TI MCU, DSP 및 Xilinx FPGA 프로그래밍 전문가 과정

Innova Lee(이상훈)
gcccompil3r@gmail.com

TIDL API User's Guide

Introduction

**TI Deep Learning(TIDL) API 는 EVE 및 C66x DSP 연산 엔진에서 TI 의 독점적이고
고도로 최적화 된 CNN/DNN 구현을 활용할 수 있도록 지원함으로써 심층적인 학습을 가능하게 한다.**

**이 사용 설명서는 TIDL API 를 다룬다.
전반적인 개발 흐름, TI 의 SoC 에 대한 CNN/DNN 의 성능 최적화,
성능/벤치마킹 데이터 및 지원되는 Layer 목록 등의 TIDL 에 대한 자세한 내용은
Processor SDK Linux Software Developer's Guide 의 TIDL 섹션을 참조하라.**
http://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/index.html

참고:

**TIDL API 는 AM57x SoC 에서만 사용할 수 있다.
OpenCL 버전 1.1.15.1 이상이 필요하다.**

Key Features: Ease of use

- OpenCV 와 같은 프레임워크에 TIDL API 를 쉽게 통합 할 수 있다.
  http://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/Foundational_Components.html#opencv

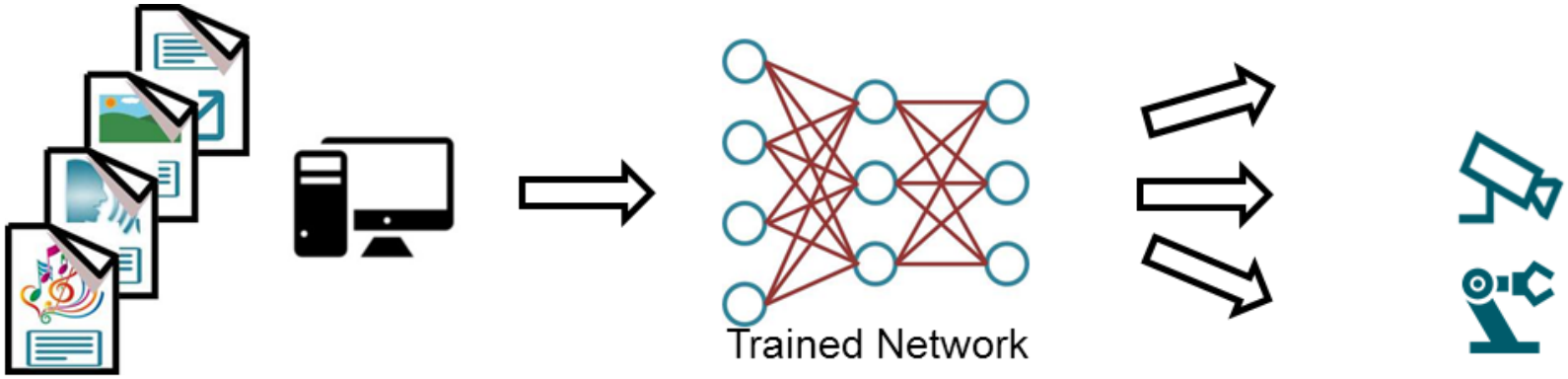
- 여러 컴퓨팅 엔진(EVE 및 C66x DSP)에서 사용자 응용 프로그램에 공통적인 Host 추상화 제공

Low overhead

Host 에서 TIDL API 실행 시간은 전체적인 Frame 당 실행 시간 중 아주 작은 비율이다.
예로 jseg21 Network 의 경우 1024 x 512 Frame 에 3 채널이 있는 경우
API 는 전체 Frame 당 처리 시간의 1.5% 를 차지한다.

Software Architecture

TIDL API 는 TI 의 OpenCL 제품을 활용하여 Deep Learning 애플리케이션을 EVE(s) 및 DSP(s) 모두에 Offload 한다.
http://downloads.ti.com/mctools/esd/docs/opencl/index.html
TIDL API 는 사용자를 위한 기본 학습 경험을 크게 향상시키고 전반적인 사용 사례에 집중할 수 있도록 한다.
ARM <-> DSP/EVE 통신의 메커니즘이나 EVE(s) 및 / 또는
DSP(s)에서 최적화 된 Network Layer 구현에 시간을 할애 할 필요가 없다.
이 API 를 통해 고객은 OpenCV 와 같은 Framework 를 손쉽게 통합하고
Deep Learning 애플리케이션을 신속하게 프로토타입 할 수 있다.



Deep Learning/CNN training

Caffe*

TensorFlow*

Caffe-Jacinto

*Only certain models supported. There are constraints on supported layers that must be met. This is not a universal translator.

TI network translator tool

TI network translator (format conversion)

TI-DL

TIDL API

OpenCL Runtime

Sitara™ AM57x

Texas Instruments

Training (PC/GPU)

Format Conversion (PC / Linux)

Inference – deep learning application (embedded device)

위의 그림은 전반적인 개발 절차를 보여준다.
Deep Learning 개발은 training 과 배포 단계에서의 추론 두 단계로 구성된다.

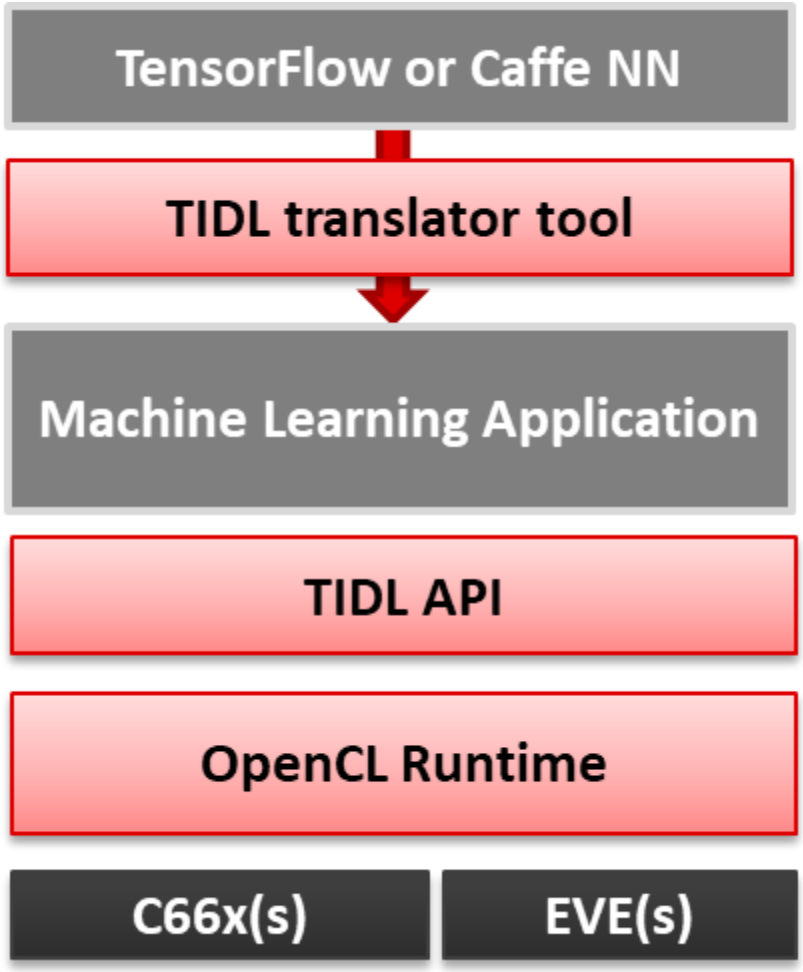
training 에는 Neural Network Model 을 설계하고 Network 를 통해
Training Data 를 실행하여 Model Parameter 를 조정한다.

추론은 Parameter 를 포함하여 사전 훈련된 Model 을 취하여 새로운 입력에 적용하고 출력을 생성한다.
Training 은 계산 집약적이며 Caffe/Tensorflow 와 같은 프레임워크를 사용하여 수행된다.
Network 가 훈련되면 TIDL Converter Tool 을 사용하여 Network 및 Parameter 를 TIDL 로 변환 할 수 있다.

