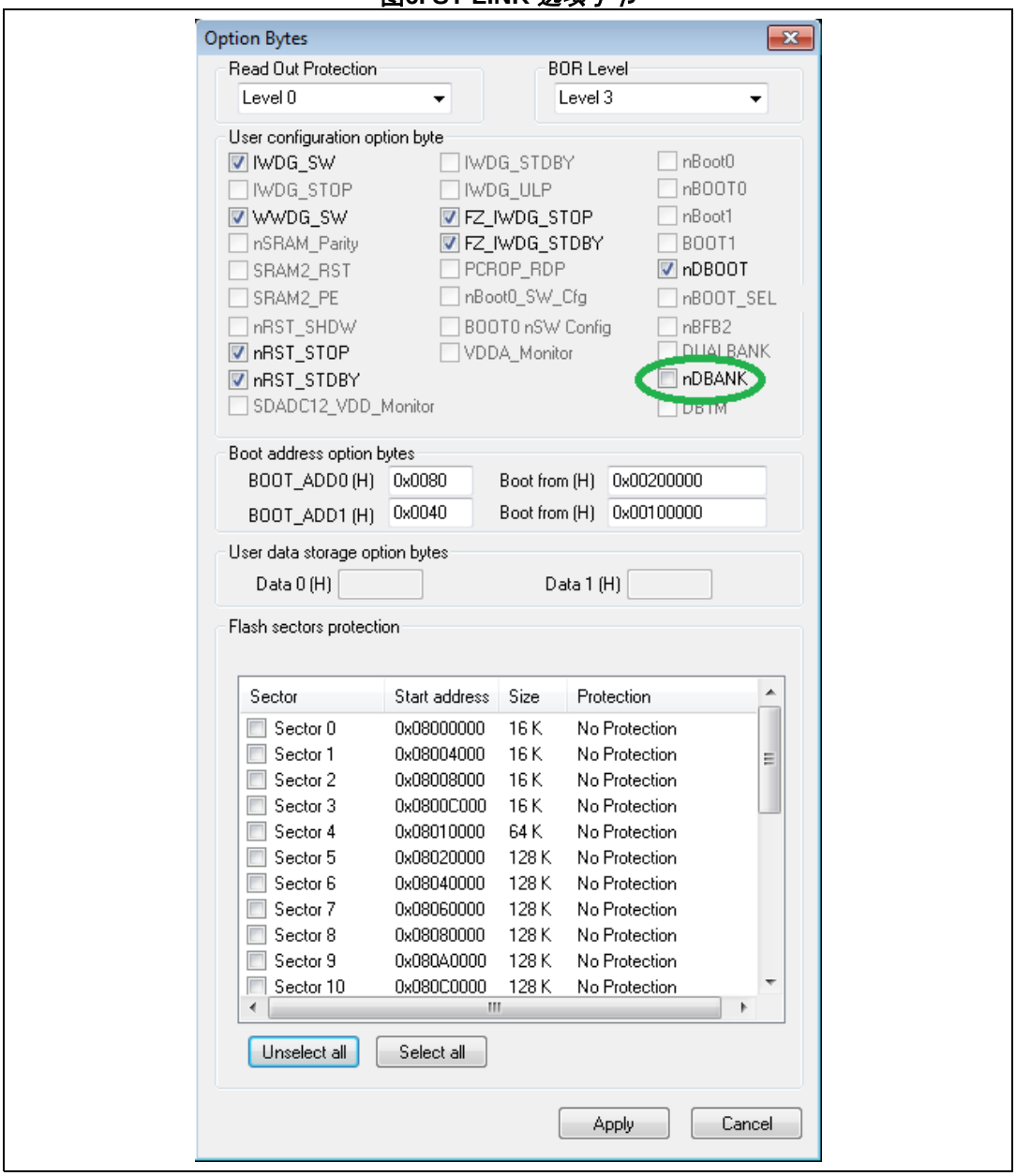
5.4 Flash编程

该应用将Flash用作双存储区。通过选项字节配置该模式，步骤如下：

1. 运行ST-LINK工具
2. 使用*目标*菜单连接到板
3. 使用*目标*菜单显示*选项字节*窗口
4. 取消选中*nDBANK*并应用更改（如有必要）

*图 6*显示ST-LINK的*选项字节*窗口和*nDBANK*复选框的位置。

图6. ST-LINK *选项字节*



应用中嵌入了可靠性测试套件。这些测试需要Flash中保存的一些数据。在大多数编译配置中，数据都保存在外部*QSPI* Flash中。在这种情况下，调试器无法烧录这部分数据，需通过ST-LINK按以下操作顺序对可执行文件进行编程：

