Application Notes
ZHCAA33 – 9.2020

基于阿里云和 TIDL 的智能网关的研究

Denny Yang / Zongqi Hu　　TI 嵌入式处理器技术支持工程师

摘要

随着物联网的发展，越来越多的设备具备了联网功能，而当大量的设备（尤其是摄像头这样传输数据量大的设备）接入网络时，对网络服务器和云服务器的负载都是巨大的，同时数据传输带来的延迟也是明显的。同时，随着工厂自动化程度的提高，传感设备数量的增大，如何有条理的管理传感器设备，如何高效的处理传感器上传的数据并及时做出正确的反应，是很多自动化工厂都需要解决的问题。解决问题的有效途径之一，便是赋予近数据源网关必要的数据处理能力，完成任务划（分成简单任务和复杂任务），并处理简单任务。简单任务由网关进行处理，能大大减少数据传输带来的延迟，而网关无法处理的复杂任务再交由云端处理。阿里云是阿里巴巴集团下的云计算产品，提供卓越的云计算服务与技术。TIDL 是德州仪器公司的开源深度学习软件包，受 AM57xx Sitara 设备的支持（本文使用的是 AM5749），适合处理一维、二维、三维数据，并支持将计算任务从 ARM 核上转移到硬件加速器上（EVE 和 C66x DSP），使 ARM 核能够从深度学习的任务中释放出来，能够继续执行其他非计算型任务。本文间介绍阿里云和 TIDL 的环境搭建以及如何由阿里云和 TIDL 构建一个云-边缘联合计算系统。

目录

1 系统总体介绍

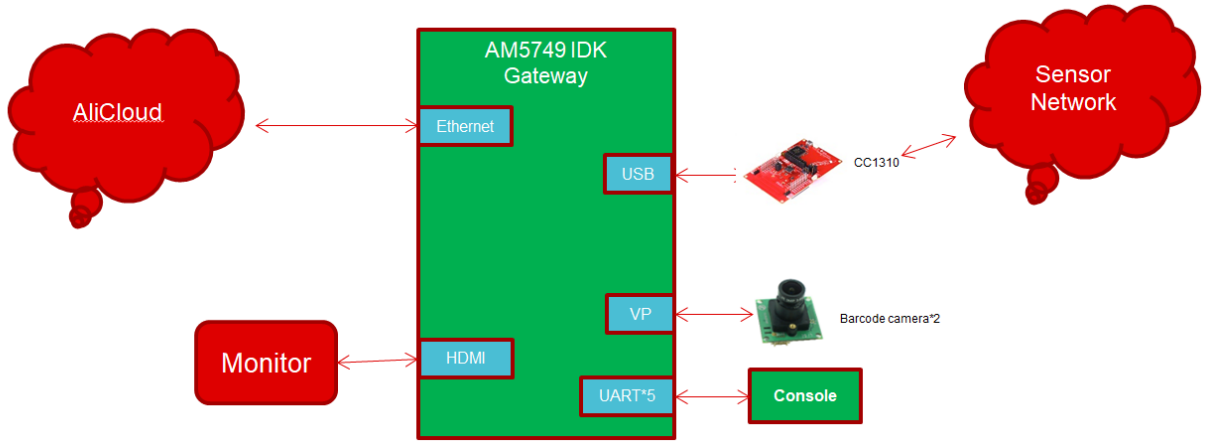

图 1：系统构架

本系统由本地 AM5749 IDK 开发板，加上 CC1310 无线传感器网络和摄像头组成，远程连接阿里云。本地是边缘端，通过 AM5749 强大的人工智能运算能力，进行摄像头人工智能识别，把识别结果发送到云端做进一步处理。同时 AM5749 连接 CC1310 传感器网络，AM5749 可以当作边缘网关使用，可以收集、存储、分析、上传传感器网络的数据。本例的一个典型应用可以在智能工厂里通过摄像头进行人工智能分析，产品缺陷检测，同时连接传感器网络对整个车间进行边缘计算管理。对比传统方案，本方案的优势是集成度更高，可以单芯片解决 AI 视频和传感器网络两个问题。得益于 AM5749 是一个包含 ARM/DSP/AI 加速器的高集成 SOC。

2 阿里云简介

阿里云是全球领先的云计算服务及人工智能公司，服务着制造、金融、政务、交通、医疗、电信、能源等众多领域。其支持的服务有比如：弹性计算、存储、云通信、移动云、物联网、大数据等十几个大类，各个大类下又有数个具体服务。本文主要使用了物联网下的物联网边缘计算服务，在存储上使用了存储的对象存储 OSS 服务。

2.1 物联网边缘计算：

阿里云提供了一个功能完备的物联网控制台和网关软件包，用来设置管理连接的网关和网关连接的设备，以及使用阿里云提供的物联网服务。本文涉及几个主要专用词：产品、设备、边缘实例、驱动。

产品：定义一类产品，类似于 C++中的类。

设备：每一设备都属于一类产品。类似于 C++中的对象。说明：为做区分，在本文中虚拟设备均指云端上的设备，实际设备均指边缘端的设备。

边缘实例：边缘计算中的基本单位，由一个网关、多个设备，以及其他阿里云提供的服务组成。通过部署实例，将实例中的资源、设置部署到网关中。

驱动：与设备对应，由阿里云的使用者（笔者）开发。面向云端，完成设备上线、数据上报、接收命令等功能，面向边缘端，其功能由使用者自定义：管理实际设备的连接、设备数据的接受与发送等。