Processor SDK Linux Software Developer's Guide 는 개발 흐름 및 Converter Tool 에 대한 자세한 내용을 제공한다.
http://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/index.html

Converter Tool 은 TIDL Network Binary 파일과 Model 또는 Parameter 파일을 생성한다.
Network 파일은 Network Graph 를 지정한다.
Parameter 파일은 가중치를 지정한다.

그림 2 는 TIDL API SW 아키텍처를 보여준다.


TensorFlow or Caffe NN
TIDL translator tool
Machine Learning Application
TIDL API
OpenCL Runtime
C66x(s)
EVE(s)

TIDL API 는 직관적인 세 가지 C++ 클래스를 제공한다.
구성은 Network 및 parameter 이진 파일에 대한 포인터를 포함하여 Network 구성을 캡슐화한다.
Executor 는 On-Device Memory 할당, Network 설정 및 초기화를 캡슐화 한다.
이러한 클래스의 구현은 OpenCL Run-Time 을 호출하여
Network 처리를 EVE/DSP 장치로 Offload 하여 사용자로부터 이러한 세부 정보를 추출한다.

Listing 1 은 TIDL API 를 사용하여 사용자 애플리케이션에서
Deep Learning 애플리케이션을 얼마나 쉽게 활용하는지를 보여준다.
이 예제에서 TIDL Network 구성 파일을 읽음으로써 구성 객체가 생성된다.
실행 프로그램 개체는 두 개의 EVE Devices 로 생성된다.
구성 객체를 사용하여 EVE 에서 TIDL Network 를 설정하고 초기화 한다.
두 실행 객체 각각은 TIDL 처리를 다른 EVE Core 로 전달한다.
OpenCL kernel 실행은 비동기이므로 두 개의 EVE 에서 Frame 을 Pipeline 할 수 있다.
한 Frame 이 EVE 에 의해 처리 될 때 다음 Frame 은 다른 EVE 에 의해 처리 될 수 있다.

Listing 1. TIDL API 를 사용하는 애플리케이션

```
// TI DL Network 구성 파일 읽기
Configuration configuration;
bool status = configuration.ReadFromFile("./tidl_j11v2_net");

// 2 개의 EVE 및 구성으로 실행 프로그램을 작성하라.
DeviceIds ids = {DeviceId::ID0, DeviceId::ID1};
Executor executor(DeviceType::EVE, ids, configuration);

// 생성 된 ExecutionObject 집합에 대한 Query 실행기
const ExecutionObjects& eos = executor.GetExecutionObjects();
int num_eos = eos.size();  // 2 EVEs

// 각 실행 개체에 대한 입력 및 출력 버퍼를 할당한다.
for (auto &eo : eos)
{
    ArgInfo in(eo->GetInputBufferSizeInBytes());
    ArgInfo out(eo->GetOutputBufferSizeInBytes());
    eo->SetInputOutputBuffer(in, out);
}

// 2 개의 EVE Core 를 사용한 파이프라인 처리
for (int idx = 0; idx < configuration.numFrames + num_eos; idx++)
{
    ExecutionObject* eo = eos[idx % num_eos].get();

    // 동일한 EO 에서 이전 Frame 이 처리를 마칠 때까지 기다린다.
    if (eo->ProcessFrameWait())  WriteFrameOutput(*eo);

    // Frame 을 읽고 현재 eo 로 처리를 시작한다.
    if (ReadFrameInput(*eo, idx))  eo->ProcessFrameStartAsync();
}
```

ReadFrameInput 및 WriteFrameOutput 함수는 입력 프레임을 읽고 처리 결과를 쓰는데 사용된다.
예를 들어 OpenCV 를 사용하면 OpenCV API 를 사용하여 Frame 을 캡쳐하는 ReadFrameInput 이 구현된다.
DSP 에서 동일한 Network 를 실행하려면 Listing 1 의 유일한 변경 사항은
DeviceType::EVE 를 DeviceType::DSP 로 바꾸는 것이다.

섹션 TIDL API 사용에는 API 사용에 대한 세부 정보가 들어 있다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/using_api.html#using-tidl-api
API 자체는 TIDL API Reference 섹션에 설명되어 있다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#api-documentation

일부 유형의 Layer 는 EVE 에서 더 빠르게 실행될 수 있고
다른 유형의 DSP 에서 더 빠르게 실행될 수 있기 때문에 Network 를 분할하고
다른 Core 에서 다른 부분을 실행하는 것이 유용할 때도 있다.
TIDL API 는 EVE 및 DSP 에서 분할 Network 를 실행할 수 있는 유연성을 제공한다.
자세한 내용은 SSD 예제를 참조하라.

Using the TIDL API

이 예는 TIDL API 를 사용하여 Linux 애플리케이션에서 AM57x 디바이스의 C66x DSP
또는 EVE 로의 Deep Learning Network 처리를 Offload 하는 방법을 보여준다.
API 는 세 가지 클래스로 구성된다:
Configuration, Executor 및 ExecutionObject 에 해당한다.

Step 1

AM57x SoC 에 TIDL 가능 장치가 있는지 확인하라:

```
uint32_t num_eve = Executor::GetNumDevices(DeviceType::EVE);
uint32_t num_dsp = Executor::GetNumDevices(DeviceType::DSP);
```

참고:

기본적으로 OpenCL Run-Time 은 충분한 Global 메모리(CMEM 을 통해)로 구성되어
TIDL Network 를 2 개의 OpenCL Device 로 Offload 한다.

Executor::GetNumDevices 가 4 를 반환하는 장치
(예: 4 개의 EVE OpenCL 장치가 있는 AM5729) 에서는 Run-Time 에 사용할 수 있는 메모리 양을 늘려야 한다.
자세한 내용은 Insufficient OpenCL global memory 를 참조하라.
http://downloads.ti.com/mctools/esd/docs/tidl-api/faq/out_of_memory.html#opencl-global-memory

Step 2

Configuration 객체를 파일로부터 읽어들이거나 직접 초기화 해 Configuration 객체를 생성한다.
아래 예에서는 Configuration 파일을 파싱하고 Configuration 객체를 초기화한다.
Configuration 파일의 예는 examples/test/testvecs/config/infer 를 참조하라.

```
Configuration configuration;
bool status = configuration.ReadFromFile(config_file);
```

참고:

TensorFlow 및 Caffe 에서 TIDL Network 및 Parameter 바이너리를 작성하려면
Processor SDK Linux Software Developer's Guide 를 참조하라.
http://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/index.html

Step 3

적절한 장치 유형, 장치 집합 및 구성으로 실행 프로그램을 작성하라.
아래 Snippet 에서 Execve 는 2 개의 EVE 에 생성된다.

```
DeviceIds ids = {DeviceId::ID0, DeviceId::ID1};
Executor executor(DeviceType::EVE, ids, configuration);
```

Step 4

사용 가능한 ExecutionObject 집합을 가져와서 각 ExecutionObject 에 대한 입력 및 출력 버퍼를 할당한다.

```
const ExecutionObjects& execution_objects = executor.GetExecutionObjects();
int num_eos = execution_objects.size();

// Allocate input and output buffers for each execution object
std::vector<void *> buffers;
for (auto &eo : execution_objects)
{
    ArgInfo in  = { ArgInfo(malloc(frame_sz), frame_sz)};
    ArgInfo out = { ArgInfo(malloc(frame_sz), frame_sz)};
    eo->SetInputOutputBuffer(in, out);

    buffers.push_back(in.ptr());
    buffers.push_back(out.ptr());
}
```

Step 5

각 입력 프레임에서 Network 를 실행한다.
Frame 은 파이프라인을 flush 하는 추가 num_eos 반복을 사용하여
파이프라인 방식으로 사용 가능한 실행 객체로 처리된다(epilogue).

```
for (int frame_idx = 0; frame_idx < configuration.numFrames + num_eos; frame_idx++)
{
    ExecutionObject* eo = execution_objects[frame_idx % num_eos].get();

    // Wait for previous frame on the same eo to finish processing
    if (eo->ProcessFrameWait())
        WriteFrame(*eo, output_data_file);

    // Read a frame and start processing it with current eo
    if (ReadFrame(*eo, frame_idx, configuration, input_data_file))
        eo->ProcessFrameStartAsync();
}
```

API 사용에 대한 전체 예제는 EVM 파일 시스템의 /usr/share/ti/tidl/examples 에 있는 예를 참조하라.

