OPEN AI LAB
开 放 智 能

Tengine后端之OpenDLA概述

LeiWang1999（wanglei21c@mails.ucas.ac.cn）

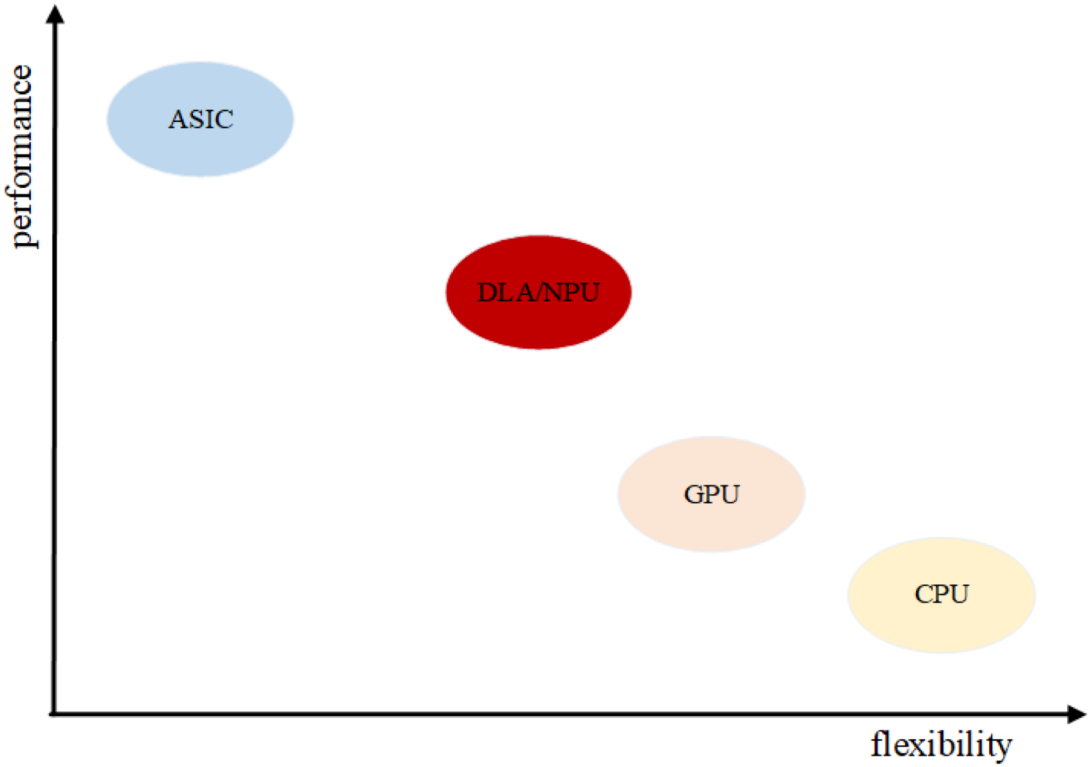
中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

智能计算机研究中心
Research Center for Intelligent Computing Systems

Outline

- Brief Introduction of Deep Learning Accelerator
- Nvidia Deep Learning Accelerator Overview and FPGA Implementation
- Integrate Tengine with NVDLA

Deep Learning Heterogeneous Computing



Tradeoff between performance and flexibility

ASIC设计缺陷：

- 实现大规模网络不现实
- 开发周期过长

NPU需要在能效上接近ASIC，又能保证通用性，所以我们需要抽象出这一类算法的计算特征和仿存特征：

计算特征：

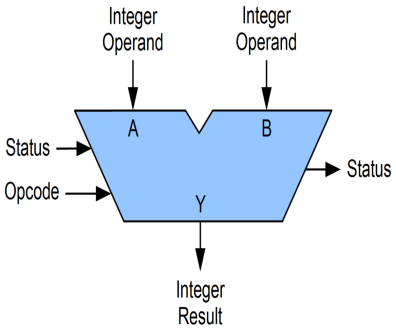
99% 以上的矩阵和向量操作

访存特征：

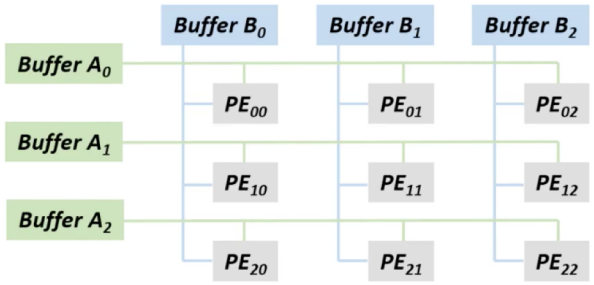
具有可解耦性和可重用性

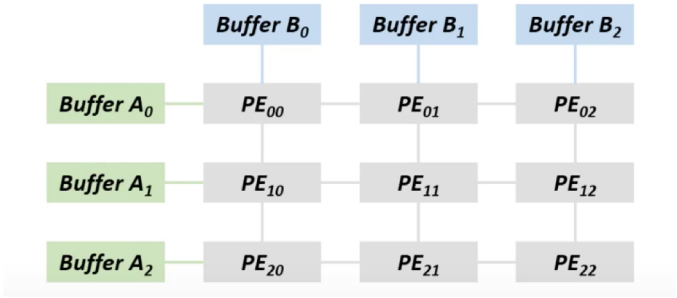
内容参考自：《智能计算系统》 陈云霁

ALU vs PE Array

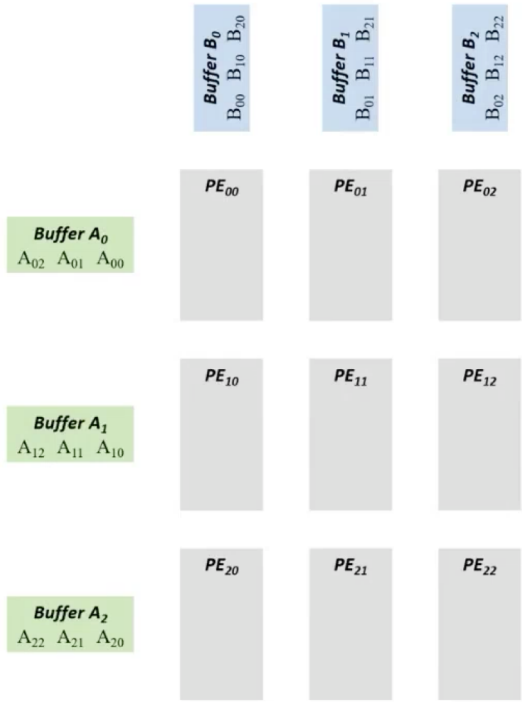


ALU





扇出太大，频率上不去！



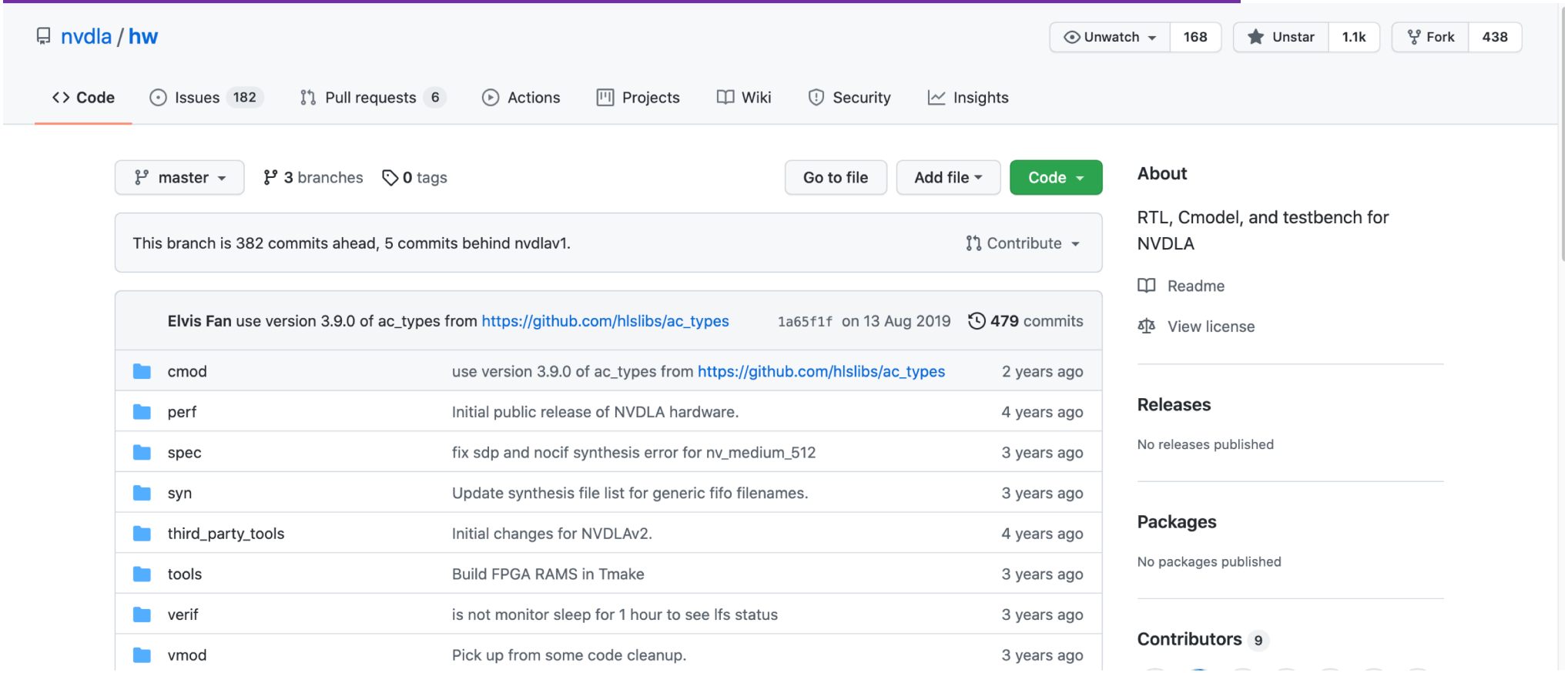
Systolic Array

Architecture	DianNao	NVDLA	TPUv1
Performance	16	64	256

参考链接： https://zhuanlan.zhihu.com/p/267799427

脉动阵列的简单介绍（动画引用：
https://www.bilibili.com/video/BV1ih411o7Yy?p=7

Nvidia Deep Learning Accelerator >> hardware repo



Nvidia Deep Learning Accelerator >> spec

master hw / spec / defs / nv_small.spec / <> Jump to Go to file ...