阿里云发布了多个版本的边缘网关软件包，本文使用的是标准版 v1.8.1-ARMv7 软浮点型。

为了减轻开发者开发设备驱动的工作量，阿里云提供了一套用于云端与边缘端通信的 SDK 供开发者使用。

2.2 对象存储 OSS：

该服务是阿里云提供的云存储服务，适合于图片和音视频文件的海量存储。为方便使用者管理 OSS，操作存储空间和对象，阿里云 OSS 服务提供了 Web 服务页面。同时也提供了丰富的 API 接口和各种语言的 SDK 包，方便使用者通过调用 SDK 包提供的 API 来编写满足使用需求的程序，灵活的管理 OSS。

涉及的基础概念如下：

存储类型：OSS 提供了三种存储类型，依据存储时间和存储频率划分，本文采用标准存储类型，其提供高可靠、高可用、高性能的对象存储服务，并且支持频繁的访问数据。

存储空间（Bucket）：用来存储对象的容器，所有对象都必须隶属某个存储空间。

对象/文件（Object）：基本的数据单元。

地域（Region）：OSS 的数据中心所在物理位置。

访问域名（Endpoint）：指向一个具体的存储空间，访问该存储空间的访问域名。

访问密钥（AccessKey）：用来验证访问者是否具有访问 OSS 存储空间的权限。

3 TIDL 简介

目前 TIDL 软件包仅支持使用事先训练好的模型进行卷积神经网络推理，而不需要在目标设备上进行训练。理论上，使用通过 Caffe 或者 Tensorflow-slim 框架训练的模型能够直接高效的导入到 TI 的设备中。但是，由于 TIDL 的设计之初是为了在功率受限的嵌入式系统上进行卷积神经网络推理，因此 TIDL 并不支持所有能够在 Caffe 或者 TF 框架上跑起来的拓扑结构，而只支持一些特定的层和拓扑结构。这些层和拓扑结构既不是计算密集型，也不要求非常大的存储空间。到目前为止，TIDL 支持的拓扑结构有：

•Jacinto11（类似于 ResNet10），分类网络

•JSeg21，像素级分割网络

•JDetNet（类似于 SSD-300/512），目标检测网络

•SqueezeNet

•InceptionV1

•MobileNetV1

TIDL 支持的层类型如下：

Convolution Layer、Pooling Layer（平均池化和最大池化）、ReLU Layer、Element Wise Layer、Inner Product Layer、Soft Max Layer、Bias Layer、Deconvolution Layer、Concatenate Layer、ArgMax Layer、Scale Layer、PreLU Layer、Batch Normalization Layer、ReLU6 Layer、Crop Layer、Slice Layer、Flatten Layer、Split Layer、Detection Output Layer

TIDL 对一些术语做了定义：

1. Computer core

   指一个硬件加速器，EVE 或者 DSP。一个执行对象运行在一个 Computer core 上。本文使用的 AM5749 有四个 Computer core：EVE1，EVE2，DSP1，DSP2。

2. Configuration

   TIDL 提供的一个类。用来为处理器类的指定包括指向网络和二进制参数文件的指针一类的配置信息。

3. ExecutionObject (EO)

   TIDL 提供的一个类。用来管理每一个层组（LG）在一个 Computer core 上的执行。与 Computer core 之间存在一一对应关系。借助 OpenCL 来管理执行和分配网络处理任务。

4. ExecutionObjectPipeline (EOP)

   TIDL 提供的一个类。通常在两种情况下使用：

   - 一个帧在多个 Execution Object 上，采用流水线的方式进行处理；
   - 采用双缓存（输入输出使用独立的缓存区）执行时；

5. Executor

   TIDL 提供的一个类。用来为 Execution Object 根据 Configuration 类进行初始化。同时也负责对 OpenCL 的初始化工作。

6. Frame

   一个二维数据存储区，比如图片。

7. Layer

   每一层均由一些数学运算组成，比如：过滤器、整流线性单元(ReLU)运算、下采样运算(通常称为平均池、最大池或跨行)、元素加法、连接、批处理标准化和完全连接的矩阵乘法。