Network Viewer

TIDL Network Viewer 유틸리티인 tidl_viewer 를 사용하여 Network Graph 를 볼 수 있다.
dot 유틸리티를 사용할 수 있는 경우 tidl_viewer 는 이 파일을 사용하여 dot 파일을 svg 로 변환한다.
dot 은 Graphviz Ubuntu 패키지에 포함되어 있으며 일반적으로 /usr/bin/dot 에 설치된다.

예를 들어 아래 명령은 ssd.dot 의 tidl_net_jdetNet_ssd.bin 네트워크 바이너리에 대한 dot 그래프를 설명한다.

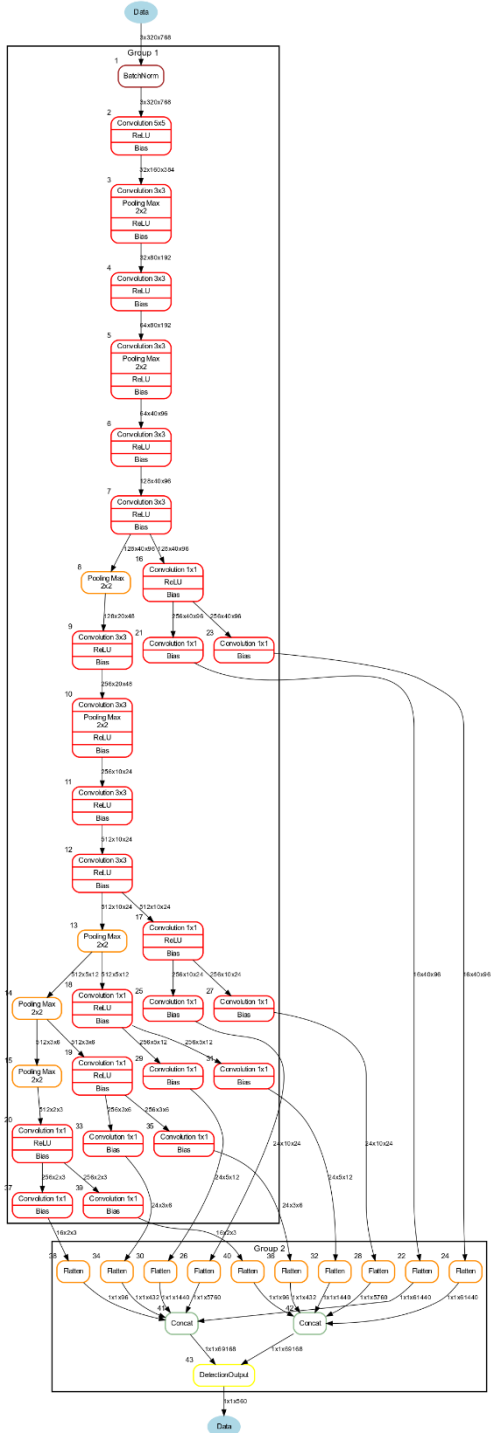
```
$ tidl_viewer examples/test/testvecs/config/tidl_models/tidl_net_jdetNet_ssd.bin -d ssd.dot
```

x86/Linux 에서 /usr/bin/dot 을 사용할 수 있는 경우 tidl_viewer 는 svg 파일인 ssd.dot.svg 도 생성한다.

참고:

dot 이 /usr/bin/dot 에 설치되어 있지 않으면 tidl_viewer 는 점을 svg 로 변환하지 않다.
다음 명령을 사용하여 svg 파일을 별도의 단계로 생성할 수 있다:
dot -Tsvg ssd.dot -o ssd.dot.svg
Graphviz 및 dot 에 대한 추가 정보는 아래와 같다:

Graphviz
Ubuntu Graphviz package


Data
3x320x768
Group 1
1
BatchNorm
3x320x768
2
Convolution 5x5
ReLU
Bias
32x160x384
3
Convolution 3x3
Pooling Max 2x2
ReLU
Bias
32x80x192
4
Convolution 3x3
ReLU
Bias
64x80x192
5
Convolution 3x3
Pooling Max 2x2
ReLU
Bias
64x40x96
6
Convolution 3x3
ReLU
Bias
125x40x96
7
Convolution 3x3
ReLU
Bias
128x40x96
125x40x96
8
Pooling Max 2x2
16
Convolution 1x1
ReLU
Bias
128x20x48
256x40x96
9
Convolution 3x3
ReLU
Bias
21
Convolution 1x1
Bias
23
Convolution 1x1
Bias
256x20x48
10
Convolution 3x3
Pooling Max 2x2
ReLU
Bias
256x10x24
11
Convolution 3x3
ReLU
Bias
512x10x24
12
Convolution 3x3
ReLU
Bias
512x10x24
512x10x24
13
Pooling Max 2x2
17
Convolution 1x1
ReLU
Bias
256x10x24
256x10x24
256x40x96
256x40x96
512x5x12
512x5x12
14
Pooling Max 2x2
18
Convolution 1x1
ReLU
Bias
25
Convolution 1x1
Bias
27
Convolution 1x1
Bias
512x2x6
512x3x6
256x5x12
256x3x12
15
Pooling Max 2x2
19
Convolution 1x1
ReLU
Bias
29
Convolution 1x1
Bias
31
Convolution 1x1
Bias
512x2x3
256x3x6
256x3x6
20
Convolution 1x1
ReLU
Bias
33
Convolution 1x1
Bias
35
Convolution 1x1
Bias
24x10x24
24x10x24
256x2x3
256x2x3
24x5x12
24x5x12
37
Convolution 1x1
Bias
39
Convolution 1x1
Bias
24x2x6
24x3x8
16x2x3
16x2x3
Group 2
38
Flatten
34
Flatten
30
Flatten
26
Flatten
40
Flatten
36
Flatten
32
Flatten
28
Flatten
22
Flatten
24
Flatten
1x1x96
1x1x432
1x1x1440
1x1x5760
1x1x96
1x1x432
1x1x1440
1x1x5760
1x1x61440
1x1x61440
41
Concat
42
Concat
1x1x69168
1x1x69168
43
DetectionOutput
1x1x560
Data

Examples

우리는 tidl-api 패키지 내에 세 가지 포괄적인 예를 제공하여 세 가지 범주의 Deep Learning 네트워크를 시연한다.
첫 번째 두 예제는 EVE 또는 DSP Device 로 AM57x SoC 에서 실행 될 수 있다.
마지막 예에서는 EVE 와 DSP 를 모두 갖춘 AM57x SoC 가 필요하다.
여기에 제시된 성능 수치는 1.5 GHz 에서 실행되는 ARM A15 Core 2 개,
535 MHz 에서 4 개의 EVE Core 및 750 MHz 에서
2 개의 DSP Core 를 포함하는 AM5729 EVM 에서 얻은 성능 수치다.

각 예에서 Device 처리 시간, Host 처리 시간 및 TIDL API Overhead 를 보고한다.
Device 처리 시간은 처리가 완료 될 때까지 Frame 에 대한 처리가 시작되는 순간부터 장치에서 측정된다.
Host 처리 시간은 사용자 으용 프로그램에서 ProcessFrameWait() 가 반환 될 때까지
ProcessFrameStartAsync() 가 호출되는 순간부터 Host 에서 측정된다.
여기에는 TIDL API Overhead, OpenCL Run-Time Overhead 및
사용자 입력 데이터를 패딩된 TIDL 내부 버퍼로 복사하는 시간이 포함된다.