Nvidia-DorisLei adding missing file RAMPDP_128X32_GL_M2_D2.v Latest commit 9eefdf7 on 25 May 2018 History

2 contributors

43 lines (40 sloc) 1.09 KB Raw Blame

```
1  //#define NV_SMALL 1
2  #define FEATURE_DATA_TYPE_INT8
3  #define WEIGHT_DATA_TYPE_INT8
4  #define WEIGHT_COMPRESSION_DISABLE
5  #define WINOGRAD_DISABLE
6  #define BATCH_DISABLE
7  #define SECONDARY_MEMIF_DISABLE
8  #define SDP_LUT_DISABLE
9  #define SDP_BS_ENABLE
10 #define SDP_BN_ENABLE
11 #define SDP_EW_DISABLE
12 #define BDMA_DISABLE
13 #define RUBIK_DISABLE
14 #define RUBIK_CONTRACT_DISABLE
15 #define RUBIK_RESHAPE_DISABLE
16 #define PDP_ENABLE
17 #define CDP_ENABLE
18 #define RETIMING_DISABLE
19 #define MAC_ATOMIC_C_SIZE_8
20 #define MAC_ATOMIC_K_SIZE_8
21 #define MEMORY_ATOMIC_SIZE_8
22 #define MAX_BATCH_SIZE_x
23 #define CBUF_BANK_NUMBER_32
24 #define CBUF_BANK_WIDTH_8
```

Nvidia Deep Learning Accelerator >> spec

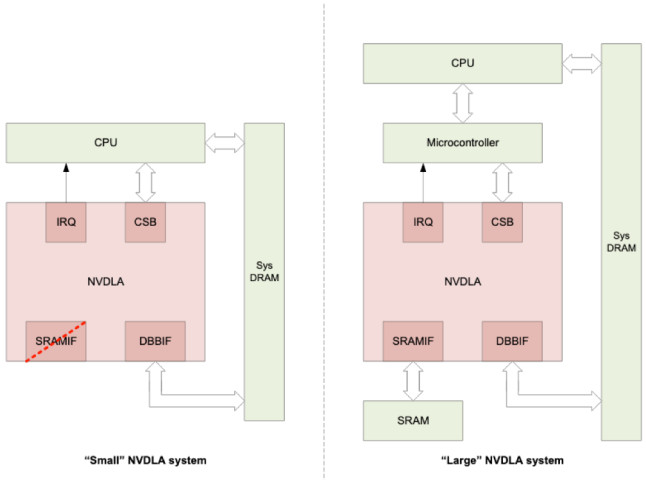
master ▾ hw / spec / defs / Go to file Add file ▾ ···

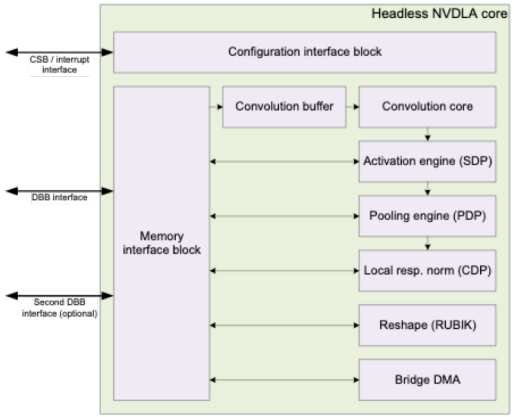
This branch is 382 commits ahead, 5 commits behind nvdlav1. Contribute ▾

ellenzhangnvdla and nvdsmith fix sdp and nocif synthesis error for nv_medium_512 a8619ac on 22 Jul 2018 History

..		
Makefile	Update designware enable flow:	3 years ago
README.md	Initial changes for NVDLAv2.	4 years ago
nv_large.spec	retiming disable in nv_large.spec	3 years ago
nv_medium_1024_full.spec	1. Fix DBB agent bug in burst length > 0 cases	3 years ago
nv_medium_512.spec	1. Fix DBB agent bug in burst length > 0 cases	3 years ago
nv_small.spec	adding missing file RAMPDP_128X32_GL_M2_D2.v	3 years ago
nv_small_256.spec	update mc burst length from 4 to 1, and update DMA read client num fr...	3 years ago
nv_small_256_full.spec	1. Fix DBB agent bug in burst length > 0 cases	3 years ago
projects.spec	fix sdp and nocif synthesis error for nv_medium_512	3 years ago

NAME	nv_full	nv_large	nv_medium_1024_full	nv_medium_512	nv_small_256_full	nv_small_256	nv_small
DATA TYPE	FP16/INT16/INT8	INT8	INT8	INT8	INT8	INT8	INT8
# MAC_CELL (ATOMIC_K)	64	64	32	16	8	8	8
CBUF_BANK_WIDTH (ATOMIC_C)	32	32	32	32	32	32	8
CBUF_BANK_DEPTH	512	512	512	512	128	128	512
CBUF_BANK_NUM	16	16	32	32	32	32	32
SDP_BS/BN_THROUGHPUT	16	16	8	4	2	1	1
SDP_EW_THROUGHPUT	4	4	2	X	1	X	X
PDP_THROUGHPUT	8	8	4	2	1	1	1
CDP_THROUGHPUT	8	8	4	2	1	1	1
DRAM IF Data Bus Width	512	256	256	128	64	64	64
SRAM IF Data Bus Width	512	256	256	X	64	X	X
RUBIK / BDMA	YES	NO	NO	NO	NO	NO	NO





Nvidia Deep Learning Accelerator >> generate RTL Code

http://nvdla.org/hw/v2/environment_setup_guide.html

Project name setup ¶

During tree.make file setup process, project name has been specified. The project name could be changed after tree.make generation. It's allowed to build multiple projects. To change target project names, open generated TOT/tree.make , and modify line which starts with projects, for example, we can remove other project name, only build nv_small:

```
##======================
## Project Name Setup, multiple projects supported
##======================
PROJECTS := nv_small
```

```
root@c6eb93bc312b:/usr/local/nvdla/nvdla_hw# ./tools/bin/tmake -build vmod
[TMAKE]: building nv_large in spec/defs
[TMAKE]: building nv_large in spec/manual
[TMAKE]: building nv_large in spec/odif
[TMAKE]: building nv_large in vmod/vlibs
[TMAKE]: building nv_large in vmod/include
[TMAKE]: building nv_large in vmod/rams/model
[TMAKE]: building nv_large in vmod/rams/synth
[TMAKE]: building nv_large in vmod/rams/fpga/model
[TMAKE]: building nv_large in vmod/fifos
[TMAKE]: building nv_large in vmod/nvdla/apb2csb
[TMAKE]: building nv_large in vmod/nvdla/cdma
[TMAKE]: building nv_large in vmod/nvdla/cbuf
[TMAKE]: building nv_large in vmod/nvdla/csc
[TMAKE]: building nv_large in vmod/nvdla/cmac
[TMAKE]: building nv_large in vmod/nvdla/cacc
[TMAKE]: building nv_large in vmod/nvdla/sdp
[TMAKE]: building nv_large in vmod/nvdla/pdp
[TMAKE]: building nv_large in vmod/nvdla/cfgrom
[TMAKE]: building nv_large in vmod/nvdla/cdp
[TMAKE]: building nv_large in vmod/nvdla/bdma
[TMAKE]: building nv_large in vmod/nvdla/rubik
[TMAKE]: building nv_large in vmod/nvdla/car
[TMAKE]: building nv_large in vmod/nvdla/glb
[TMAKE]: building nv_large in vmod/nvdla/csb_master
```

```
docker pull farhanaslam/nvdla
```

Two chisel Solutions :

CHIPYARD

https://github.com/ucb-bar/chipyard

Painting

soDLA

soDLA-publishment

http://nprocessor.com

https://github.com/soDLA-publishment/soDLA

Nvidia Deep Learning Accelerator >> FPGA Optimization

RAM 的替换

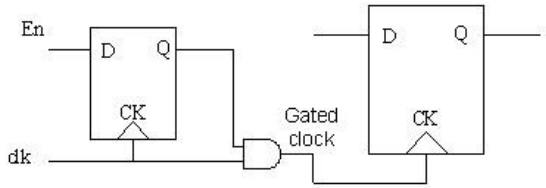
RESOURCE	UTILIZATION	AVAILABLE	Utilization %
LUT	421698	218600	192.91
LUTRAM	8	70400	0.01
FF	92622	437200	22.33
DSP	33	900	3.67
BUFG	12	32	37.5

RESOURCE	UTILIZATION	AVAILABLE	Utilization %
LUT	77248	218600	35.34
LUTRAM	372	70400	0.53
FF	87999	437200	20.13
BRAM	95	545	17.43
DSP	33	900	3.67
BUFG	2	32	6.25

