NVidia Deep Learning Accelerator(NVDLA)

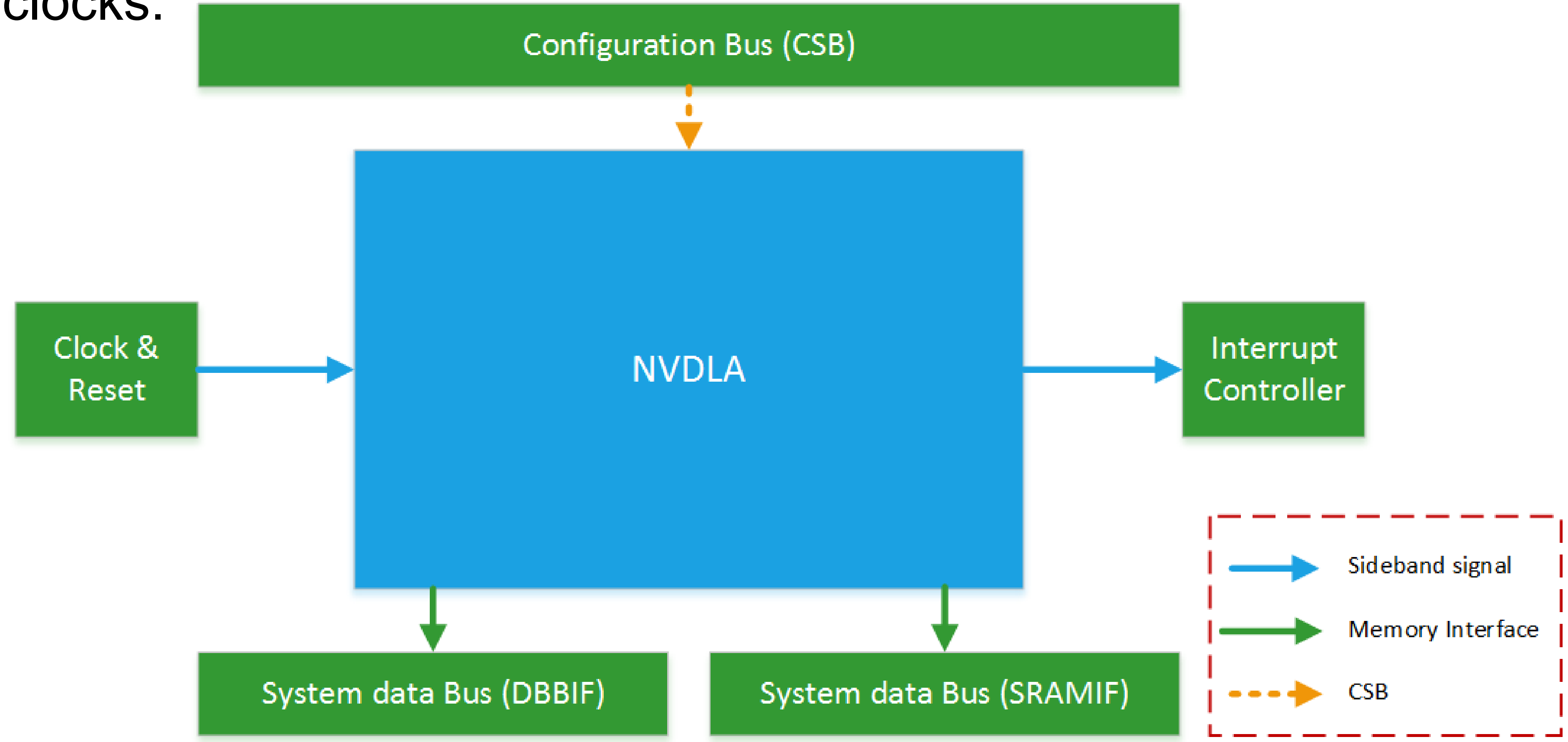
王逵读代码和文档的笔记（不保证正确）

简介

- 针对 IoT 市场，高度可配置： applications ranging from smaller, cost-sensitive Internet of Things (IoT) devices to larger performance oriented IoT devices
- 社区协作模式： NVIDIA welcomes contributions to NVDLA, and will maintain an open process for external users and developers who wish to submit changes back
- 免费： NVIDIA and each Contributor hereby grant to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable license under the NVDLA Patents to make, have made, use, offer to sell, sell, import
- NV 试图掌控生态： for users who build “NVDLA-compatible” implementations which interact well with the greater NVDLA ecosystem, NVIDIA may grant the right to use the NVIDIA trademarks.