1. *目标菜单中的擦除芯片命令*
2. *文件菜单中的打开文件命令*
3. *目标菜单中的编程和验证命令*

注： *如果测试代码相关数据没有烧录到Flash，“测试开始”按钮将不工作，但其他功能可以工作。*

5.5 使用应用

当应用初始化时，可使用类似于下面的问题与Alexa交互：

- “《哈利波特》是什么时候出版的？”
- “东京现在是几点？”
- “Chuck Norris多大年纪？”

还可以使用诸如以下请求使用闹钟和定时器：

- “设置一个上午07:00的闹钟！”
- “取消闹钟”
- “设置一个两分钟的定时器”
- “设置一个闹钟”

当请求不包含设置闹钟所需的所有详细信息时，Alexa将要求提供详细信息。

通过下面的指令可以控制板子音频输出的音量大小：

- “提高音量
- “降低音量”
- “将音量设置为3”

不同于通常为精确应用而设计的实际产品，该应用示例同时支持基于按钮的模式*点击说话*和*即按即说*。*即按即说*模式也称为*按住*模式。用户可以在触摸屏用户界面上选择模式。

蓝色按钮用于交互：

- 在*点击说话*模式下，点击并询问将触发Alexa的即时响应
- 在*按住*模式下，按下并询问将触发Alexa的即时响应
- 点击按钮可停止闹钟蜂鸣器

触摸屏显示两页式用户界面。

用户界面的两个页面均显示：

- 在应用运行时跳动的心脏
- 版本信息
- 模式按钮
  - 默认模式为*点击说话*
  - 按下该按钮切换至*即按即说*（*按住*）模式
  - 由于该应用中没有集成唤醒词检测，*语音启动*将充当*点击说话*
- 板的IP地址
- 内部状态（在准备运行时为*空闲*）
- 日期
- 用户界面页码，用于在两个页面之间切换

此外，第一个用户界面页面显示以下信息：

- 当已通过语音命令设置闹钟时，显示闹钟状态
- 音频播放器在音乐回放期间显示状态信息

相对于常用集合，第二个用户界面页面显示以下附加信息和按钮：

- 可靠性测试状态
  注意：当没有足够快地提供预期答案时，可能发生超时。
- *测试开始*按钮：启动可靠性测试套件（数据必须已保存到Flash中）
- *设置链路上行/下行*按钮：模拟以太网链路损耗
- *网络模拟关闭*按钮：从耐久性测试中去掉网络损耗模拟

注： *某些编译配置中没有第二个用户界面页面。*

5.6 可靠性测试

该应用附带可靠性测试工具，该工具以服务的形式实现。

可靠性测试可通过触摸屏上用户界面第二页上的按钮启动。在用户按下开始测试按钮后，将执行预定义测试列表。在*源文件service_endurance_test.c*中管理测试顺序。