MAC 阵列映射到 DSP

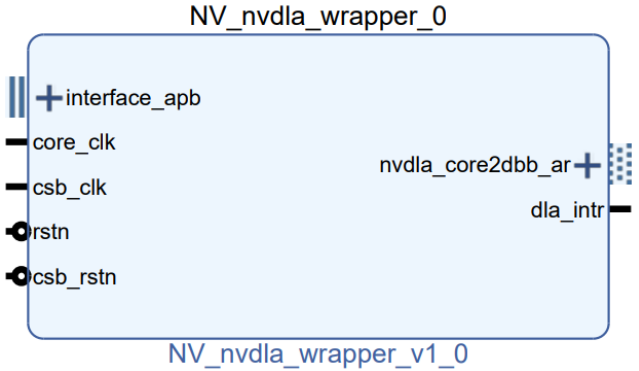
```
(* use_dsp = "yes" *)module NV_NVDLA_CMAC_core (
   nvdla_core_clk                //|< i
  ,nvdla_core_rstn               //|< i
    ...
  ,mac2accu_pvld                 //|> o
  );
```

删掉Clock Gating电路



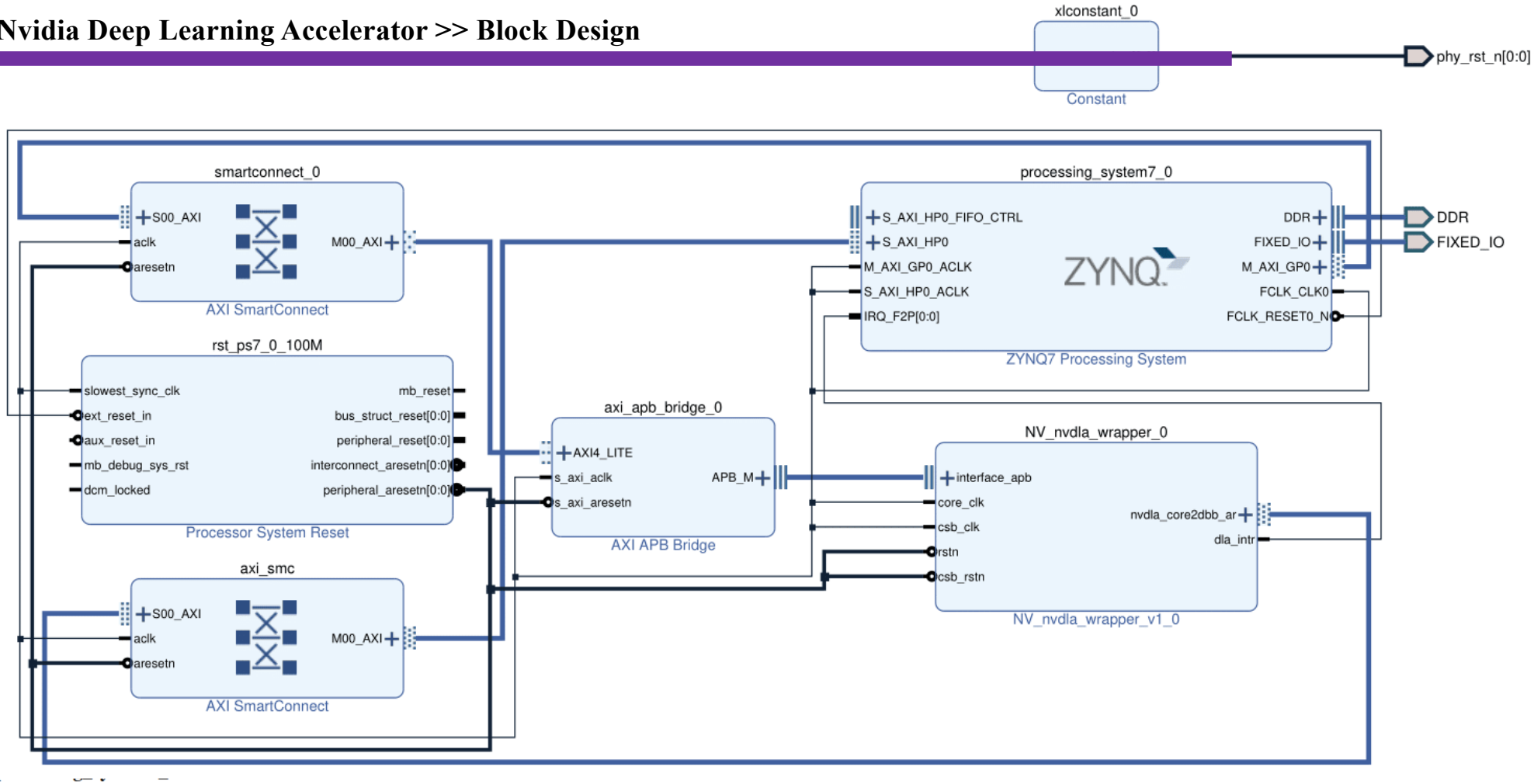
- VLIB_BYPASS_POWER_CG
- NV_FPGA_FIFOGEN
- FIFOGEN_MASTER_CLK_GATING_DISABLED
- FPGA
- SYNTHESIS

控制总线协议的封装



1. core_clk 工作时钟
2. csb_clk 控制总线时钟
3. dla_intr 事件中断
4. interface_apb 控制总线（预留4KB寻址）
5. nvdla_core2dbb_ar 外部存储总线

详细：https://zhuanlan.zhihu.com/p/378202360

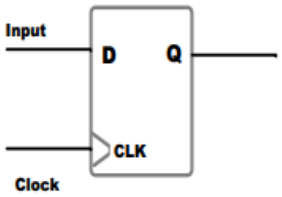


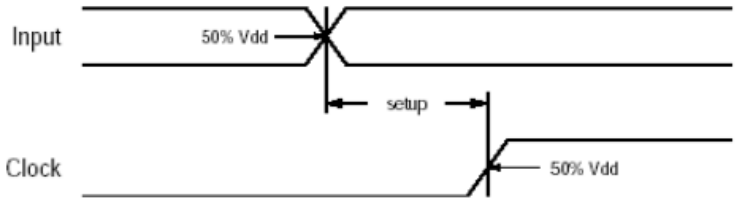
axi_apb_bridge_0	1	89	144	0	0
NV_nvdla_wrapper_0	1	76888	87943	95	42
In Total	6	81902(37.47%)	94753(21.67%)	95(17.43%)	42(4.67%)

测试与分析|最大工作频率与功耗

STA 静态时序分析（ https://www.bilibili.com/video/BV1if4y1p7Dq ）

建立时间，保证数据可以在给定时钟周期内到达触发器
保持时间，数据在被触发器采样后还能保持的时间

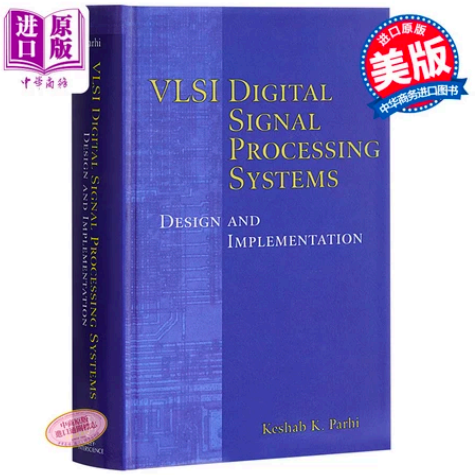




Timing Reports

Frequency	Worst Setup Slack	Worst Hold Slack	Total Negative Slack
25Mhz	29.048 ns	0.021 ns	0.000 ns
50Mhz	9.959 ns	0.007 ns	0.000 ns
75Mhz	3.832 ns	0.028 ns	0.000 ns
100Mhz	0.751 ns	0.030 ns	0.000 ns

Frequency	Worst Setup Slack	Worst Hold Slack	Total Negative Slack
150Mhz	-0.324 ns	0.033 ns	-9.728 ns



《VLSI 数字信号处理系统设计与实现》
https://detail.tmall.com/item.htm?id=619518182731

Zynq®-7000 SoC Family

		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz(1)			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support(2)	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support(2)	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA(2)	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security(3)	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM (# 36Kb Blocks)	1.8Mb (50)	2.5Mb (72)	3.8Mb (107)	2.1Mb (60)	3.3Mb (95)	4.9Mb (140)	9.3Mb (265)	17.6Mb (500)	19.2Mb (545)	26.5Mb (755)
	DSP Slices	66	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC(2)	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security(3)	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades: Commercial	-1			-1			-1			-1
	Speed Grades: Extended	-2			-2,-3			-2,-3			-2
	Speed Grades: Industrial	-1, -2			-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