8. Layer Group（LG）

   一个互相有关联的层的集合，形成执行的基本单元。Execution Object 负责把一个层组放到 Computer core 上运行。

9. Network

   由深度学习中的层和层之间的关联组成。它通过 TIDL 导入工具生成，并通过 TIDL API 供开发者使用。

为了提高帧处理的工作效率，TIDL 提供了 3 种使用示例：

1. 每一个 EO 处理一个 frame
2. 多个 EO 处理一个 frame
3. 使用双缓存方案

每一个 EO 处理一个 frame

此时网络被设置成一个单层组，卷积神经网络中的所有层被分到一个层组中，由一个 Execution Object 完成处理一个 frame 所需要的所有工作。

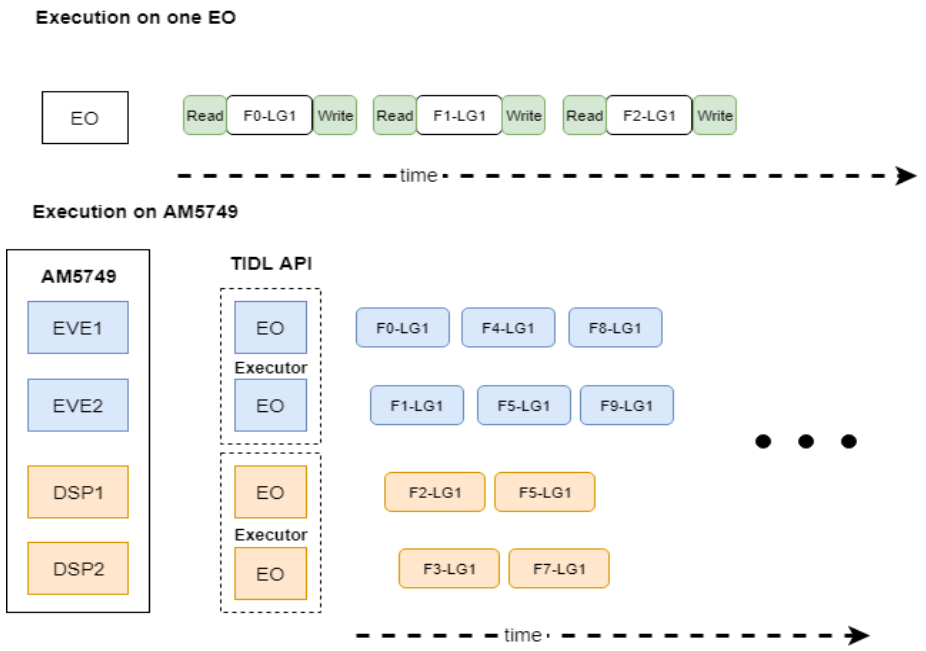

图 2：每一个 EO 处理一帧；Fn：帧数；LG：层组（Layer Group）

多个 EO 处理一个 frame

这通常用来减少单一 frame 的处理延迟时间。在该神经网络中，某些层在 C66x DSP 上的运行会比在 EVE 上更快，比如：SoftMax Layer 和 Poolling Layer，而其他层在 EVE 上运行更快。因此将卷积神经网络的层分到两个层组中，由 C66x DSP 和 EVE 来处理对应层组。

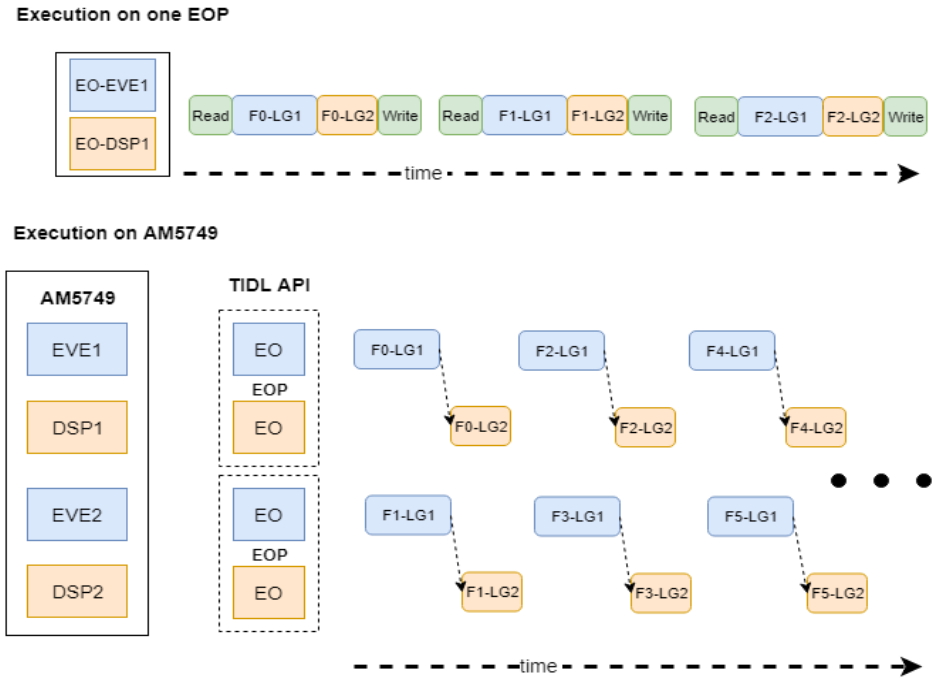

图 3：多个 EO 处理一帧；Fn：帧数；LG：层组（Layer Group）

使用双缓存方案

使用双缓存的方法能够减少在数据输入输出延迟，抽象出更多的 Execution Object Pipeline，进一步减少连续处理帧的总延迟。

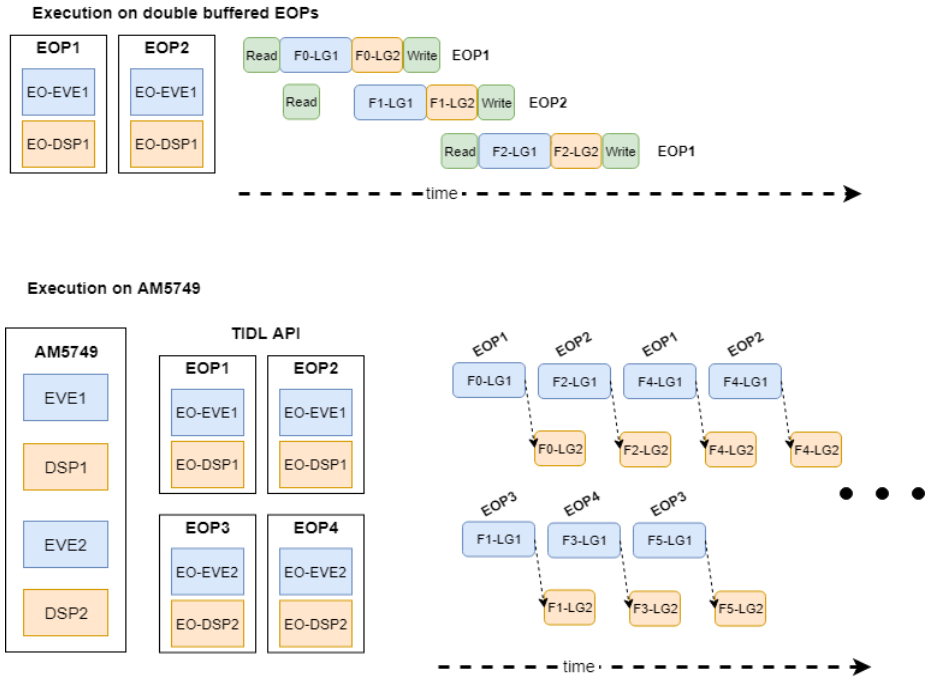

图 4：使用双缓存的 EOP 优化；Fn：帧数；LG：层组（Layer Group）

4 云端与边缘端环境搭建

云端环境搭建：

我们需要在云端创建相应的虚拟设备来与边缘端对应：网关产品及其设备，摄像头产品，设置产品属性，并创建其设备。创建边缘实例，将创建的网关设备、摄像头设备分配个该网关，并为摄像头产品分配适当的驱动，通过部署边缘实例把驱动部署到网关设备上。