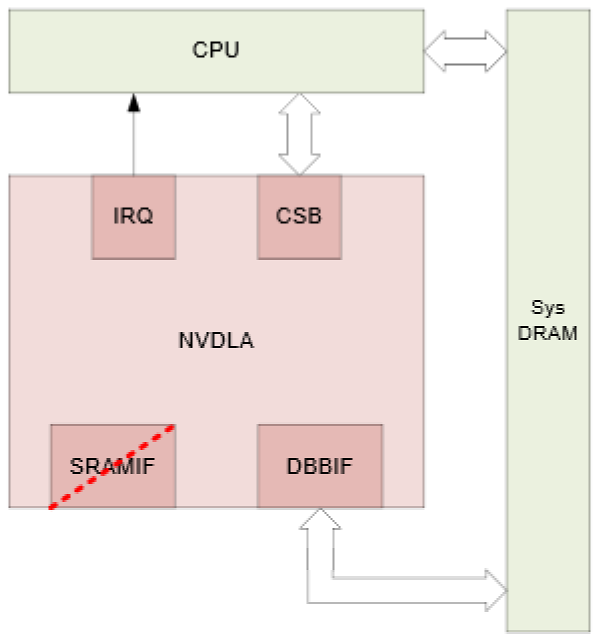
Fixed Function Accelerator (not CUDA!)

Use a MCU to programm it through configuration regiseters
The NVDLA core operates in a single clock domain; bus adapters allow for clock domain crossing from the internal NVDLA clock to the bus clocks.

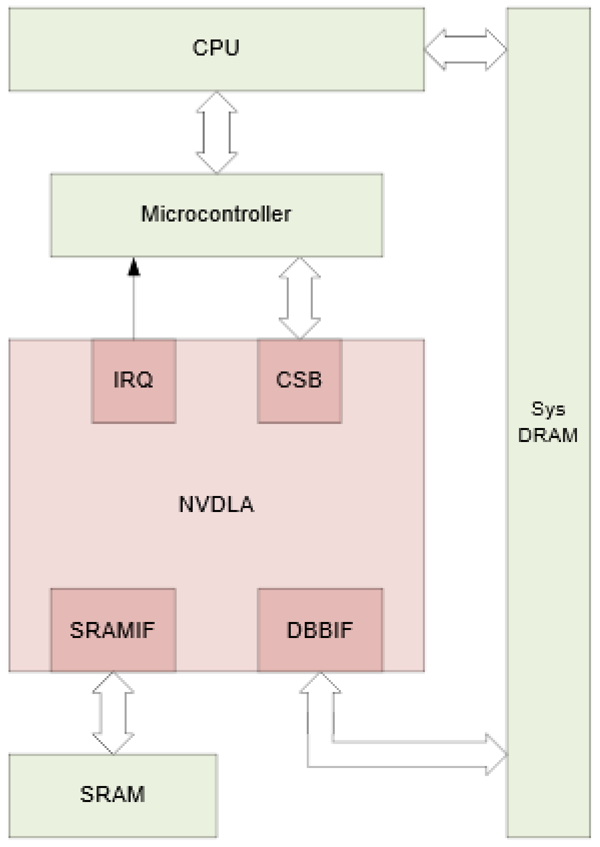


System Integeration

Headless – unit-by-unit management of the NVDLA hardware happens on the main system processor (which runs Linux). Usually used in a small system without an interface to large on-die SRAM.
Headed – delegates the high-interrupt-frequency tasks to a companion microcontroller that is tightly coupled to the NVDLA sub-system.