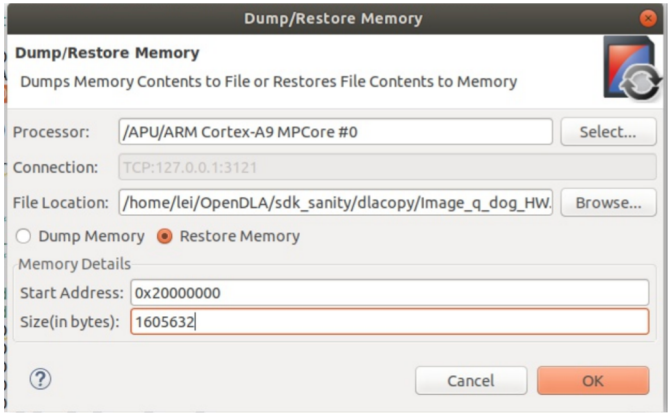
Zynq® UltraScale+™ MPSoCs: EG Devices

		Device Name[1]	ZU1EG	ZU2EG	ZU3EG	ZU4EG	ZU5EG	ZU6EG	ZU7EG	ZU9EG	ZU11EG	ZU15EG	ZU17EG	ZU19EG
Processing System (PS)	Application Processor Unit	Processor Core	**Quad-core** Arm® Cortex®-A53 MPCore™ up to 1.5GHz											
		Memory w/ECC	L1 Cache 32KB I / D per core, L2 Cache 1MB, on-chip Memory 256KB											
	Real-Time Processor Unit	Processor Core	**Dual-core** Arm Cortex-R5F MPCore™ up to 600MHz											
		Memory w/ECC	L1 Cache 32KB I / D per core, Tightly Coupled Memory 128KB per core											
	Graphic & Video Acceleration	Graphics Processing Unit	Mali™-400 MP2 up to 667MHz											
		Memory	L2 Cache 64KB											
	External Memory	Dynamic Memory Interface	x16: DDR4 w/o ECC; x32/x64: DDR4, LPDDR4, DDR3, DDR3L, LPDDR3 w/ ECC											
		Static Memory Interfaces	NAND, 2x Quad-SPI											
	Connectivity	High-Speed Connectivity	PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet											
		General Connectivity	2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO											
	Integrated Block Functionality	Power Management	Full / Low / PL / Battery Power Domains											
		Security	RSA, AES, and SHA											
		AMS - System Monitor	10-bit, 1MSPS – Temperature and Voltage Monitor											
PS to PL Interface			12 x 32/64/128b AXI Ports											
Programmable Logic (PL)	Programmable Functionality	System Logic Cells (K)	81	103	154	192	256	469	504	600	653	747	926	1,143
		CLB Flip-Flops (K)	74	94	141	176	234	429	461	548	597	682	847	1,045
		CLB LUTs (K)	37	47	71	88	117	215	230	274	299	341	423	523
	Memory	Max. Distributed RAM (Mb)	1.0	1.2	1.8	2.6	3.5	6.9	6.2	8.8	9.1	11.3	8.0	9.8
		Total Block RAM (Mb)	3.8	5.3	7.6	4.5	5.1	25.1	11.0	32.1	21.1	26.2	28.0	34.6
		UltraRAM (Mb)	-	-	-	13.5	18.0	-	27.0	-	22.5	31.5	28.7	36.0
	Clocking	Clock Management Tiles (CMTs)	3	3	3	4	4	4	8	4	8	4	11	11
	Integrated IP	DSP Slices	216	240	360	728	1,248	1,973	1,728	2,520	2,928	3,528	1,590	1,968
		PCI Express® Gen 3x16	-	-	-	2	2	-	2	-	4	-	4	5
		150G Interlaken	-	-	-	-	-	-	-	-	1	-	2	4
		100G Ethernet MAC/PCS w/RS-FEC	-	-	-	-	-	-	-	-	2	-	2	4
		AMS - System Monitor	1	1	1	1	1	1	1	1	1	1	1	1
	Transceivers	GTH 16.3Gb/s Transceivers	-	-	-	16	16	24	24	24	32	24	44	44
		GTY 32.75Gb/s Transceivers	-	-	-	-	-	-	-	-	16	-	28	28
	Speed Grades	Extended[2]	-1 -2 -2L			-1 -2 -2L -3					-1 -2 -2L -3			
		Industrial	-1 -1L -2											

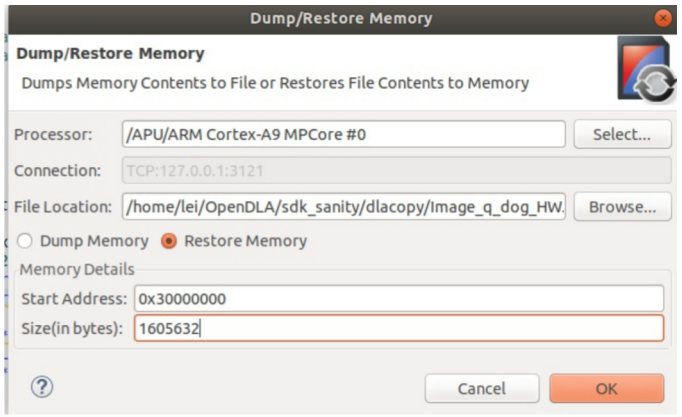
Notes:
1. For full part number details, see the Ordering Information section in DS891, *Zynq UltraScale+ MPSoC Overview*.
2. -2LE (Tj = 0°C to 110°C). For more details, see the Ordering Information section in DS891, *Zynq UltraScale+ MPSoC Overview*.

Nvidia Deep Learning Accelerator >> Sanity Test

1. 往0x20000000塞一张🐶的图片数据



2. 往0x30000000塞一张一样的🐶的图片数据



Sanity test 实现的功能：

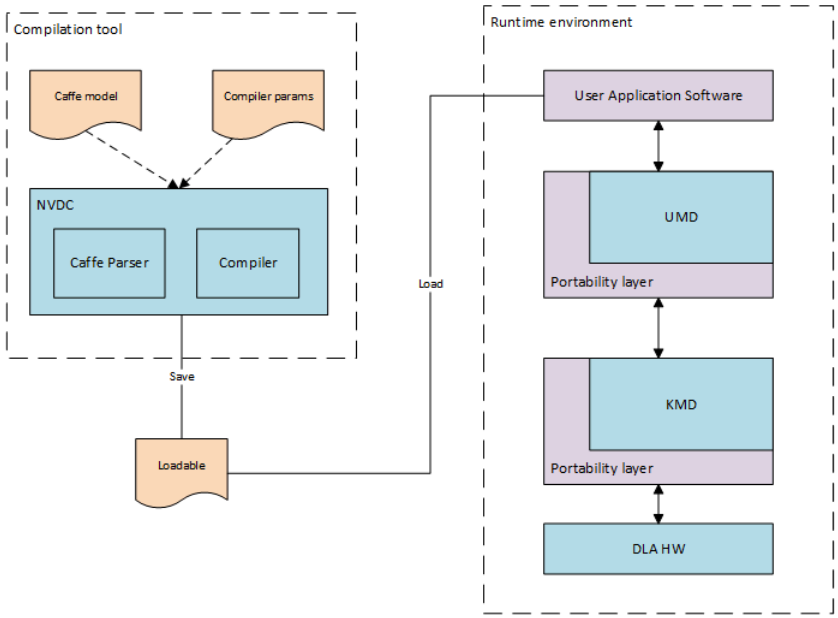
1. 通过配置寄存器调用SDP将 0x20000000 的 1605632 的数据搬运到 0x20200000
2. 搬运完毕之后将0x2020000的数据与0x30000000的golden数据进行比较

Test Pass !

Nvidia Deep Learning Accelerator >> Sanity Test

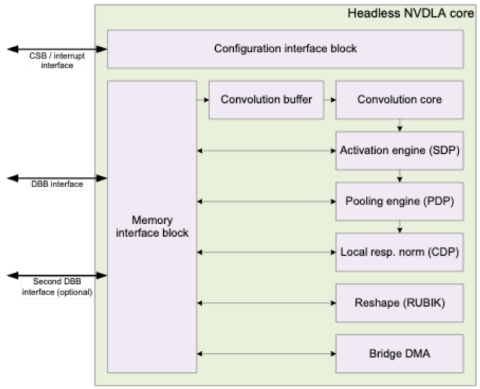
```
{
    unsigned int memory_value;

    /*
    printf("************************\n");
    printf("Begin NVDLA NV_SMALL Register Setting\n");
    printf("************************\n");
    */
    //mem_load(base_addr + 0x0, "CONV_SDP_0_input.dat");
    //mem_load(base_addr + 0x40000, "CONV_SDP_0_weight.dat");
    reg_write(NVDLA_SDP_S_POINTER_0, 0x0);
    reg_write(NVDLA_SDP_RDMA_S_POINTER_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LO_START_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_ACCESS_CFG_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LE_SLOPE_SCALE_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LO_SLOPE_SCALE_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LE_END_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_ACCESS_DATA_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_INFO_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LO_SLOPE_SHIFT_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LE_START_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_CFG_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LE_SLOPE_SHIFT_0, 0x0);
    reg_write(NVDLA_SDP_S_LUT_LO_END_0, 0x0);
    reg_write(NVDLA_SDP_D_CVT_OFFSET_0, 0x0);
    reg_write(NVDLA_SDP_D_DST_DMA_CFG_0, 0x1);
    reg_write(NVDLA_SDP_RDMA_D_SRC_SURFACE_STRIDE_0, 0x188000);
    reg_write(NVDLA_SDP_D_DST_LINE_STRIDE_0, 0xe00);
    reg_write(NVDLA_SDP_RDMA_D_SRC_LINE_STRIDE_0, 0xe00);
    reg_write(NVDLA_SDP_RDMA_D_SRC_BASE_ADDR_HIGH_0, 0x0);
    reg_write(NVDLA_SDP_RDMA_D_BRDMA_CFG_0, 0x1);
    reg_write(NVDLA_SDP_RDMA_D_BS_BATCH_STRIDE_0, 0x0);
    reg_write(NVDLA_SDP_D_DP_EW_ALU_CVT_SCALE_VALUE_0, 0x0);
    reg_write(NVDLA_SDP_RDMA_D_EW_LINE_STRIDE_0, 0x0);
    reg_write(NVDLA_SDP_RDMA_D_BN_BASE_ADDR_HIGH_0, 0x0);
    reg_write(NVDLA_SDP_D_DP_EW_ALU_SRC_VALUE_0, 0x0);
    reg_write(NVDLA_SDP_RDMA_D_NRDMA_CFG_0, 0x1);
```

The heart of NVDLA Compiler



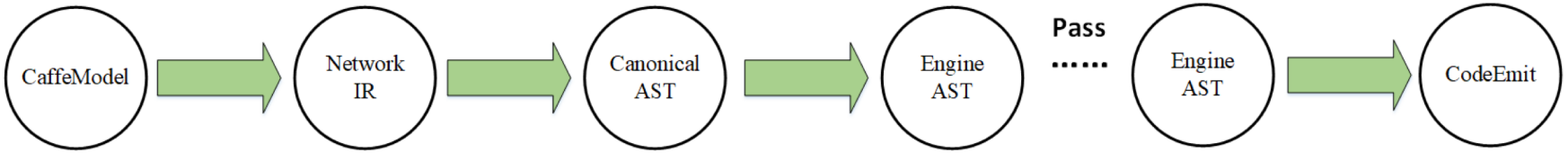
Four Phases in the compilation process

- **Caffe Parser**: read caffemodel and build Network IR
- **Canonical AST**: a compute graph without target device information
- **Engine AST:** a compute graph with target device information (Memory Alloc、Task Division …)
- **Code Emit**: serialize data to generate Loadable