边缘端环境搭建：

作为网关设备的 AM5749 EVM 板即为边缘端，为了使 TIDL 和阿里云的对应组件在边缘端成功运行，需要完成以下步骤：

1. 一张容量至少 8GB 的 SD 卡，并在其内安装 TI 的 SDK（linux procto OS）；
2. 安装阿里云提供的 Link IoT Edge 软件包；
3. 安装 Link IoT Edge 的 C SDk，此 SDK 为设备驱动的依赖组件；
4. 安装 Samba 服务器（可选），以方便使用 EVM 板的文件系统，方便查看运行日志；
5. 安装阿里 oss C-SDK；
6. 将事先设计实现的设备驱动、TIDL app 拷贝到相应的位置，并编译；
7. 创建运行必要的文件夹；

云端环境搭建和边缘端环境搭建的详细步骤请参见：

BLOG：基于 Sitara 的 ali-iot 边缘网关设置方法（请联系作者索取）

5 驱动程序的设计与实现

阿里云提供了一套方便开发者实用的 SDK，这大大减少开发者完驱动程序开发的工作量，为了能够很好的利用这一套 SDK，首先我们需要熟悉其提供的 API。

阿里云的 C 版本 SDK 的源码里，在函数声明时，阿里云提供了十分详细的注释，这极大的降低了代码阅读的难度。API 的头文件为<SDK-path>/include/leda.h，详细的定义 C 文件为<SDK-path>/src/leda.c，或者可以参见 github：https://github.com/aliyun/link-iot-edge-access-sdk-c?spm=a2c4g.11186623.2.26.237b2377wjVoZW

在此，仅对重要 API 做简单的介绍。

API 如下：

1. get_properties_callback

完整的函数定义为：

```
typedef int ( *get_properties_callback ) ( device_handle_t dev_handle,
```

```
                                        leda_device_data_t properties[],
                                        int properties_count,
                                        void *usr_data );
```

此 API 并没有声明一个具体的函数，而是给出了一个函数定义。

该 API 可以理解为开发者提供给 SDK 的一个接口，使 SDK 能够主动获取 dev_handle 对应实际设备的信息。其内部的功能逻辑取决于实际的使用情况，需要开发者自行完成。

SDK 定义这样一个函数，而非声明一个函数，这样做的好处是很明显的：即满足了开发者自定义函数名的需求，在 SDK 内部使用函数指针来代替函数名，不受开发者自定义函数名的影响，简化了代码，增加了 SDK 的通用性。

2. set_properties_callback
   完整的函数定义为：

```
typedef int ( *set_properties_callback ) ( device_handle_t dev_handle,
                                        const leda_device_data_t properties[],
                                        int properties_count,
                                        void *usr_data );
```

   与上一个定义类似，该 API 可以理解为开发者提供给 SDK 的一个接口，使 SDK 能够设置与 dev_handle 对应实际设备的属性。其内部的功能逻辑取决于实际的使用情况，需要开发者自行完成。

3. call_service_callback
   完整的函数定义为：

```
typedef int ( *call_service_callback ) ( device_handle_t dev_handle,
                                        const char *service_name,
                                        const leda_device_data_t data[],
                                        int data_count,
                                        leda_device_data_t output_data[],
                                        void *usr_data );
```

   同上。SDK 需要调用某个设备的服务时，SDK 通过改接口传递参数。具体服务的应用程序需要开发者自行完成。

4. leda_register_and_online_by_local_name
   完整的函数定义为：

```
device_handle_t leda_register_and_online_by_local_name (
                                            const char *product_key,
                                            const char *local_name,
                                        leda_device_callback_t *device_cb,
                                            void *usr_data );
```

   虽然用来定义一个设备的信息有两个 product_key 和 device_name，但是用一产品下的所有设备的 product_key 相同，因此只需要确认 local_name 即可。
   实现功能：通过本地设备名称，注册并上线设备，申请设备唯一标识符。
   说明：为了减少开发工作量，设备的注册在云端控制台上完成，仅使用该函数完成设备上线。

5. leda_register_and_online_by_device_name
   完整的函数声明：

```
device_handle_t leda_register_and_online_by_device_name (
                                                const char *product_key,
                                                const char *device_name,
                                            leda_device_callback_t *device_cb,
                                                void *usr_data );
```

实现功能：通过本地设备名称，注册并上线设备，申请设备唯一标识符。
其用法遇上一个函数相同，两者内部实现基本一致，仅仅只是用于指定设备的值不同。

6. leda_online
   完整函数声明：

```
int leda_online ( device_handle_t dev_handle );
```

   上线对应的设备，设备只有上线后，才能被 LinkEdge 识别。
   阻塞接口,成功返回 LE_SUCCESS, 失败返回错误码。
7. leda_offline
   完整函数声明：

```
int leda_offline ( device_handle_t dev_handle );
```