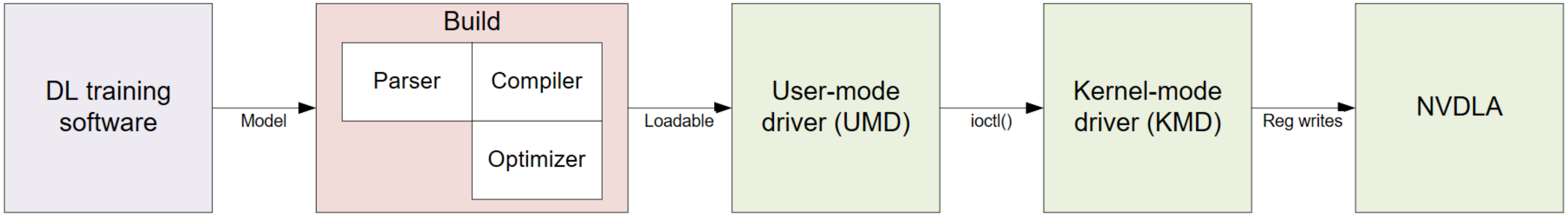
"Small" NVDLA system



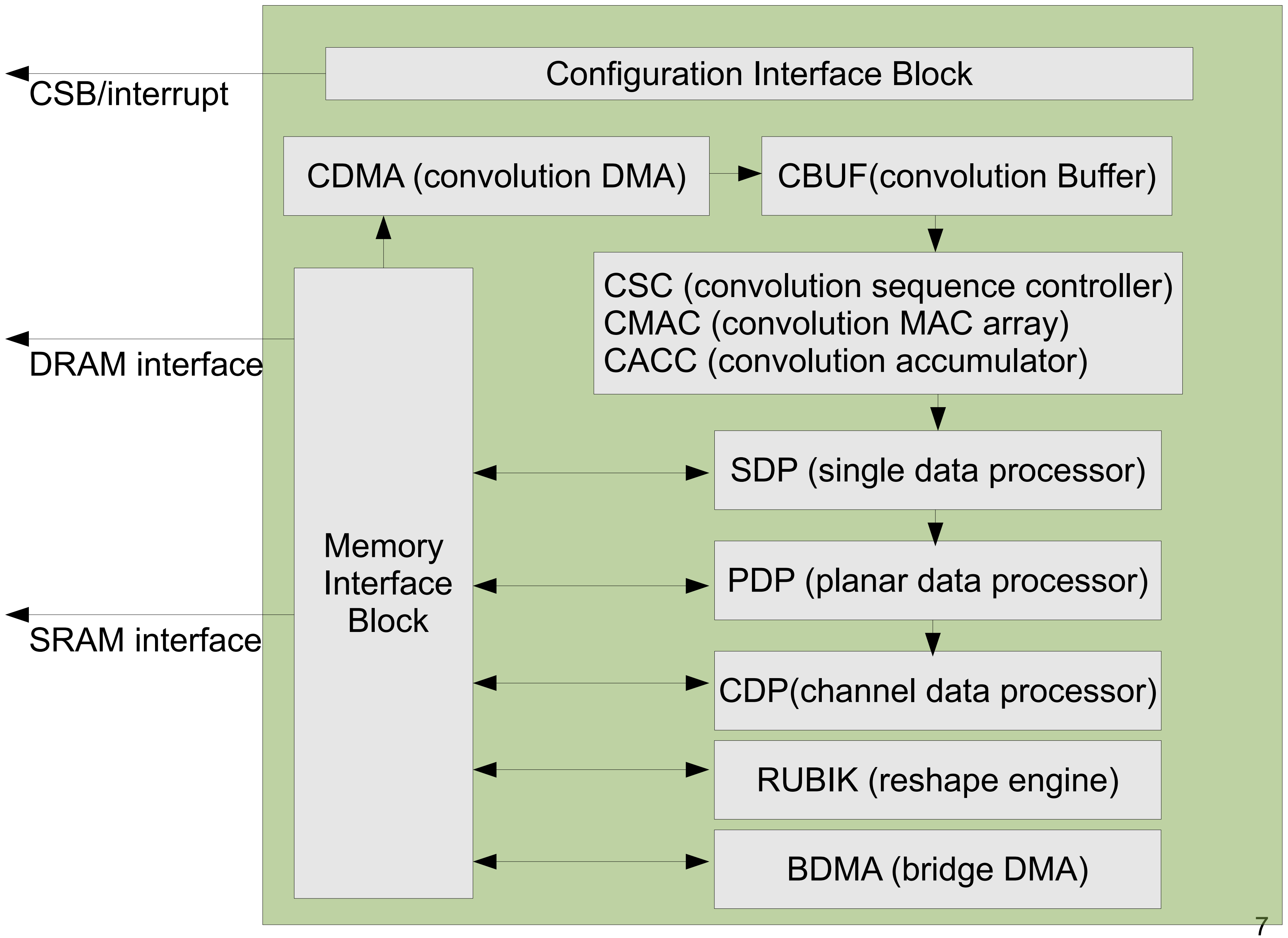
"Large" NVDLA system

软件系统

- runtime environment: an on-device software stack (part of the open source release)
- a full training infrastructure to build new models that incorporate Deep Learning
- compilation tools: parsers that convert existing models to a form that is usable by the on-device software. HW layer 和 SW layer 之间未必是一一对应的关系，可能是一对多或多对一



	2017Q3	2017Q4	2018H1
Document	Conceptual Overview Hardware Architecture Software Interface Integration Manual	Software Porting Guide	Software architecture document
Hardware	Large Config Verilog RTL, early access testbench and traces synthesis scripts Integration Manual	Large config Verilog RTL released Small config Verilog RTL, early access	Small config Verilog RTL released RTL supporting fine grained configuration UVM testbench for custom configurations
Software	Performance Estimator Tool	Linux Kernel Mode Driver source QEMU based C model Linux User Mode Driver Runtime source Binary compiler Application supporting Caffe and TensorFlow models most CNN related layers Software sanity tests	FPGA support for accelerating software development Customizable compiler NVDLA compliance test suite TensorRT and all supported frameworks


CSB/interrupt
DRAM interface
SRAM interface
Configuration Interface Block
CDMA (convolution DMA)
CBUF(convolution Buffer)
Memory Interface Block
CSC (convolution sequence controller)
CMAC (convolution MAC array)
CACC (convolution accumulator)
SDP (single data processor)
PDP (planar data processor)
CDP(channel data processor)
RUBIK (reshape engine)
BDMA (bridge DMA)

组成部件

•Convolution Core – optimized high-performance convolution engine. tf.nn.conv2d
•Single Data Processor – single-point lookup engine for activation functions. tf.nn.batch_normalization, tf.nn.bias_add, tf.nn.elu, tf.nn.relu, tf.sigmoid, tf.tanh
•Planar Data Processor – planar averaging engine for pooling. tf.nn.avg_pool, tf.nn.max_pool, and tf.nn.pool
•Channel Data Processor – multi-channel averaging engine for advanced normalization functions. tf.nn.local_response_normalization
•Dedicated Memory and Data Reshape Engines – memory-to-memory transformation acceleration for tensor reshape and copy operations. Supports deconvolution. tf.nn.conv2d_transpose, tf.concat, tf.slice, and tf.transpose.

高度可配置

- Many configurabel features.
- Each of these blocks are separate and independently configurable