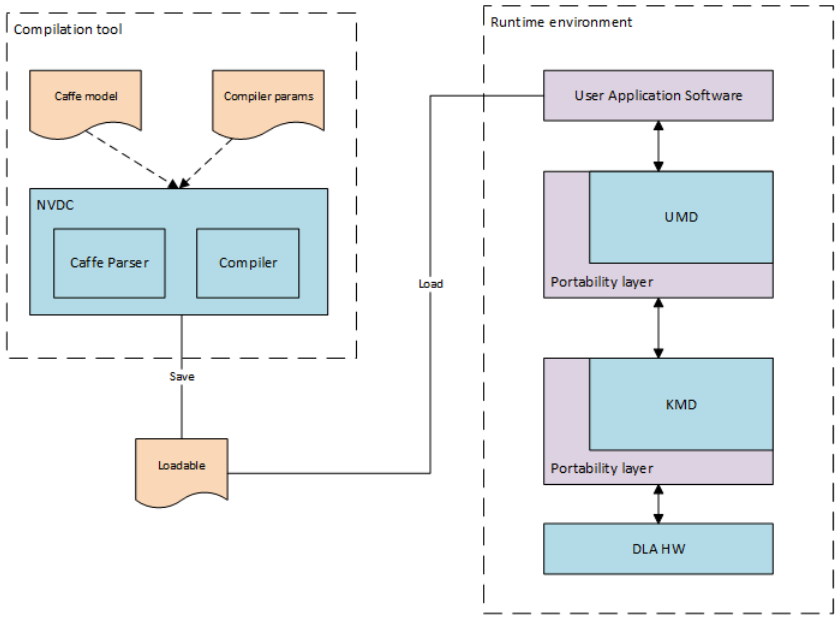


NVDLA INT8 量化笔记：

- https://zhuanlan.zhihu.com/p/378120804
- 只给出Layer的Input和Output Scale
- 权重数据用 Min Max 量化策略
- Weight -> int8 Bias->INt16



Runtime Environment



Runtime: an operator library (include DLA Operators and CPU operators) which hidden register level operations.

Petalinux (Linux Kernel 4.19) Based BOOT、Image.ub generation.

Detail：https://zhuanlan.zhihu.com/p/378202360

```
root@arm:~# insmod /lib/modules/4.19.0-xilinx-v2019.1/extra/opendla.ko
```

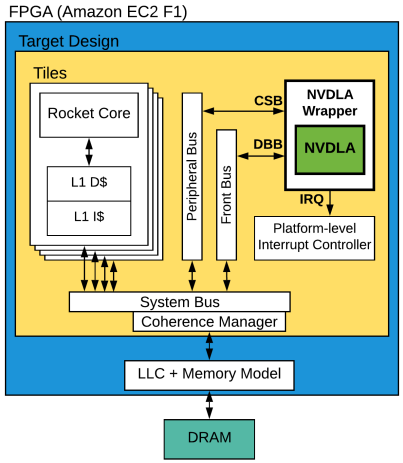
```
root@arm:~/OpenDLA/umd/out/apps/runtime/nvdla_runtime# ./nvdla_runtime --loadable ~/lenet-mnist-caffe/fast-
math.nvdla --image ~/lenet-mnist-caffe/mnist_image/0_7.jpg --rawdump
creating new runtime context...
Emulator starting
dlaimg height: 28 x 28 x 1: LS: 224 SS: 0 Size: 6272
submitting tasks...
Work Found!
Work Done
execution time = 298671.000000 us
Shutdown signal received, exiting
Test pass
root@arm:~/OpenDLA/umd/out/apps/runtime/nvdla_runtime# cat output.dimg
0 0 0 0 0 0 0 120 0 0 root@arm:~/OpenDLA/umd/out/apps/runtime/nvdla_runtime#
```

移植带带串口终端的Ubuntu 16.04::
https://zhuanlan.zhihu.com/p/392974835

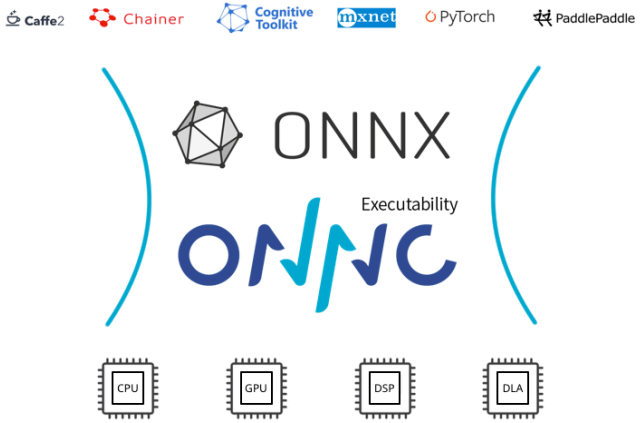
NVDLA Small 支持算子: Conv、Max/Min/AVG Pooling、ReLU ……　　**加速器设计之后的通病：缺少工具链的支持!**

Yolo 需要的另外的算子: **Reshape**、**Transpose**、Swish、HardSwish……

ITRI Yolov1-Tiny



firesim-nvdla



ONNC

1. 商业授权 **200万一年**
2. 支持的算子也很有限
3. NVDLA的Runtime不仅要管DLA的算子、CPU的算子也需要在Runtime实现

参考链接:

https://www.youtube.com/watch?v=sQ9oIjHF5ac

https://github.com/CSL-KU/firesim-nvdla

https://github.com/ONNC/onnc

Target

- 开源、免费、功能比同类型工具链更丰富
- 能够接受各种主流训练框架的模型
- 工具链本身支持量化，不需要依赖TensorRT
- 对异构计算系统有极好的兼容能力