   与前者相反，下线对应设备（断开设备与云的连接）。可以在本地设备出现异常时使用，避免传输错误信息到云端。
   阻塞接口,成功返回 LE_SUCCESS, 失败返回错误码。
8. lede_report_properties
   完整函数声明：

```
int leda_report_properties ( device_handle_t dev_handle,
                             const leda_device_data_t properties[],
                                                int properties_count );
```

   上报属性，可以一次上报一个，或者一次上报数个。
9. lede_report_event

```
int leda_report_event ( device_handle_t dev_handle,
                                                const char *event_name,
                                        const leda_device_data_t data[],
                                                int data_count );
```

   上报事件，可以一次上报一个，或者一次上报数个。

驱动的设计与开发

1. 设计思路：
   a. 概述：由于阿里云的 C SDK 处在不断更新的状态、再加上边缘网关下连接的设备多样性，因此将整套系统拆分成三部分：由边缘网关 linkedge 拉起的 SDK 驱动程序 A；负责从边缘设备接受数据、发送指令到边缘设备的边缘网络驱动程序 B；作为前两者数据的中转站，以及边缘网络中设备的驱动程序 C，负责转发数据以及管理边缘设备的上、下线。
   b. 数据存储：为保证云端虚拟设备与边缘网络实际设备之间对应关系在时间上的连续性，实际设备下线后重新上线连接的虚拟设备应与下线前保持一致。因此在第一次建立连接之后，需要在本地网关中将对应关系保存下来，故作如下设计：

在数据库中建议一个五元组，分别为：product_key（char*）、device_name（char *）、sem_id（int，用来实现异步通信的信号量）、device_ID（char *，本地设备设备号），设备类型（char），该五元组即代表了一对连接关系。

c. 数据通信：进程之间使用管道通信，进程内的线程间通过消息队列。

d. SDK 驱动程序 A：该驱动程序需要通过云端部署到边缘网关上才能被 linkedge 拉起运行，驱动程序与设备一一对应，而不是与产品一一对应。
   其详细工作流程：
   i. 驱动程序被 gateway 拉取后，初始视实际设备未连接；首先从云端获取虚拟设备的 PK 和 DN，获取该驱动程序所独有的信号量 id（驱动程序会创建与程序 B 通信的专用管道文件，sem_id 根据该管道文件产生的键值获得），device_name 为空（txt 文件中由特殊字符代替），将它们全部放入数据库中。等待对应关系建立或设备连接，若该虚拟设备曾在此网关上连接过，则仅对 sem_id 进行更新。
   ii. 等待设备上线（被上述信号量阻塞）：
   当程序 C 从程序 B 收到信息，根据信息中的 ID，在数据库寻找匹配的 device_ID，若找到，则将信息发送给对应虚拟设备的驱动；否则，寻找 device_ID 为空的一组，将 device_ID 置为 ID，并将信息发送给对应虚拟设备的驱动。
   iii. 打开上述管道文件读端，等待写端打开，等待接入设备的 ID：
   B 程序将打开 DN 对应 FIFO 管道的写端，并将 ID 发送过来。
   iv. for(){读取 C 程序写入管道的信息，并上报云端}

   注意：虚拟设备与实际设备的对应关系在实际设备第一次上报数据时建立。

e. 负责从边缘设备接受数据、发送指令到边缘设备的边缘网络驱动程序 B：
   说明：由于本文在开发时，网关并未接入边缘端网络，因此通过一个程序来模拟网关接受到的数据（终端标准输入）和将下发的命令（终端标准输出）。

f. 作为前两者数据的中转站，以及边缘网络中设备的驱动程序 C：由于程序 A、B、C 之间的数据通信可以分为上行数据通信和下行数据通信，因此将程序 C 划分为上行通信部分和下行通信部分，分别对应程序 C 中的两个线程。
   用主线程创建上行通信管理线程和下行通信线程，再由上行通信管理线程创建数据上行线程。

   上行通信：
   i. 说明：由于在网关上只有一个 B 程序在运行，所以当网管连接多个设备时，由上行通信管理线程为每个设备创建专用的上行通信线程。
   ii. 完整工作流程：
      1. 当上行通信管理线程接收到来自边缘端设备上报的数据后首先检查该设备是否首次上报数据，若是，则为该设备创建上行通信线程，若不是，则将数据转发到对应线程；
      程序 B 会使用一个结构体链表来记录已经与云端连接的设备的信息，包括：设备在边缘端的 id，云端虚拟设备的 product key 和 device name，用来控制异步通信的信号量 id。后三者从数据库中读出，并将设备 id 存入数据库。

2. 对应的上行通信线程创建后，它会通过与上一步读取的 device name 相匹配的管道，与对应的驱动程序 A 通信，完成设备上线；
3. 设备上线后，该线程进入{读取，发送}的循环，从消息队列中获取，通过上述管道发送。

下行通信：

i. 说明：下行数据的工作前提是边缘端设备与云端虚拟设备已经建立连接。
ii. 下行通信通常是用做云端给边缘端设备下发命令，用于云端在监查到异常后对边缘端做出的一定调整。
iii. 工作流程：当下行通信线程接受到驱动程序 A 下发的命令后，根据命令中的信息，将命令转发给实际设备的驱动，由驱动完成该命令的动作。

6 TIDL API 的编程

TIDL 提供给开发者的 API 分为 4 类：

1. tidl::Configuration
2. tidl::Excutor
3. tidl::ExcutionObject
4. tidl::ExcutionObjectPipeline

其中，第三类和第四类均继承至 tidl::ExcutionObjectInternalInterface。

1. class tidl::Configuration

该类用来为神经网络设置必须的参数。可以通过直线初始化参数字段，也可以通过使用 ReadFromFile()函数来从本地文件中读取配置信息。

该类包含 3 个函数（除了构造函数）：

bool Validate() const:

使配置对象中的参数字段生效。

bool Print(std::ostring&os=std::out) const:

调试-打印配置信息。

bool ReadFromFile(const std::string & file_name):
从本地文件中读取配置信息，并使之生效。

2. class tidl::Excutor

该类用来指定神经网络的配置文件并管理神经网络中层组的执行工作和建立该类对象与实际设备（EVE 和 DSP）的对应关系。

该类主要功能在其构造函数上：

Excutor(DeviceType device_type, const DeviceIds %ids, const Configure &configure, int layers_group_id=OCL_TIDL_DEFAULT_LAYERS_GROUP_ID)

该构造建立 excutor 与实际设备的连接关系。

3. class tidl::ExcutionObject

该类对象将在硬件加速器上运行 TIDL 的神经网络。

a. void SetInputOutputBuffer (const ArgInfo &in, const ArgInfo &out)

为 ExecutionObject 对象指定输入输出的缓冲区

b. char *GetInputBufferPtr ( ) const

获得输入缓冲区的指针

c. size_t GetInputBufferSizeInBytes ( ) const

获得输入缓冲区的大小

d. char *GetOutputBufferPtr ( ) const

获得输出缓冲区的指针

e. size_t GetOutputBufferSizeInBytes ( ) const

获得输出缓冲区的大小

f. void SetFrameIndex (int idx)

设置 ExecutionObject、正在处理的帧的索引值，在追踪函数运行痕迹和调试使用。

g. int GetFrameIndex ( ) const

获得正在处理的帧的 id 号，该 id 号根据上一函数产生。

h. bool ProcessFrameStartAsync ( )

开始处理一帧，该函数是一个异步执行函数，调用会会立刻返回。此时需要下一个函数 ProcessFrameWait 来等待处理结果。

i. bool ProcessFrameWait ( )

用来等待上一函数执行完成。如果上一函数并没有被调用而直接调用该函数，该函数返回错误。

j. float GetProcessTimeInMilliSeconds ( ) const

获取 TIDL 处理一帧所花费的毫秒数。

k. const std::string &GetDeviceName ( ) const

获取运行 ExecutionObject 设备的名称。

l. void WriteLayerOutputsToFile (const std :: string &filename_prefix = "trace_dump_") const

将每一层输出缓存都保存到文件中，文件名格式为：<filename_prefix>_<ID>_HxW.bin

m. const LayerOutput *GetOutputFromLayer (uint32_t layer_index, uint32_t output_index = 0) const

返回与某一层对应的 LayerOutput 对象。该对象不会自动释放，需要由调用者将其删除。

n. const LayerOutputs *GetOutputsFromAllLayers ( ) const

返回所有层的输出缓存数据。

o. int GetLayersGroupId ( ) const

返回 ExecutionObject 正在运行的层数。

4. class tid l:: ExcutionObjectPipeline

用来管理 ExecutionObject 流水线的执行。

该类的函数与上一类基本一致，在次就不再多做说明。

程序的设计与开发：

TIDL 中提供了丰富的 demo 让开发者来熟悉 TIDL 的工作模式和使用方法。同时，这些丰富的 demo 也起到了减少开发者工作量的作用。在现有的 demo 上进行自己的开发可以让开发者专注于实现自己需要的功能，而较小 TIDL 环境初始化工作所带来的负担。本文对 TIDL 的使用就是在 TIDL 原始的 demo 上做调整、修改和添加完成的。

使用的 demo：imagenet。为了能够顺利的在该 demo 上进行自己的修改和添加，首先需要先读懂此程序。此处笔者把在阅读该程序过程中需要注意的地方给出说明：

1. 结构体 cmdline_opts_t：定义在 examples / common / video_utils.h 中，用来确定 tidl 需要的参数；

```
typedef struct cmdline_opts_t_ {
  std::string config;                    //configure 文件的路径
  uint32_t    num_dsps;                  //computer core : dsp 的数量;
uint32_t    num_eves;                  //computer core : eve 的数量;
uint32_t    num_layers_groups;         //层组数量：通常一个层组跑在一个 computer core 上，
被视为一个执行单元;
uint32_t    num_frames;                //程序运行将处理的帧数
std::string input_file;                //在单一图片作为程序输入时有效，存储图片的路径;
std::string object_classes_list_file;  //该文件声明了能够识别出的所有物体，1000 个
uint32_t    output_width;              //输出图片的长宽，一般是读取 configure 文件中的数据
bool        verbose;                   //用来启动（true）/禁用（false）程序运行期间的详细
输出
```

```
bool        is_camera_input;            //如果是摄像头输入，其值为 true
bool        is_video_input;             //如果是本地视频数据输入，其值为 true
bool        is_preprocessed_input;      //用来判断输入数据是否经过预处理：成为.y；TIDL 中每
层的输出文件后缀均为.y
uint32_t    output_prob_threshold;      //用来设定输出概率的阈值
} cmdline_opts_t;
```

2. 函数 SetVideoInputOutput ( )：定义在 examples / common / video_utils.cpp 中；该函数用来确认输入的数据源是本地视频文件还是摄像头。本文测试摄像头输入分辨率 640×480 可以处理 15～20Fps。从文件输入数据 320×320 可以稳定处理 15FPS。

   笔者在查找了 OpenCV 的官方文档去熟悉该类拥有的函数和成员，连接在本文尾贴出。不一一列出所有的函数，仅给出使用函数的说明：

   a. VideoCapture::VideoCapture ( ):