Imagenet

imagenet 예제는 Image 를 입력으로 사용하여 1000 개의 확률을 출력한다.
각 확률은 Network 가 사전 훈련한 1000 개의 객체 중 하나의 객체에 해당한다.
이 예에서 입력 이미지가 될 가능성이 가장 높은 개체로 상위 5 개의 예측을 출력한다.

다음 그림과 표는 입력 이미지, 출력으로 예상되는 상위 5 개 개체 및 EVE 또는 DSP 에서의 처리 시간을 보여준다.

Rank	Object Classes
1	tabby
2	Egyptian_cat
3	tiger_cat
4	lynx
5	Persian_cat

Device Processing Time	Host Processing Time	API Overhead
EVE: 123.1 ms	124.7 ms	1.34 %
OR		
DSP: 117.9 ms	119.3 ms	1.14 %

이 범주에서 실행한 특정 Network 인 jacintonet11v2 에는 14 개의 Layer 가 있다.
사용자는 가속을 위해 EVE 또는 DSP 에서 Network 를 실행할 것인지 여부를 지정할 수 있다.
EVE 시간이 DSP 시간보다 약간 높다는 것을 알 수 있다.
전체 Overhead 가 1.5% 미만임을 알 수 있다.

참고:

여기에 report 된 예측은 실제 확률에 대해 정규화되지 않은 Network 의 softmax Layer 의 출력을 기반으로한다.

Segmentation

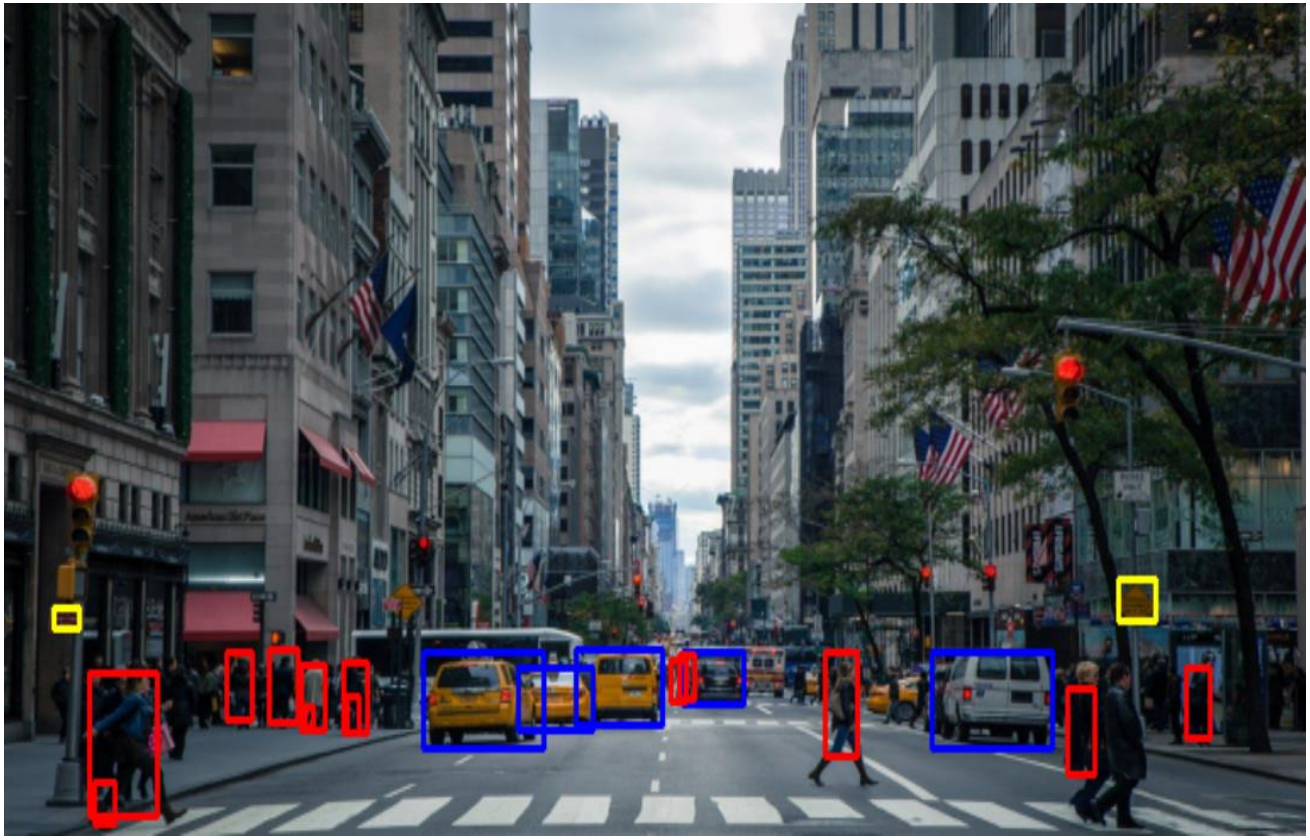
세분화 예는 Image 를 입력으로 상요하고 사전 훈련된 Category 에 따라 Pixel 수준 분류를 수행한다.
아래 그림은 거리 장면을 입력하고 표시하고 픽셀 수준 분류로 출력 장면을 중첩한것을 보여준다.
녹색의 길거리, 빨간색의 보행자, 파란색의 차량 및 회색의 배경을 볼 수 있다.

이 카테고리에서 실행한 Network 는 26 개의 Layer 가 있는 jsegnet21v2 다.
아래 표에서 보고 된 시간을 보면 이 Network 가 EVE 에서 DSP 보다 훨씬 빠르다.

Device Processing Time	Host Processing Time	API Overhead
EVE: 296.5 ms	303.3 ms	2.26 %
OR		
DSP: 812.0 ms	818.4 ms	0.79 %

SSD

SSD 는 Single Shot Multi-Box Detector 의 약자다.
ssd_multibox 예제는 Image 를 입력으로 사용하고 사전 훈련된 카테고리에 따라 경계 상자가 있는 여러 객체를 감지한다.
아래 그림은 다른 거리 장면을 입력으로 인식하고 인식된 객체가 출력에서 상자에 걸린 모습을 보여준다.
빨간색이 보행자, 파란색은 차량, 노란색은 교통 신호에 해당한다.



Network 는 완전히 EVE 또는 DSP 에서 실행될 수 있다.
그러나 이 특별한 jdetnet_ssd Network 에서 EVE 의 처음 30 개 Layer 와
DSP 의 다음 13 개 Layer 를 실행하면 최상의 성능을 얻을 수 있다.
보고 된 시간에 대해서는 아래 표의 AND 에 유의하라.
이 end-to-end 예는 Layer Group id 를 Executor 에 할당하는 것이 얼마나 쉬운지를 보여주며
하나의 ExecutionObject 에서 다른 ExecutionObject 로의 출력을 연결하는 것이 얼마나 쉬운지를 보여준다.

Device Processing Time	Host Processing Time	API Overhead
EVE: 175.2 ms	179.1 ms	2.14 %
AND		
DSP: 21.1 ms	22.3 ms	5.62 %

Test

이 예는 TIDL API 패키지(test/testvecs/config/tidl_models)에 포함된 사전 변환된 Network 를 test 하는데 사용된다.
인수 없이 실행하면 test_tidl 프로그램은 SoC 에서 사용 가능한
C66x DSP 및 EVE 에서 사용 가능한 모든 Network 를 실행한다.
단일 Network 를 지정하려면 -c 옵션을 사용하라.
자세한 내용은 test_tidl -h 를 실행하라.