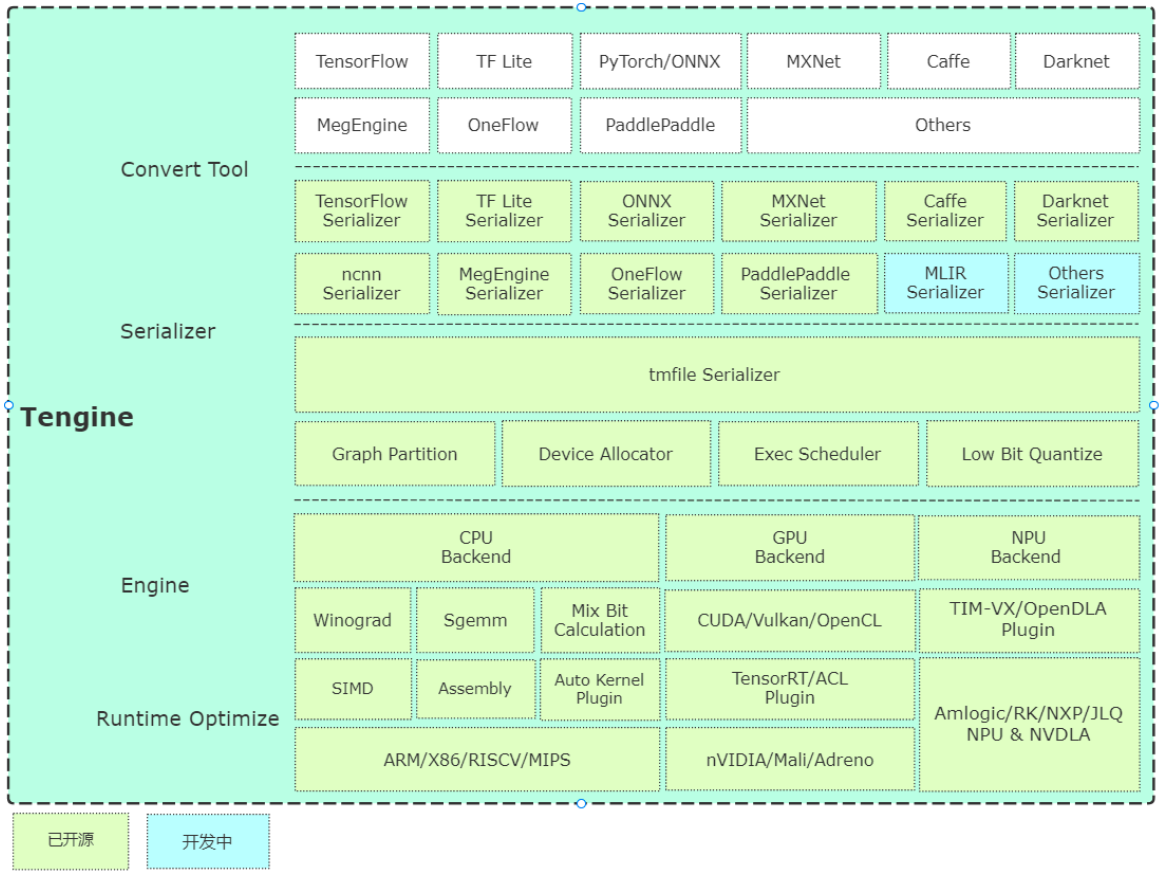
Tengine 简介

Tengine的项目目的：

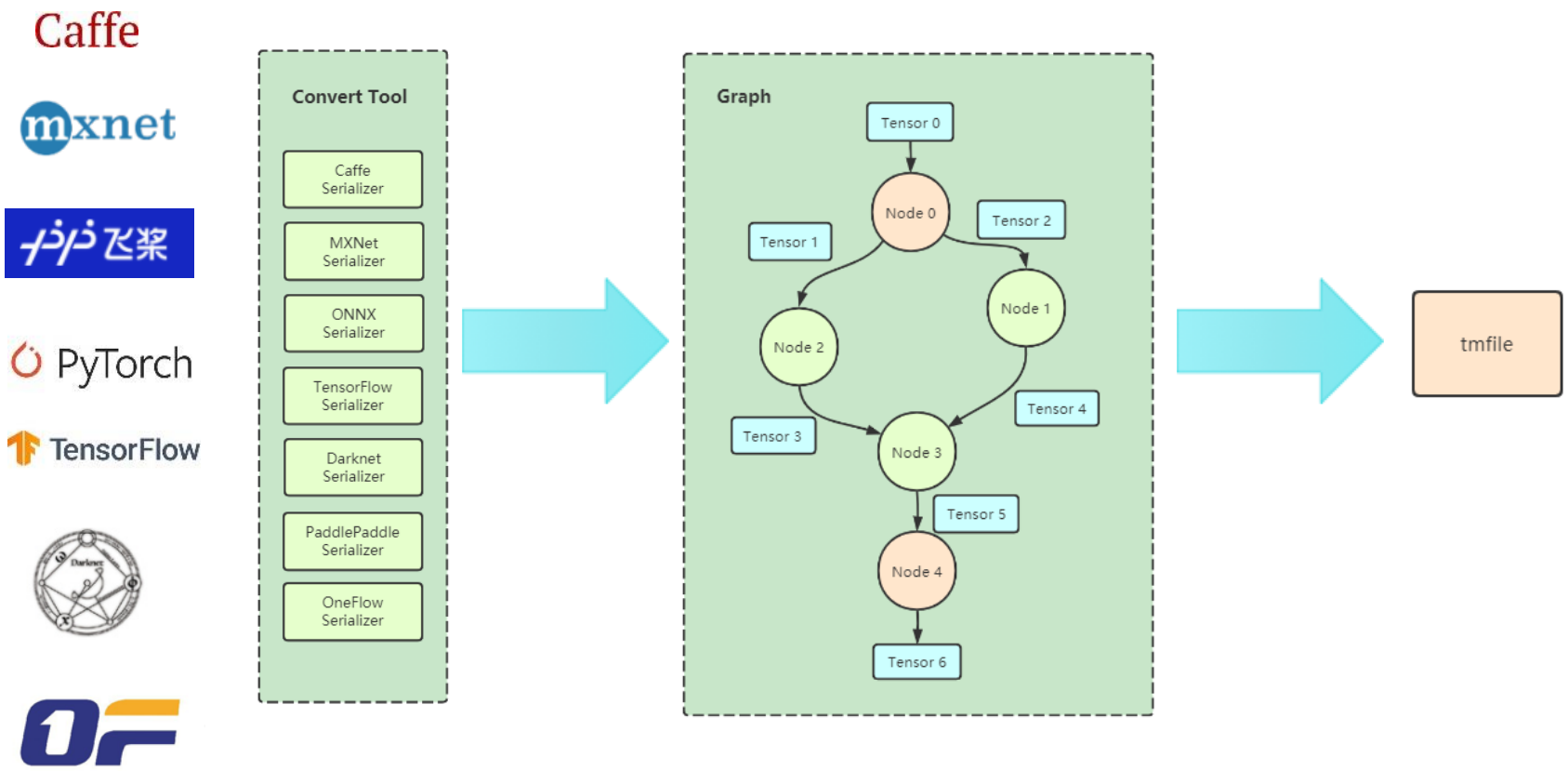
- 广受欢迎的边缘 AI 计算框架
- 快速对接各种CPU、GPU、NPU平台
- 充分发挥边缘AI计算设备的异构能力

项目地址：

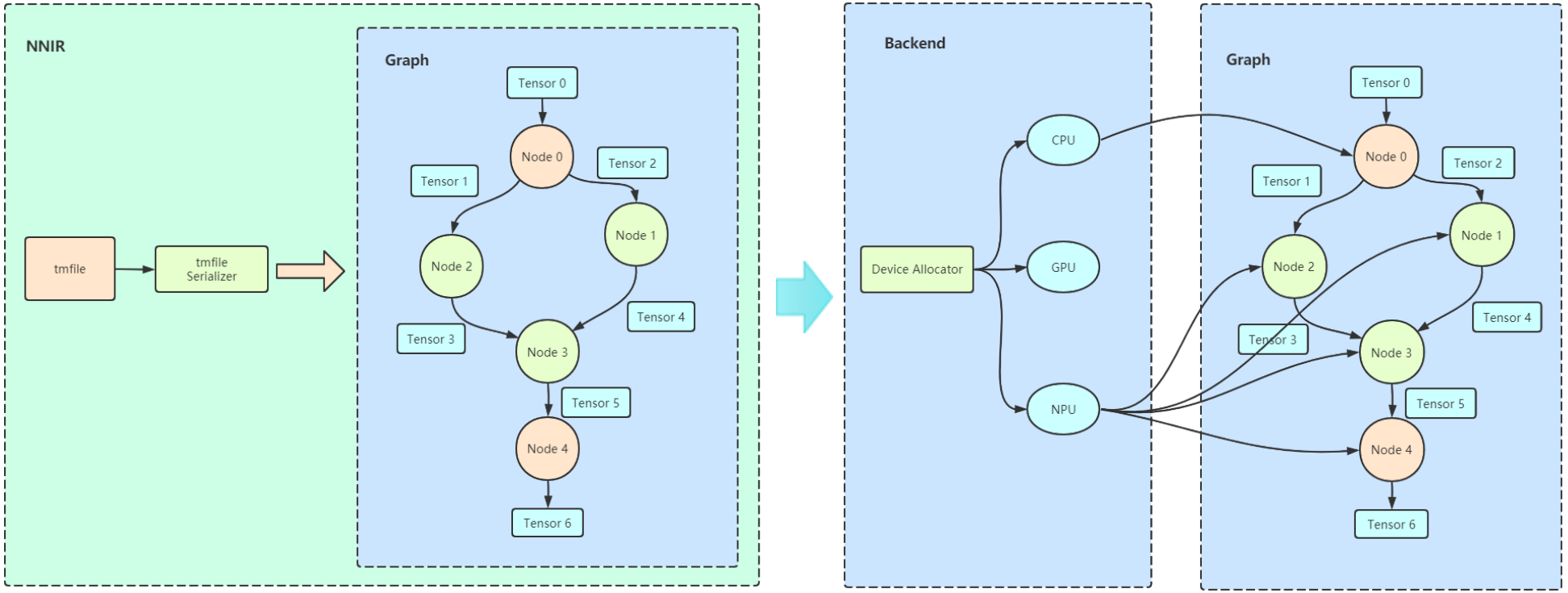
- https://github.com/OAID/Tengine
- https://gitee.com/OAL/Tengine