      其参数有 3 种：

      空：构造该类对象，但是不打开数据源；

      const string &filename：构造该类对象，并从本地视频获取数据；

      int device：构造该类对象，并从对应设备（摄像头）获得数据。

   b. VideoCapture::grab ( ):

      用来从对应设备中获取数据，然后调用下面的函数 6 来解码数据获得图片

   c. VideoCapture::retrieve ( ):

      传入参数为：Mat &image，int channel=0；

      解码上一函数获得的帧。如果上一函数没有获取到帧（视频文件的帧已经全部获取，或者摄像头断开连接），此函数将返回 false，并且返回空指针。

3. 函数 PreprocessImage ( ) ：定义在 tidl / tidl_api / src / imgutil.cpp 中，对输入图片进行预处理。

程序设计：该程序需要完成的主要功能是识别从视频中读取或摄像头拍摄的帧，并将识别结果输出到文本文件中，程序会将运行状态实时的上报给阿里云。如果某一帧无法识别或者识别的结果小于预设的阈值，那么该帧将被提取出来保存到本地，通过阿里云 OSS C-SDK 发送到 OSS 存储空间中。

在 demo 程序上修改后，程序的运行流程如下：

说明：每个非叶子节点的子节点为其调用的子函数，顺序从上到下依次调用。

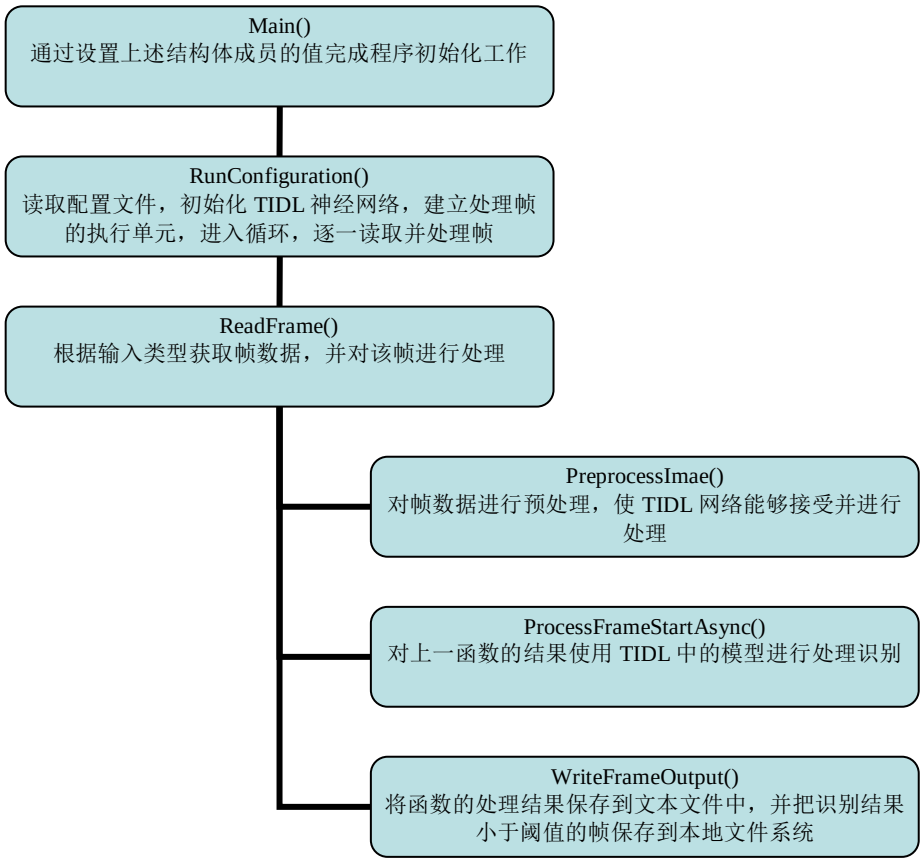

图 5：TIDL 程序调用层次

阿里云 OSS C-SDK 服务程序：一个简单的 OSS 程序，完成图片上传功能，与 TIDL 的成语一样，也是在阿里云 OSS 服务提供的上传模板上做修改添加。

7 测试

整个系统的运行步骤如下：

1. 运行阿里云 Link IoT Edge 组件，在网页控制台可以看到 网关设备在线；

   输入命令：/linkedge/gateway/build/script/iot_gateway_start.sh

2. 后台运行程序网关管理程序；

   运行命令：<TI-TEST>/4.28/log/gateway_management &

3. 后台运行 oss 程序；

   进入目录：cd /usr/share/ti/tidl/examples/hzq_imagenet/

   运行命令：./hzq_oss/main &

4. 开始运行 TIDL 程序。

运行命令：./imagenet_hzq -i ../classification/clips/test2.mp4

运行结果如下：

1. 设备驱动程序：

a. 云端查看结果：

2019/04/28 17:25:13	frame_count	帧数计数	info	{"Success_num":85}
2019/04/28 17:25:08	frame_count	帧数计数	info	{"Pass_num":100}
2019/04/28 17:25:07	frame_count	帧数计数	info	{"Success_num":93}
2019/04/28 17:24:54	frame_count	帧数计数	info	{"Pass_num":100}