Running Examples

예제는 EVM 파일 시스템의 /usr/share/ti/tidl/exammples 에 있다.
각 예제는 자체 디렉토리를 실행해야 한다.
-h 를 사용하여 예제를 실행하면 옵션 집합이 있는 도움말 메시지가 표시된다.
아래 코드 섹션은 예제를 실행하는 방법과 지원되는 모든 TIDL Network 구성을 테스트하는 프로그램을 보여준다.

```
root@am57xx-evm:~# cd /usr/share/ti/tidl-api/examples/imagenet/
root@am57xx-evm:/usr/share/ti/tidl-api/examples/imagenet# make -j4
root@am57xx-evm:/usr/share/ti/tidl-api/examples/imagenet# ./imagenet -t d
Input: ../test/testvecs/input/objects/cat-pet-animal-domestic-104827.jpeg
frame[0]: Time on device:  117.9ms, host:  119.3ms API overhead:   1.17 %
1: tabby, prob = 0.996
2: Egyptian_cat, prob = 0.977
3: tiger_cat, prob = 0.973
4: lynx, prob = 0.941
5: Persian_cat, prob = 0.922
imagenet PASSED
```

```
root@am57xx-evm:/usr/share/ti/tidl-api/examples/imagenet# cd ../segmentation/; make -j4
root@am57xx-evm:/usr/share/ti/tidl-api/examples/segmentation# ./segmentation -i ../test/testvecs/input/roads/pexels-photo-972355.jpeg
Input: ../test/testvecs/input/roads/pexels-photo-972355.jpeg
frame[0]: Time on device:  296.5ms, host:  303.2ms API overhead:   2.21 %
Saving frame 0 overlayed with segmentation to: overlay_0.png
segmentation PASSED

root@am57xx-evm:/usr/share/ti/tidl-api/examples/segmentation# cd ../ssd_multibox/; make -j4
root@am57xx-evm:/usr/share/ti/tidl-api/examples/ssd_multibox# ./ssd_multibox -i ../test/testvecs/input/roads/pexels-photo-378570.jpeg
Input: ../test/testvecs/input/roads/pexels-photo-378570.jpeg
frame[0]: Time on EVE:  175.2ms, host:    179ms API overhead:    2.1 %
frame[0]: Time on DSP:  21.06ms, host:  22.43ms API overhead:   6.08 %
Saving frame 0 with SSD multiboxes to: multibox_0.png
Loop total time (including read/write/print/etc):  423.8ms
ssd_multibox PASSED
```

```
root@am57xx-evm:/usr/share/ti/tidl-api/examples/ssd_multibox# cd ../test; make -j4
root@am57xx-evm:/usr/share/ti/tidl-api/examples/test# ./test_tidl
API Version: 01.00.00.d91e442
Running dense_1x1 on 2 devices, type EVE
frame[0]: Time on device:  134.3ms, host:  135.6ms API overhead:  0.994 %
dense_1x1 : PASSED
Running j11_bn on 2 devices, type EVE
frame[0]: Time on device:  176.2ms, host:  177.7ms API overhead:  0.835 %
j11_bn : PASSED
Running j11_cifar on 2 devices, type EVE
frame[0]: Time on device:  53.86ms, host:  54.88ms API overhead:   1.85 %
j11_cifar : PASSED
Running j11_controlLayers on 2 devices, type EVE
frame[0]: Time on device:  122.9ms, host:  123.9ms API overhead:  0.821 %
j11_controlLayers : PASSED
Running j11_prelu on 2 devices, type EVE
frame[0]: Time on device:  300.8ms, host:  302.1ms API overhead:  0.437 %
j11_prelu : PASSED
Running j11_v2 on 2 devices, type EVE
frame[0]: Time on device:  124.1ms, host:  125.6ms API overhead:   1.18 %
j11_v2 : PASSED
Running jseg21 on 2 devices, type EVE
frame[0]: Time on device:    367ms, host:    374ms API overhead:   1.88 %
jseg21 : PASSED
```

```
Running jseg21_tiscapes on 2 devices, type EVE
frame[0]: Time on device:  302.2ms, host:  308.5ms API overhead:   2.02 %
frame[1]: Time on device:  301.9ms, host:  312.5ms API overhead:   3.38 %
frame[2]: Time on device:  302.7ms, host:  305.9ms API overhead:   1.04 %
frame[3]: Time on device:  301.9ms, host:    305ms API overhead:   1.01 %
frame[4]: Time on device:  302.7ms, host:  305.9ms API overhead:   1.05 %
frame[5]: Time on device:  301.9ms, host:  305.5ms API overhead:   1.17 %
frame[6]: Time on device:  302.7ms, host:  305.9ms API overhead:   1.06 %
frame[7]: Time on device:  301.9ms, host:    305ms API overhead:   1.02 %
frame[8]: Time on device:    297ms, host:  300.3ms API overhead:   1.09 %
Comparing frame: 0
jseg21_tiscapes : PASSED
Running smallRoi on 2 devices, type EVE
frame[0]: Time on device:  2.548ms, host:  3.637ms API overhead:   29.9 %
smallRoi : PASSED
Running squeeze1_1 on 2 devices, type EVE
frame[0]: Time on device:  292.9ms, host:  294.6ms API overhead:  0.552 %
squeeze1_1 : PASSED
```

```
Multiple Executor...
Running network tidl_config_j11_v2.txt on EVEs: 1  in thread 0
Running network tidl_config_j11_cifar.txt on EVEs: 0  in thread 1
Multiple executors: PASSED
Running j11_bn on 2 devices, type DSP
frame[0]: Time on device:  170.5ms, host:  171.5ms API overhead:  0.568 %
j11_bn : PASSED
Running j11_controlLayers on 2 devices, type DSP
frame[0]: Time on device:  416.4ms, host:  417.1ms API overhead:  0.176 %
j11_controlLayers : PASSED
Running j11_v2 on 2 devices, type DSP
frame[0]: Time on device:    118ms, host:  119.2ms API overhead:   1.01 %
j11_v2 : PASSED
Running jseg21 on 2 devices, type DSP
frame[0]: Time on device:   1123ms, host:   1128ms API overhead:  0.443 %
jseg21 : PASSED
Running jseg21_tiscapes on 2 devices, type DSP
frame[0]: Time on device:  812.3ms, host:  817.3ms API overhead:  0.614 %
frame[1]: Time on device:  812.6ms, host:  818.6ms API overhead:  0.738 %
frame[2]: Time on device:  812.3ms, host:  815.1ms API overhead:  0.343 %
frame[3]: Time on device:  812.7ms, host:  815.2ms API overhead:  0.312 %
frame[4]: Time on device:  812.3ms, host:  815.1ms API overhead:  0.353 %
frame[5]: Time on device:  812.6ms, host:  815.1ms API overhead:  0.302 %
frame[6]: Time on device:  812.2ms, host:  815.1ms API overhead:  0.357 %
frame[7]: Time on device:  812.6ms, host:  815.2ms API overhead:  0.315 %
frame[8]: Time on device:    812ms, host:    815ms API overhead:  0.367 %
Comparing frame: 0
jseg21_tiscapes : PASSED
Running smallRoi on 2 devices, type DSP
frame[0]: Time on device:  14.21ms, host:  14.94ms API overhead:   4.89 %
smallRoi : PASSED
Running squeeze1_1 on 2 devices, type DSP
frame[0]: Time on device:    960ms, host:  961.1ms API overhead:  0.116 %
squeeze1_1 : PASSED
tidl PASSED
```

TIDL API Reference: Configuration

Configuration 개체는 Network 실행에 필요한 다양한 매개 변수를 지정하는데 사용된다.
응용 프로그램은 Configuration 인스턴스에서 필드를 직접 초기화하거나
ReadFromFile 메서드를 사용하여 파일에서 구성을 읽을 수 있다.
다음 절에서는 Configuration 클래스에서 일반적으로 사용되는 필드에 대해 설명한다.

Input image description

```
std::string Configuration.inData
```

입력 이미지 파일의 경로다.
이 필드는 TIDL API 자체에서 사용되지 않는다.
응용 프로그램에서 입력 이미지를 버퍼에 Load 하는데 사용할 수 있다.
응용 프로그램이 OpenCV 와 같은 프레임워크를 사용하여 Image 를 읽는 경우 비어 있을 수 있다.
예제 사용법은 test/main.cpp 를 참조하라.

```
std::size_t Configuration.inHeight
```

입력 이미지의 높이이며 API 에서 사용하며 지정해야 한다.

```
std::size_t Configuration.inWidth
```

입력 이미지의 너비이며 API 에서 사용하며 지정해야한다.

```
std::size_t Configuration.inNumChannels
```

입력 이미지의 채널 수이며 API 에서 사용하며 지정해야 한다.