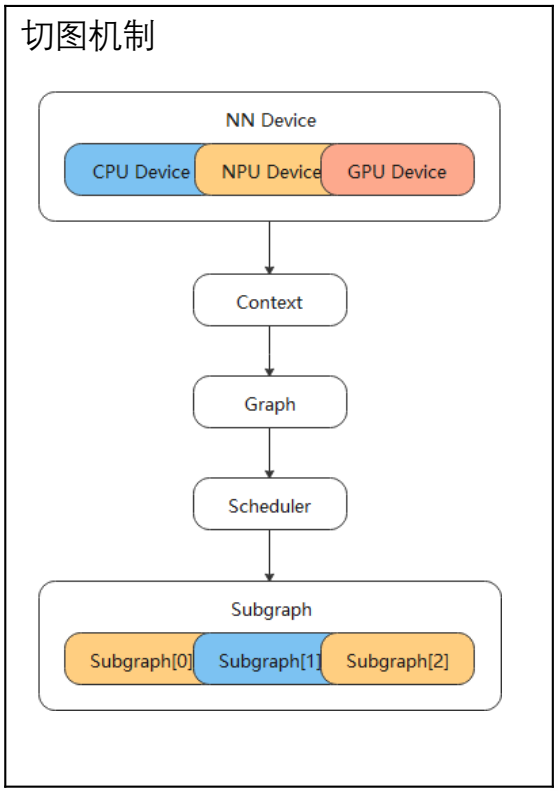
Tengine 模型转换

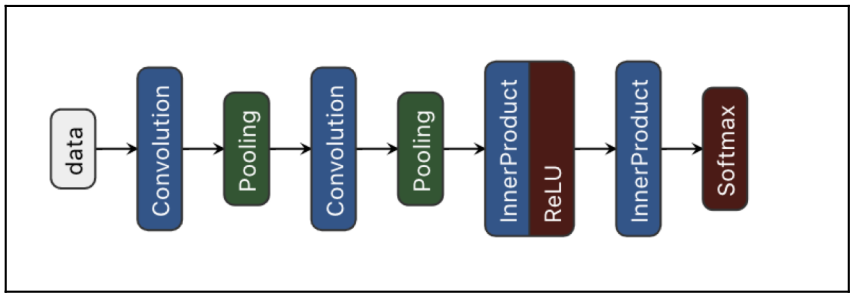


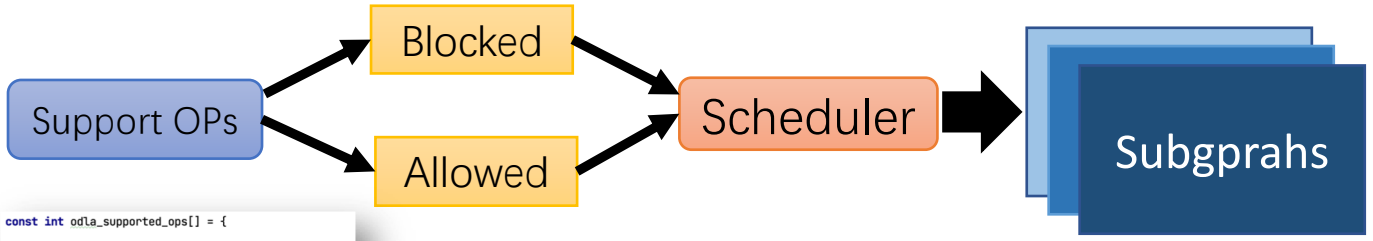
Tengine 模型运行



Tengine 切图机制







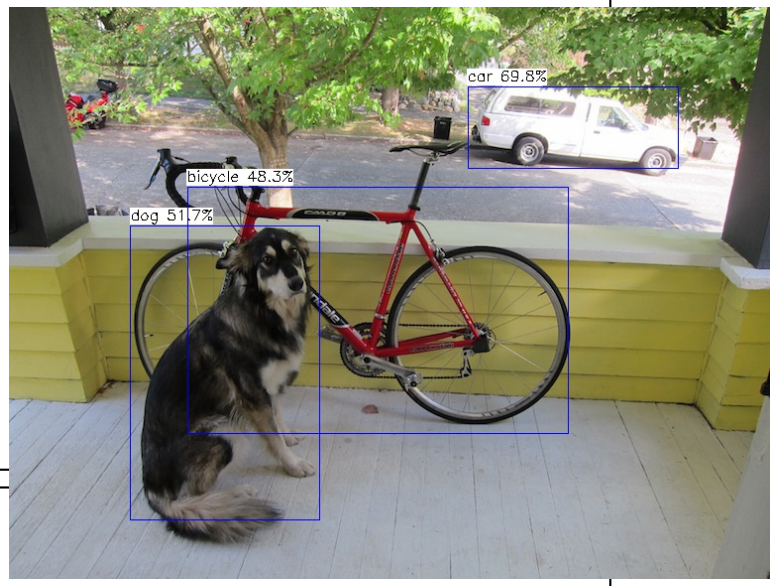

```
const int odla_supported_ops[] = {

    //    OP_GENERIC,
    //    OP_ABSVAL,
    //    OP_ADD_N,
    //    OP_ARGMAX,
    //    OP_ARGMIN,
    //    OP_BATCHNORM,
    //    OP_BATCHTOSPACEND,
    //    OP_BIAS,
    //    OP_BROADMUL,
    //    OP_CAST,
    //    OP_CEIL,
    //    OP_CLIP,
    //    OP_COMPARISON,
    //    OP_CONCAT,
    OP_CONST,
    OP_CONV,
    //    OP_CROP,
    //    OP_DECONV,
    //    OP_DEPTHTOSPACE,
    //    OP_DETECTION_OUTPUT,
    //    OP_DETECTION_POSTPROCESS,
```

```
int odla_split_graph(struct graph* ir_graph)
{
    struct device* cur_dev = ir_graph->attribute->context->device;
```

```
    cur_dev->allocator->describe(cur_dev, allowed_ops, blocked_ops, precision);

    split_graph_node_to_sub_graph(ir_graph, allowed_ops, blocked_ops, precision);

    release_vector(allowed_ops);
    release_vector(blocked_ops);
    release_vector(precision);
```

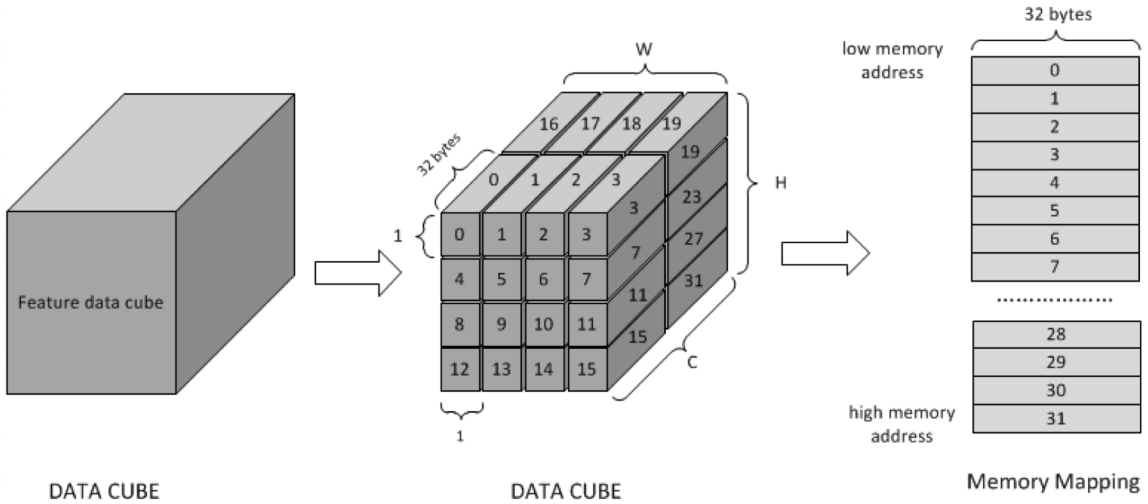
效果演示

```
$ cd examples
$ ./tm_classification_opendla -m /root/Tengine/models/resnet18-cifar10-nosoftmax-relu_int8.tmfile -i /root/Tengine/images/cat.jpg -g 32,32 -s 1,1,1
Mean value not specified, use default   104.0, 116.7, 122.7
tengine-lite library version: 1.4-dev
NVDLA time: 0.012502 seconds

model file : /root/Tengine/models/resnet18-cifar10-nosoftmax-relu_int8.tmfile
image file : /root/Tengine/images/cat.jpg
img_h, img_w, scale[3], mean[3] : 32 32 , 1.000 1.000 1.000, 104.0 116.7 122.7
Repeat 1 times, thread 1, avg time 12.62 ms, max_time 12.62 ms, min_time 12.62 ms
--------------------------------------
10.087049, 3
3.833079, 2
3.026115, 5
2.420892, 4
-0.403482, 0
--------------------------------------
```




```
$ cd <tengine-lite-root-dir>/build
$ cmake --build . --target tm_classification_opendla tm_yolox_opendla
$ cd examples
$ ./tm_yolox_opendla -m /root/Tengine/models/yolox_nano_relu_int8.tmfile -i /root/Tengine/images/dog.jpg -r 1
tengine-lite library version: 1.4-dev
Repeat 1 times, thread 1, avg time 1138.80 ms, max_time 1138.80 ms, min_time 1138.80 ms
--------------------------------------
detection num: 3
 2:  70%, [ 463,   80,  676,  163], car
16:  52%, [ 122,  220,  315,  517], dog
 1:  48%, [ 180,  181,  564,  430], bicycle
```

NVDLA 数据摆放



> In conclusion, mapping in memory follows pitch linear format. The order is C' (32byte) -> W -> H -> C (surfaces). Here C' changes fastest and C changes slowest.

For NVDLA small:

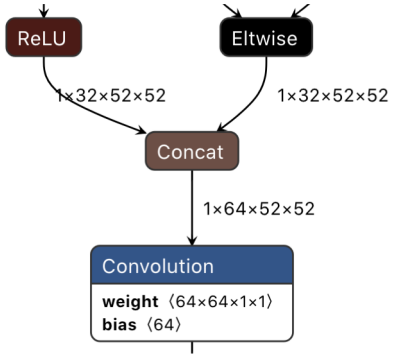
Address 0 : Channel 0 pixel 0	Address 8 : Channel 0 pixel 1
Address 1 : Channel 1 pixel 0	Address 9 : Channel 1 pixel 1
Address 2 : Channel 2 pixel 0	Address 10 : Channel 2 pixel 1
……	……
Address 7 : Channel 7 pixel 0	Address 15: Channel 7 pixel 1

NVDLA 数据摆放的代码实现与问题

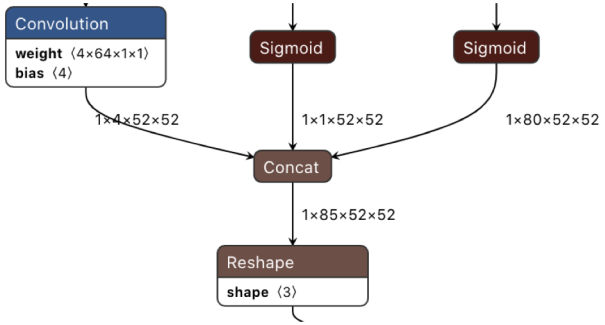
```
for (size_t n = 0; n < batch; n++){
  #pragma omp parallel for num_threads(max_thread)
  for (size_t c = 0; c < channel; c++)
  {
    NvU32 cquotient = c / atom_c_size;
    NvU32 cremainder = c % atom_c_size;
    for (size_t h = 0; h < height; h++)
    {
      for (size_t w = 0; w < width; w++)
      {
        size_t idx = n * channel * height * width + c * height * width + h * width + w;
        int8_t* _dst = (int8_t*)dst + idx;
        uint32_t _offset = (cquotient * surface_stride) + (h * line_stride) + (w * atom_k_size) + cremainder
+ n*c*h*w;
        *_dst = *((int8_t*)src + _offset);
      }
    }
  }
}
```

- **CPU实现摆放，且多张图的时候浪费太多时间!**
- **一些厂商会有硬件单元来做**
  - **芯原：**tensor process
  - **NVIDIA：**nvidia tensor core
- **NVDLA的RUBIK可能可以完成这部分工作，但是没有实现上层接口。**