可靠性测试依赖于保存在Flash存储器中的资源。请参考*第 5.4 节*了解关于Flash编程的详细信息。

在可靠性测试期间，从音频文件将语音命令（请参考文件*service_assets.c*）推送至输入通道，而不是通过BSP进行麦克风采集来获取。

可靠性测试期间还会模拟网络断开，导致在特定测试运行时显示错误通知和网络重新初始化。

注： *测试引擎将一直循环，直至停止测试按钮被按下。*

5.7 获取类似于printf的trace

当板连接到PC时，将在PC上创建虚拟COM端口以模拟串行端口。

STVS4A堆栈嵌入了类似于printf的宏，以便向连接的PC发送调试信息。为了在PC上获取这些trace，必须将串行端口配置为：

- 实际的COM端口号
- 921600波特率
- 8 位数据
- 无校验位
- 1个停止位
- 无流控

*图 7*和*图 8*显示了虚拟COM端口设置。

图7. 虚拟COM端口选择

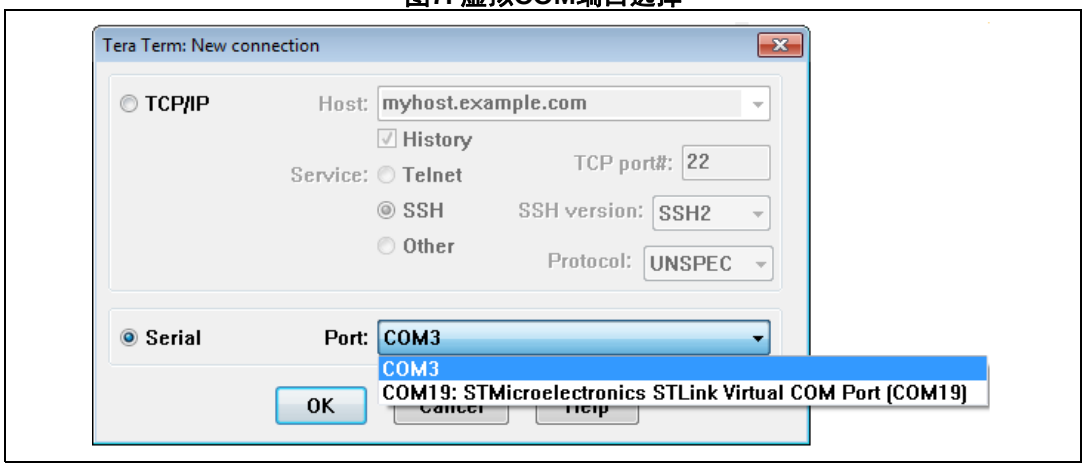
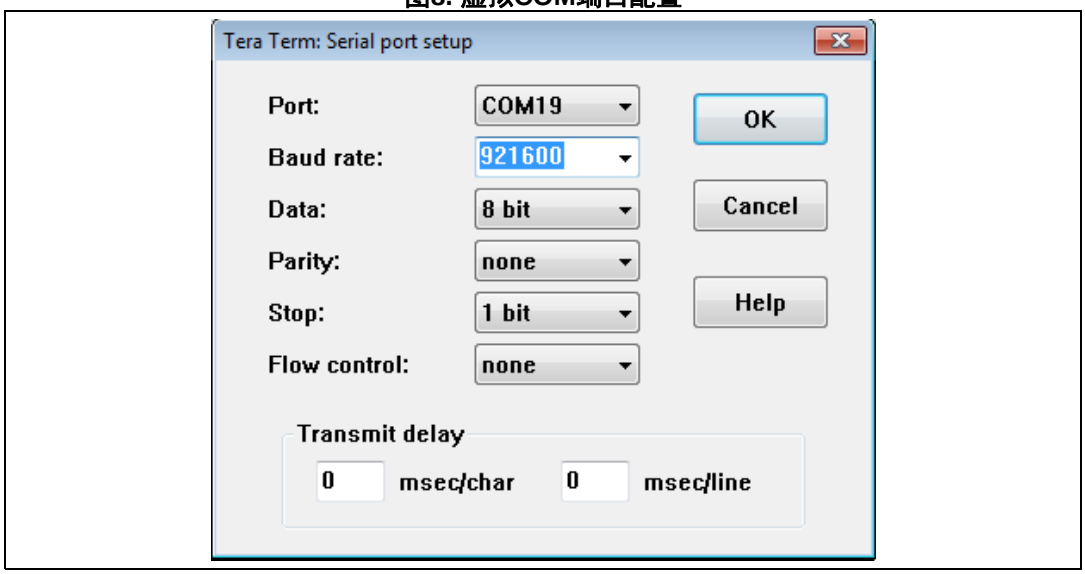


图8. 虚拟COM端口配置



6 版本历史

表11. 文档版本历史

日期	版本	变更
2018 年 3 月 20 日	1	初始版本。
2018年6月5	2	更新了*第 5.4节：Flash编程*。将STM32Cube扩展包的名称更新为X-CUBE-VS4A。将语音服务中间件名称更新为STVS4A。

表12. 中文文档版本历史

日期	版本	变更
2019年 1月 3日	1	中文初始版本。

重要通知 - 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。