Output description

```
std::string Configuration.outData
```

출력 이미지 파일의 경로다.
이 필드는 TIDL API 자체에서 사용되지 않는다.
응용 프로그램에서 버퍼를 파일에 쓰는데 사용할 수 있다.
응용 프로그램이 OpenCV 와 같은 프레임워크를 사용하여 이미지를 읽는 경우 비어있을 수 있다.
예제 사용법은 test/main.cpp 를 참조하라.

Network

```
std::string Configuration.netBinFile
```

TIDL 네트워크 바이너리 파일의 경로이며 API 에서 사용하고 지정해야 한다.

```
std::string Configuration.paramBinFile
```

TIDL parameter 파일의 경로이며 API 에서 사용하고 지정해야 한다.

Memory Management

Configuration 객체는 2 개의 힙 크기를 지정한다.
이러한 힙은 Host 와 Device 에서 공유되는 OpenCL Global 메모리에서 할당된다.
OpenCL Global 메모리 힙의 크기를 늘리는 단계는 Insufficient OpenCL global memory 섹션을 참조하라.
http://downloads.ti.com/mctools/esd/docs/tidl-api/faq/out_of_memory.html#opencl-global-memory

```
std::size_t Configuration.PARAM_HEAP_SIZE
```

이 필드는 Network parameter 에 사용되는 Device Heap 의 크기를 지정하는데 사용된다.
크기는 parameter 바이너리 파일의 크기에 따라 다르다.
예를 들어 jsegnet21v2 의 parameter 파일인 tidl_param_jsegnet21v2.bin 은 2.6 MB 이다.
정렬 이유로 인해 parameter 힙은 바이너리 파일 크기(이 경우 2.9 MB) 보다 10% 커야 한다.
Configuration 의 생성자는 PARAM_HEAP_SIZE 를 9 MB 로 설정한다.
Executor 의 각 인스턴스에 대해 하나의 parameter 힙이 있다.

```
std::size_t Configuration.EXTMEM_HEAP_SIZE
```

이 필드는 Network parameter 이외의 모든 할당에 사용되는 Device Heap 의 크기를 지정하는데 사용된다.
Configuration 의 생성자는 EXTMEM_HEAP_SIZE 를 64 MB 로 설정한다.
ExecutionObject 의 각 인스턴스에 대해 하나의 외부 메모리 힙이 있다.

API Reference

namespace **tidl**

Typedefs

typedef std::set<DeviceId> **DeviceIds**

Executor 에서 사용할 수 있는 Device 집합을 지정하는데 사용된다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1Executor

typedef std::vector<std::unique_ptr<ExecutionObject>> **ExecutionObjects**

http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl15ExecutionObjectE
Executor::GetExecutionObjects 의 반환 형식을 정의한다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1Executor_1afc042378540b27bddb231284b5c49cf4

using **tidl::up_malloc_ddr = typedef std::unique_ptr<T, decltype(&__free_ddr)>**

__free_ddr deleter 이 있는 unique_ptr 의 템플릿 typedef 에 해당한다.

Enums

enum **DeviceType**

Network 에서 Offload 할 수 있는 Device 의 type 을 열거한다.

Values:

DSP

C66x DSP 로 Offload

EVE

TI EVE 로 Offload

enum **DeviceId**

지정된 유형의 Device 에 대한 ID 를 열거한다.

Values:

ID0 =0	**DSP1 혹은 EVE1**
ID1	**DSP2 혹은 EVE2**
ID2	**EVE3**
ID3	**EVE4**

Functions

```
void __free_ddr(void *ptr)
void *__malloc_ddr(size_t s)
template <class T> T *malloc_ddr()
```

__malloc_ddr wrapper - sizeof(T) 에 의해 결정된 할당 바이트

```
template <class T> T *malloc_ddr(size_t size)
```

__malloc_ddr wrapper - 인수로 전달 된 할당 된 바이트

```
class Configuration
```

#include <configuration.h>
Network 에 필요한 구성을 지정한다.

Public Functions

```
Configuration()
```

기본 생성자

```
bool Validate() const
```

configuration 객체의 필드를 확인한다.

```
void Print(std::ostream &os = std::cout) const
```

configuration 을 인쇄하라.

```
bool ReadFromFile(const std::string &file_name)
```

지정된 파일에서 configuration 을 읽고 유효성을 검사하라.

Public Members

`int numFrames`

입력 데이터의 Frame 수(데이터가 파일에서 읽히지 않으면 0 이 될 수 있음)

`int inHeight`

입력 프레임의 높이

`int inWidth`

입력 프레임의 너비

`int inNumChannels`

입력 프레임의 채널 수(예: BGR 의 경우 3)

`int preProcType`

입력 프레임에 적용되는 전처리 유형, 각 Network 에 고유하며 0 ~ 4 까지의 값을 취할 수 있으며 기본값은 0 이다.

`int runFullNet`

layersGroupId 파티셔닝에 관계없이 모든 Layer 를 강제로 실행한다.

`int enableInternalInput`

설정시 사용자 응용 프로그램이 아닌 이전 layersGroupId 의 출력을 포함하는 TIDL 내부 버퍼에서 입력을 가져온다.

`size_t EXTMEM_HEAP_SIZE`

Execution Object Heap 당 TI DL 의 크기에 해당한다.

`size_t PARAM_HEAP_SIZE`

parameter 데이터에 사용 된 Heap 의 크기

`std::string inData`

입력 파일의 위치 입력 데이터가 OpenCV 와 같은 프레임워크에서 제공되는 경우 비어 있을 수 있다.

`std::string outData`

출력 데이터가 OpenCV 와 같은 프레임워크에 의해 소비되는 경우 출력 파일의 위치가 비어 있을 수 있음

`std::string netBinFile`

TIDL 네트워크 바이너리 파일의 경로

`std::string paramsBinFile`

TIDL parameter 바이너리 파일의 경로

class `ExecutionObject`

#include <execution_object.h>
OpenCL Device 에서 TIDL Network 를 실행한다.

Public Functions

`void SetInputOutputBuffer(const ArgInfo &in, const ArgInfo &out)`

http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl7ArgInfoE
EO 에서 사용하는 입력 및 출력 버퍼 지정

Parameters

- in: 입력 용 버퍼
- Out: 출력 용 버퍼

char *GetInputBufferPtr() const

SetInputOutputBuffer 를 개입시켜 설정된 입력 버퍼의 포인터를 돌려준다.

size_t GetInputBufferSizeInBytes() const

입력 버퍼의 크기를 돌려준다.

void SetFrameIndex(int idx)

ExecutionObject 에 의해 현재 처리되는 Frame 의 Frame Index 를 설정한다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1ExecutionObject
trace/debug 메시지에 사용된다.

Parameters

- idx: 프레임의 인덱스

int GetFrameIndex() const

처리중인 프레임의 인덱스를 반환한다(SetFrameIndex 로 설정)

char *GetOutputBufferPtr() const

출력 버퍼에 대한 포인터를 반환한다.

size_t GetOutputBufferSizeInBytes() const

출력 버퍼에 기록 된 바이트 수를 반환한다.

bool ProcessFrameStartAsync()

프레임 처리를 시작하며 호출은 비동기적이고 즉시 반환된다.
ExecutionObject::ProcessFrameWait 를 사용하여 대기하라.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1ExecutionObject_1a3e4d4d15e628a5c4c218f7b5c353b0f2

bool **ProcessFrameWait**()

Execution Object 가 프레임 처리를 완료 할 때까지 기다린다.