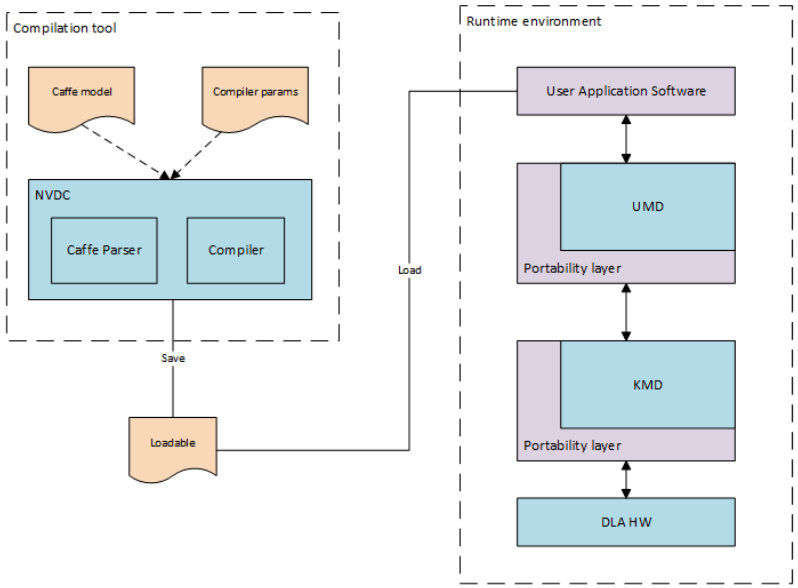
NVDLA可以支持的 Concat



NVDLA不可以支持的 Concat



Tengine NVDLA后端添加思路

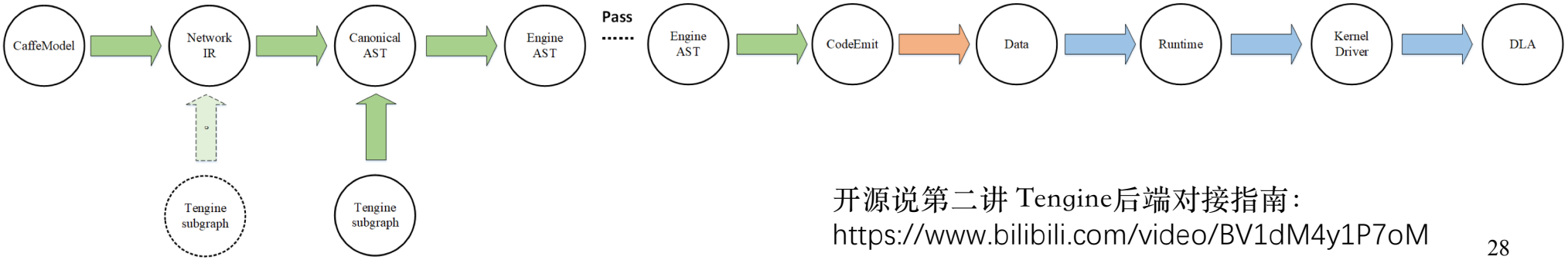


ONNC: 本质是一个Compiler，替换了 Compilation Tool

Tengine：一个Runtime框架，后端可以同时存在多个Runtime

```
if (this->compiler.priv()->emit(finalEngineAST, &this->loadable) != NvDlaSuccess) {
    fprintf(stderr, "Failed to emit Loadable Data. \n");
    return -1;
}
(void)this->loadable.priv()->serialize();

// Get Loadable Image Size
NvU64 loadableSize = 0;
this->loadable.priv()->getSerializedDataSize(&loadableSize);
if(!loadableSize){
    fprintf(stderr, "No Loadable Generated. \n");
    return -1;
}
NvU8 * buffer  = (NvU8 *)NvDlaAlloc(loadableSize);
```

```
// deserialize Loadable image
this->runtime->load(buffer, instance: 0);
```



开源说第二讲 Tengine后端对接指南：
https://www.bilibili.com/video/BV1dM4y1P7oM

Tengine NVDLA的后端添加

```
extern "C"
{
static struct interface odla_interface = {
        .init           = odla_dev_init,
        .pre_run        = odla_dev_prerun,
        .run            = odla_dev_run,
        .post_run       = odla_dev_postrun,
        .async_run      = nullptr,
        .async_wait     = nullptr,
        .release_graph  = nullptr,
        .release_device = odla_dev_release,
};
```

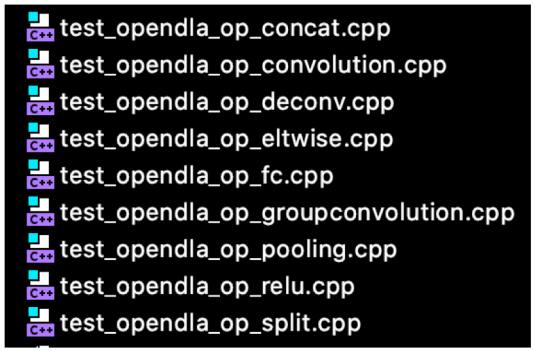
pre_run: subgraph解析、内存分配

run: 填充Tensor、运行

post_run: 内存释放

Link library

OP test



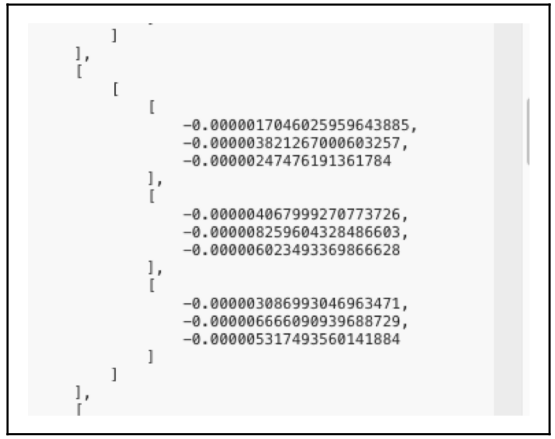

```
/* create OpenDLA backend */
context_t odla_context = create_context( context_name: "odla", empty_context: 1);
int rtt = set_context_device(odla_context, dev_name: "OPENDLA", dev_option: NULL, dev_opt_size: 0);
if (0 > rtt)
{
    fprintf(stderr, " add_context_device VSI DEVICE failed.\n");
    return NULL;
}

graph_t graph = create_graph(odla_context, model_format: NULL, file_name: NULL);
```

算子实现

我的调试/开发环境《讲给碰到BUG的小伙伴

设置环境变量 TG_ODLA_DEBUG_DATA=1，可以：

1. 在当前目录下创建一个output文件夹输出Loadable与每一层的量化过后的权重和Bias数据。
2. 输出NVDLA的每个子图的输入和输出数据
3. 输出每个子图的Loadable文件，这样仍然可以用NVDLA的那一套去做

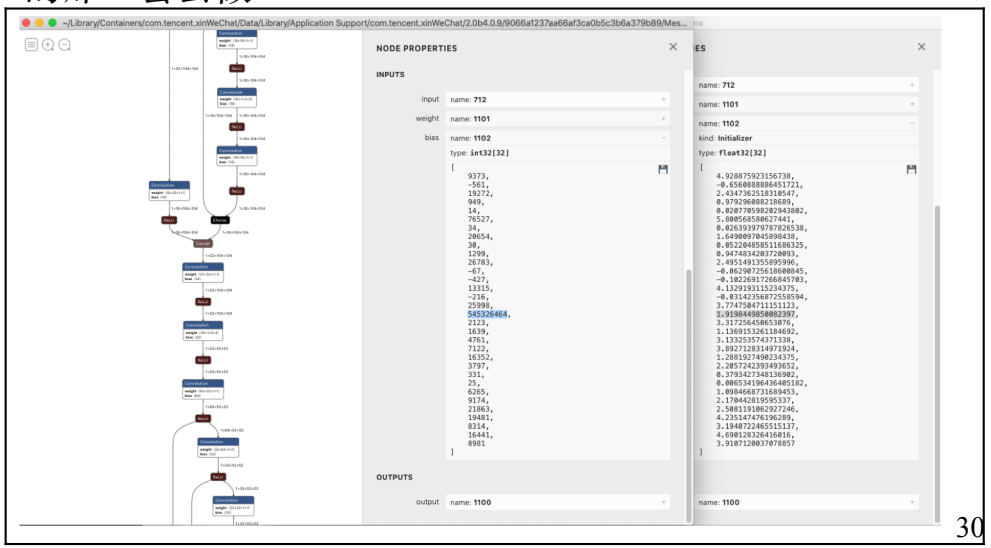


设置环境变量 TG_DEBUG_TIME=1，可以：

1. 打印出CPU运行每一层所消耗的时间
2. 打印出切图的情况

设置环境变量 TG_DEBUG_DATA=1，可以：

1. 保存CPU运算的每一个节点的输入输出数据



CLion Full Remote Mode:
https://www.jetbrains.com/help/clion/remote-projects-support.html

为什么不是TVM

How to extend NNVM/TVM as a static compiler for HW accelerator like NVDLA?

Questions

teabun Sep '18

I have an interest in using NNVM/TVM as a static compiler to generate a runtime image for HW accelerator like that of NVDLA.

NVDLA accelerates at a much higher level of abstraction than CPU and GPU, e.g. conv2d, batchborm, etc.

Graph level optimisation like ops fusion by NNVM will be very useful.

Can you advise how best I should approach, e.g. relevant topics in the documentation/tutorial etc.

Thanks.
–kahho

Sep 2018

1 / 11
Sep 2018

社区有过一些讨论，但是都没有下文。

Integrating NVDLA to VTA (Diary)

Development

redpanda3 Feb '20

Here is my branch

GitHub 49

redpanda3/incubator-tvm 49

Open deep learning compiler stack for cpu, gpu and specialized accelerators - redpanda3/incubator-tvm

1 Reply 1 Reply

Feb 2020

1 / 5
Feb 2020

Tengine简单易用方便社区开发
架构师加鸡腿！

欢迎加入Tengine开源社区