图 6：云端运行日志

b. 本地日志：

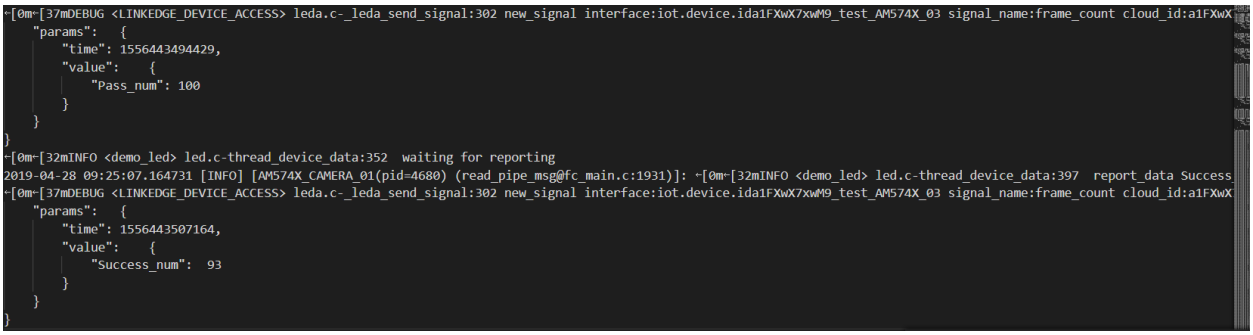

图 7：驱动程序运行日志

2. 网关管理程序运行结果：

a. 在 txt 文件中保存网络虚拟设备（test_AM574X_03）与边缘设计设备（TICMR1）的连接关系（长期有效）。

```
a1FXwX7xwM9 test_AM574X_03 0000000000000065538 TICMR1 b
a1FXwX7xwM9 test_AM574X_04 0000000000000131076 ****** b
```

图 8：存储虚拟设备与实际设备关系的文本文件

b. 接受 TIDL 发送的信息，并发送到设备驱动程序，以下为本地日志记录的信息：

```
[2019-04-28 09:24:53] [recieve_from_client] [420] : driver_client : I am waiting for data...

[2019-04-28 09:24:54] [recieve_from_client] [433] :
data for client:
ID: TICMR1
device_type: b
data_type: 1
evnt_name: frame_count
key: Pass_num
value: 100

[2019-04-28 09:24:54] [recieve_from_client] [442] : create head node!

[2019-04-28 09:24:54] [loud_pd_dn] [264] : txt exist!

[2019-04-28 09:24:54] [loud_pd_dn] [282] : fsem : 32769

[2019-04-28 09:24:54] [loud_pd_dn] [353] : sem_id : 65538

[2019-04-28 09:24:54] [recieve_from_client] [464] : head node create success!

[2019-04-28 09:24:54] [recieve_from_client] [512] : create thread!

[2019-04-28 09:24:54] [recieve_from_client] [538] : driver_client : msg_path : /linkedge/gateway/build/bin/iot-gravity/ali-sdk/msg/TICMR1
```

图 9：网关管理程序

3. TIDL 运行结果：

```
[2019-04-28 09:24:40]: baseball : prob = 95.686272
                       book_jacket : prob = 1.176471
                       menu : prob = 0.392157
                       web_site : prob = 0.392157
                       tiger_shark : prob = 0.000000

[2019-04-28 09:24:40]: baseball : prob = 99.215683
                       tiger_shark : prob = 0.000000
                       goldfish : prob = 0.000000
                       hammerhead : prob = 0.000000
                       great_white_shark : prob = 0.000000
```

图 10：TIDL 运行结果

8 结束语

本文简单的介绍了被笔者使用的阿里云服务和 TIDL 的一些基本知识以及常用 API 的使用方法，介绍了如何搭建起边缘环境。然后阐述了笔者设计编写驱动程序和 TIDL 程序的思路，展示了最终的完成结果。本文描述的实施过程中，搭建起适合阿里云 Linkedge 网关运行的边缘环境，这花费了笔者较多时间和精力，经过一系列的兼容性测试、实际验证最后获得一个稳定的软件包。

在此特别感谢阿里云技术支持为本文的完成提供的帮助与支持。

9 参考文献

物联网边缘计算官方文档：
https://help.aliyun.com/product/69083.html?spm=a2c4g.11186623.3.1.3de3211epMejGg

对象存储 OSS 官方文档：

https://help.aliyun.com/document_detail/31817.html?spm=5176.11065259.1996646101.searchclickresult.4eed47bepVaiMC

阿里云边缘计算 C-SDK：

https://github.com/aliyun/link-iot-edge-access-sdk-c?spm=a2c4g.11186623.2.26.237b2377wjVoZW

TI 处理器开发指南：

http://software-dl.ti.com/processor-sdk-linux/esd/docs/05_01_00_11/linux/index.html

TIDL API 用户指南：

http://downloads.ti.com/mctools/esd/docs/tidl-api/intro.html

重要声明和免责声明

TI 均以“原样”提供技术性及可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证其中不含任何瑕疵，且不做任何明示或暗示的担保，包括但不限于对适销性、适合某特定用途或不侵犯任何第三方知识产权的暗示担保。

所述资源可供专业开发人员应用TI 产品进行设计使用。您将对以下行为独自承担全部责任：(1) 针对您的应用选择合适的TI 产品；(2) 设计、验证并测试您的应用；(3) 确保您的应用满足相应标准以及任何其他安全、安保或其他要求。所述资源如有变更，恕不另行通知。TI 对您使用所述资源的授权仅限于开发资源所涉及TI 产品的相关应用。除此之外不得复制或展示所述资源，也不提供其它TI或任何第三方的知识产权授权许可。如因使用所述资源而产生任何索赔、赔偿、成本、损失及债务等，TI对此概不负责，并且您须赔偿由此对TI 及其代表造成的损害。

TI 所提供产品均受TI 的销售条款 (http://www.ti.com.cn/zh-cn/legal/termsofsale.html) 以及ti.com.cn上或随附TI产品提供的其他可适用条款的约束。TI提供所述资源并不扩展或以其他方式更改TI 针对TI 产品所发布的可适用的担保范围或担保免责声明。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122