Return

ExecutionObject::ProcessFrameStartAsync 에 대한 해당 호출없이
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1ExecutionObject_1a3e4d4d15e628a5c4c218f7b5c353b0f2
ExecutionObject::ProcessFrameWait 가 호출 된 경우 false 에 해당한다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1ExecutionObject_1a7d4af5d0cd17ce667b3a6c7004781b28

uint64_t **GetProcessCycles**() *const*

프로세스 호출을 실행하기 위해 Device 에서 취한 사이클 수를 반환한다.

Return

Device 에서 프레임을 처리하기 위한 사이클 수에 해당한다.

float **GetProcessTimeInMilliSeconds**() *const*

프로세스 호출을 실행하기 위해 Device 에서 취한 밀리 초를 반환한다.

Return

Device 의 프레임을 처리 할 시간(밀리 초)에 해당한다.

class Executor

#include <executor.h>
지정된 구성과 실행 프로그램이 사용할 수 있는 Device 집합을 사용하여 Network 의 전체 실행을 관리한다.

Public Functions

```
Executor(DeviceType device_type, const  DeviceIds &ids, const  Configuration &configuration, int layers_group_id =
OCL_TIDL_DEFAULT_LAYERS_GROUP_ID)
```

http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl10DeviceTypeE
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl9DeviceIdsE
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl13ConfigurationE
Executor Object 를 생성한다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1Executor

Executor 는 필요한 ExecutionObject 를 생성하고 지정된 TI DL Network 를 초기화한다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1ExecutionObject
예는 아래와 같다:

```
Configuration configuration;
configuration.ReadFromFile("path to configuration file");
DeviceIds ids1 = {DeviceId::ID2, DeviceId::ID3};
Executor executor(DeviceType::EVE, ids, configuration);
```

Parameters

- **device_type: DSP 혹은 EVE 장치**
- **ids: 이 Executor 의 인스턴스가 사용하는 장치 집합**
- **configuration: Executor 를 초기화하는데 사용되는 Configuration**
  http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1Configuration
- **layers_group_id: 이 Executor 가 실행해야 하는 Layers group**

~Executor()

Executor 를 분해하고 Executor Object 가 사용하는 리소스를 해제한다.

const ExecutionObjects &GetExecutionObjects() const

http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl16ExecutionObjectsE
이 Executor 의 인스턴스에서 사용 가능한 Execution Objects 에 대한 unique_ptr 의 벡터를 반환한다.

Public Static Functions

static uint32_t GetNumDevices(DeviceType device_type)

TI DL 에서 사용할 수 있는 지정된 유형의 Device 수를 반환한다.

Return

사용 가능한 Device 수

Parameters

- device_type: DSP 또는 EVE/EVE Device

static std::string GetAPIVersion()

API 버전에 해당하는 문자열을 반환함

Return

<major_ver>.<minor_ver>.<patch_ver>.<git_sha>

```
class PipeInfo
```

#include <executor.h>
Execution Object 간에 파이프 출력 및 입력에 필요한 입출력을 기술한다.

```
class ArgInfo
```

#include <executor.h>
ExecutionObject 에 필요한 입출력 버퍼를 기술한다.

Public Types

```
enum Kind
```

ArgInfo 로 나타나는 인수의 형식을 열거한다.
http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#classtidl_1_1ArgInfo

Values:

```
BUFFER =0

SCALAR
```

Public Functions

```
ArgInfo(void *p, size_t size)
```

포인터와 메모리 덩어리의 크기를 비교하여 ArgInfo 객체를 구성한다.

`ArgInfo(void *p, size_t size, Kind kind)`

http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl4KindE
포인터와 메모리 덩어리의 크기 및 종류에 대한 ArgInfo 객체를 생성한다.

`void *ptr() const`

Return

ArgInfo 로 나타내지는 버퍼 또는 스칼라의 포인터

`size_t size() const`

Return

ArgInfo 로 나타내지는 버퍼 또는 스칼라의 크기

`class Exception`

#include <executor.h>
오류 보고에 사용된다.

예외로부터 상속

Public Functions

`virtual const char *what() const`

Return

에러 메시지와 그 위치를 기술하는 문자열

namespace **imgutil**

Functions

bool **PreProcImage**(Mat &*image*, char **ptr*, int16_t *roi*, int16_t *n*, int16_t *width*, int16_t *height*, int16_t *pitch*, int32_t *chOffset*, int32_t *frameCount*, int32_t *preProcType*)

PreProcImage - 이미지 데이터를 DL Network 가 기대하는 값 범위로 사전 처리한다.

Parameters

- **image: 입력 이미지 데이터(OpenCV 데이터 구조)**
- **ptr: TI DL 이 입력으로 받는 출력 버퍼**
- **roi: 관심 분야의 수**
- **n: 채널 수**
- **width: 입력 이미지 너비**
- **height: 입력 이미지 높이**
- **pitch: 각 라인의 출력 버퍼(ptr) 피치**
- **chOffset: 각 채널의 출력 버퍼(ptr) 오프셋**
- **frameCount: 입력 이미지 데이터의 프레임 수**
- **preProcType: 사전 처리 유형, 네트워크 구성에 저장됨s**

Building from sources

TIDL API 의 소스는 https://git.ti.com/tidl/tidl-api 에서 확인할 수 있다.
https://git.ti.com/tidl/tidl-api
tidl-api/makefile 은 API, viewer 및 예제를 빌드하기 위한 Target 을 포함한다.
makefile 은 EVM 에서 원시 컴파일을 지원하고 x86/Linux 에서 cross compilation 을 지원한다.

Frequently Asked Questions

- malloc_ddr 에서 assertion 오류가 발생하는 이유는 무엇인가 ?
  – free 되지 않는 할당
    http://downloads.ti.com/mctools/esd/docs/tidl-api/faq/out_of_memory.html#allocations-not-freed
  – OpenCL global memory 부족
    http://downloads.ti.com/mctools/esd/docs/tidl-api/faq/out_of_memory.html#insufficient-opencl-global-memory

Why do I get an assertion failure from malloc_ddr ?

응용 프로그램 실행이 다음 오류 메시지와 함께 실패한다:

```
tidl: device_alloc.h:31: T* tidl::malloc_ddr(size_t) [with T = char; size_t = unsigned int]: Assertion `val != nullptr' failed
```

Allocations not freed

한 가지 가능한 이유는 응용 프로그램의 이전 실행이 중단되었으며(예: Ctrl-C 사용) 할당을 해제하지 않았기 때문이다.
"-c" 옵션과 함께 ti-mct-heap-check 명령을 사용하여 정리하라.

```
root@am57xx-evm:~# ti-mct-heap-check -c
-- ddr_heap1 ------------------------------
   Addr : 0xa2000000
   Size : 0xa000000
   Avail: 0xa000000
   Align: 0x80
-----------------------------------------
```

Insufficient OpenCL global memory

또 다른 가능한 이유는 EXTMEM_HEAP_SIZE 및 PARAM_HEAP_SIZE 를 사용하는 구성에 지정된
총 메모리 요구 사항이 OpenCL 에 사용할 수 있는 기본 메모리를 초과한다는 것이다.
다음 지침에 따라 CMEM(OpenCL 에서 사용할 수 있는 연속 메모리)의 양을 늘리도록 한다.

```
$ sudo apt-get install device-tree-compiler # In case dtc is not already installed
$ scp root@am57:/boot/am57xx-evm-reva3.dtb .
$ dtc -I dtb -O dts am57xx-evm-reva3.dtb -o am57xx-evm-reva3.dts
$ cp am57xx-evm-reva3.dts am57xx-evm-reva3.dts.orig
$ # increase cmem block size
$ diff -u am57xx-evm-reva3.dts.orig am57xx-evm-reva3.dts
--- am57xx-evm-reva3.dts.orig    2018-01-11 14:47:51.491572739 -0600
+++ am57xx-evm-reva3.dts    2018-01-16 15:43:33.981431971 -0600
@@ -5657,7 +5657,7 @@
         };

         cmem_block_mem@a0000000 {
-            reg = <0x0 0xa0000000 0x0 0xc000000>;
+            reg = <0x0 0xa0000000 0x0 0x18000000>;
             no-map;
             status = "okay";
             linux,phandle = <0x13c>;
@@ -5823,7 +5823,7 @@
         cmem_block@0 {
             reg = <0x0>;
             memory-region = <0x13c>;
-            cmem-buf-pools = <0x1 0x0 0xc000000>;
+            cmem-buf-pools = <0x1 0x0 0x18000000>;
         };

         cmem_block@1 {
$ dtc -I dts -O dtb am57xx-evm-reva3.dts -o am57xx-evm-reva3.dtb
$ scp am57xx-evm-reva3.dtb root@am57:/boot/
# reboot to make memory changes effective (run "cat /proc/iomem" to check)
```

자세한 내용은 OpenCL User's Guide 의 OpenCL 에 대한 DDR3 파티션 변경 섹션을 참조하라.
http://downloads.ti.com/mctools/esd/docs/opencl/memory/ddr-partition.html#change-ddr3-partition-for-opencl

Important Notice

Texas Instruments Incorporated 및 자회사(TI) 는 JESD46 당 반도체 제품 및
서비스에 대한 수정, 개선 및 기타 변경 사항, 최신 문제 및
JESD48 최신 제품에 대한 제품 또는 서비스를 중단 할 수 있는 권리를 보유한다.
구매자는 주문을 하기 전에 최신 관련 정보를 입수해야하며 그러한 정보가 최신이고 완전한지 확인해야 한다.
모든 반도체 제품(이하 "components" 라고도 함)은 주문 확인시 제공된 TI 판매 약관에 따라 판매된다.

TI 는 TI 제품의 판매 조건에 따라 보증 기간에 따라 해당 components 의 성능을 판매 시점의 해당 사양으로 보증한다.
테스트 및 기타 품질 관리 기술은 TI 가 이 보증을 지원하는데 필요한 것으로 판단되는 범위까지 사용된다.
관련 법률에 의거하여 위임된 경우를 제외하고는
각 components 의 모든 매개 변수 테스트가 반드시 수행되는 것은 아니다.

TI 는 애플리케이션 지원이나 구매자 제품의 설계에 대해 어떠한 책임도 지지 않는다.
구매자는 TI components 를 사용하여 제품 및 응용 프로그램을 담당한다.
구매자의 제품 및 응용 프로그램과 관련된 위험을 최소화하기 위해
구매자는 적절한 설계 및 운영 안전 장치를 제공해야 한다.

TI 는 TI componenets 또는 서비스가 사용되는 조합, 기계 또는 프로세스와 관련된
모든 특허권, 저작권, 마스크 작업 권한 또는 기타 지적 재산권에 대해 명시적
또는 묵시적인 라이센스가 부여됨을 보증하거나 진술하지 않는다.
타사 제품 또는 서비스와 관련하여 TI 가 게시한 정보는
그러한 제품 또는 서비스를 사용하기 위한 라이센스 또는 보증 또는 보증을 구성하지 않는다.
그러한 정보를 사용하려면 제 3 자의 특허 또는 기타 지적 재산권에 의거한
제 3 자의 라이센스 또는 TI 의 특허 또는 기타 지적 재산권에 대한 TI 의 라이센스가 필요할 수 있다.

TI 데이터 북 또는 데이터시트에 있는 TI 정보의 상당 부분을 복제하는 것은
복제본이 변경되지 않은 경우에만 허용되며 모든 관련 보증, 조건, 제한 사항 및 주의 사항이 수반된다.
TI 는 변경된 문서에 대해 책임을 지지 않는다.
제 3 자의 정보에는 추가 제한 사항이 적용될 수 있다.

해당 components 또는 서비스에 대해 TI 가 명시한 매개 변수와 다른 진술을 포함한 TI components
또는 서비스의 재판매는 관련된 TI components 또는 서비스에 대한
모든 명시적 및 묵시적 보증을 나타내며 불공정하고 기망적인 비즈니스 관행이다.
TI 는 그러한 진술에 대해 책임을 지지 않는다.

구매자는 TI 가 제공할 수 있는 응용 프로그램 관련 정보 또는 지원에도 불구하고
해당 제품과 관련된 모든 법척, 규제 및 안전 관련 요구 사항 및
해당 응용프로그램에서의 TI componets 사용에 대한 전적인 책임이 있음을 인정하고 이에 동의한다.
구매자는 실패의 위험한 결과를 예측하고, 실패 및 그 결과를 모니터하고,
해를 입히고 적절한 조취를 취할 수 있는 실패 가능성을 줄이는
안전 장치를 작성하고 구현하는데 필요한 모든 전문 기술을 보유하고 있음을 표명하고 동의한다.
구매자는 안전에 중요한 응용 프로그램에서 TI 구성 요소의 사용으로 인해
발생하는 모든 손해에 대해 TI 및 그 대리인은 책임지지 않는다.

경우에 따라 안전 관련 응용 프로그램을 용이하게 하기 위해 TI 구성 요소가 특별히 승격 될 수 있다.
이러한 구성 요소를 통해 TI 의 목표는 고객이 해당 기능 안전 표준 및
요구 사항을 충족하는 자체 최종 솔루션을 설계 및 제작할 수 있도록 지원하는 것이다.
그럼에도 불구하고 그러한 구성 요소는 이 조항의 적용을 받는다.

TI components 는 당사자의 권한을 가진 담당자가
해당 사용에 대한 특별 계약을 체결하지 않는 한
FDA Class III(또는 유사한 생명 유지 의료 장비)에서의 사용이 허가되지 않는다.

TI 가 군용 등급 또는 "Enhanced Plastic" 으로 특별히 지정한 TI components 만
군사/항공우주 응용 프로그램 또는 환경에서 사용하도록 설계 및 의도되었다.
구매자는 그렇게 지정되지 않은 TI 구성 요소의 군대 또는 항공우주 기체에의 사용은 전적으로 구매자의 위험 부담이며
구매자는 그러한 사용과 관련된 모든 법적 및 요구 사항의 준수에 전적으로 책임이 있음을 인정하고 동의한다.

TI 는 주로 자동차 용으로 ISO/TS16949 요구 사항을 충족하는 특정 부품을 지정했다.
그렇게 지정되지 않은 components 는 자동차 용으로 설계되거나 의도된 것이 아니며;
TI 는 그러한 components 가 이와 같은 요구 사항을 충족시키지 못하는 것에 대해 책임을 지지 않는다.

Disclaimer

이 사이트의 모든 컨텐츠 및 자료는 "있는 그대로" 제공된다.
TI 및 해당 공급 업체는 이 자료의 적합성에 대한 어떠한 진술도 하지 않으며
이 자료와 관련한 모든 보증 및 조건(상품성, 특정 목적에의 적합성,
제목에 대한 묵시적인 모든 보증 및 조건을 포함하되 이에 국한되지 않음)을 부인한다.
제 3 자의 지적 재산권을 침해하지 않으며 권리를 침해하지 않는다.
ESTOPPEL 또는 기타 방법에 의한 명시적 혹은 묵시적 사용권은 TI 가 부여한 것이다.
이 사이트의 정보를 사용하는 경우 제 3 자 라이센스 또는 TI 로부터의 라이센스가 필요 할 수 있다.

이 사이트의 내용에는 사용에 대한 특정 지침이나 제한 사항이 포함되어 있거나 해당 내용이 적용될 수 있다.
이 사이트의 모든 게시 및 내용은 사이트 이용 약관의 적용을 받는다.
이 내용을 사용하는 제 3 자는 제한이나 지침을 준수하고 이 사이트의 이용 약관을 준수하는데 동의한다.
TI 와 공급 업체는 언제든지 내용, 자료, 제품, 프로그램 및
서비스에 대한 수정, 삭제, 수정, 개선 및 기타 변경을 할 권리가 있으며
제품, 프로그램 또는 서비스를 사전 통보없이 언제든지 이동하거나 중단 할 수 있다.