Data types: Binary/INT4/INT8/INT16/INT32/FP16/FP32/FP64

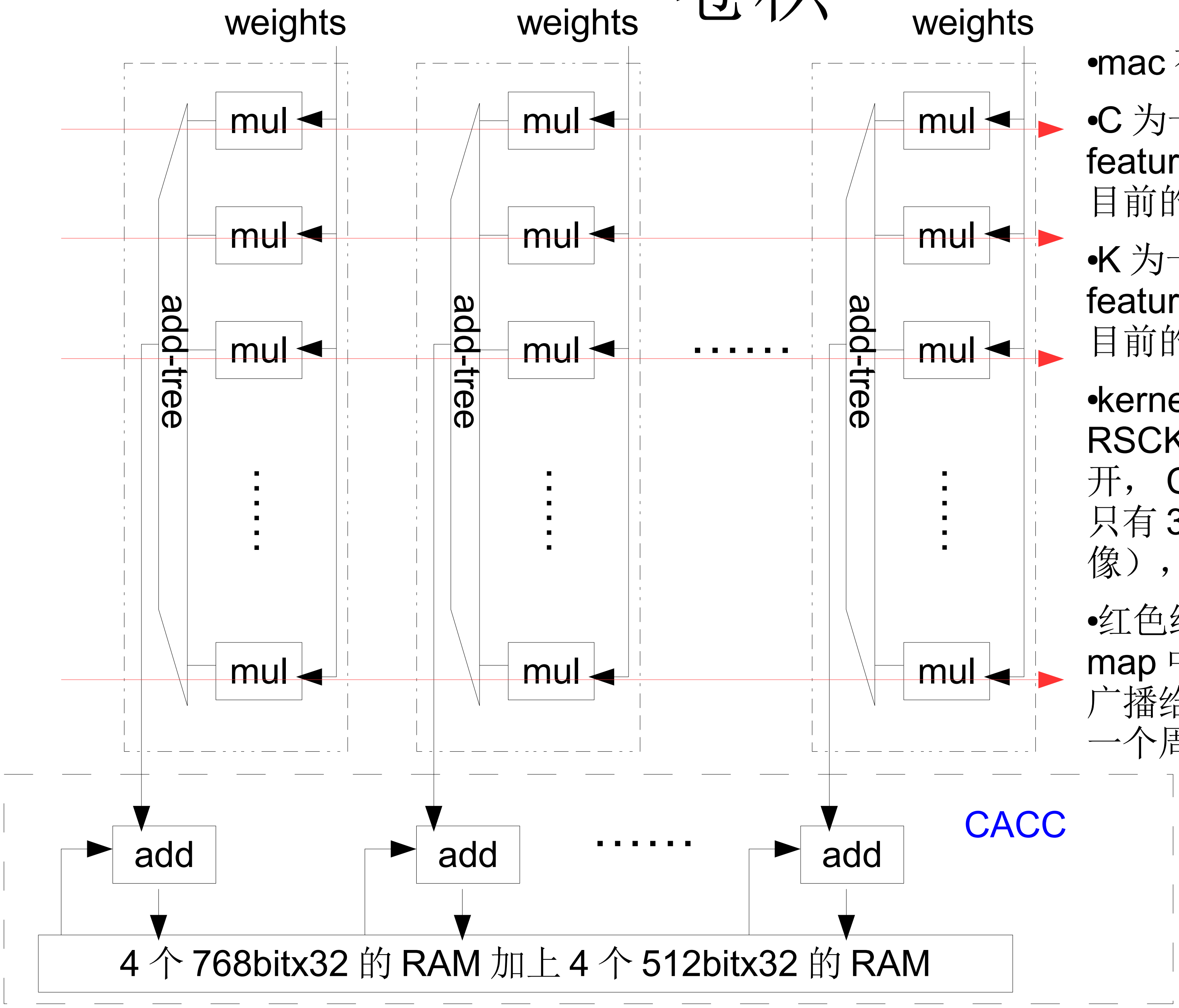
Input image memory formats
Weight compression
Winograd convolution
Batched convolution
Convolution buffer size
MAC array size
Second memory interface
Non-linear activation functions
Activation engine size
Bridge DMA engine
Data reshape engine
Pooling engine presence
Pooling engine size
Local response normalization engine presence
Local response normalization engine size
Memory interface bit width
Memory read latency tolerance

部件之间的协作关系

Independent. When operating independently, each functional block is configured for when and what it executes, with each block working on its assigned task (akin to independent layers in a Deep Learning framework). Independent operation begins and ends with the assigned block performing memory-to-memory operations, in and out of main system memory or dedicated SRAM memory.
Fused. Fused operation is similar to independent operation, however, some blocks can be assembled as a pipeline. This improves performance by bypassing the round trip through memory, instead having blocks communicate with each other through small FIFOs (i.e., the convolution core can pass data to the Single Data Point Processor, which can pass data to the Planar Data Processor, and in turn to the Cross-channel Data Processor).

The NVDLA hardware does NOT have intrinsic support for dependency tracking; that is to say, hardware layers that are running or pending do not have any mechanism of blocking each other, if one depends on the output of the other. As such, the CPU is responsible for ensuring that if a layer depends on the output of a previous layer, the consuming layer is not scheduled until the producing layer has finished executing.

卷积



- mac 有 C 行 K 列。
- C 为一次可处理的 input feature map 的 channel 数。目前的配置中 C=64 。
- K 为一次可处理的 output feature map 的 channel 数。目前的配置中 K=8 。
- kernel 的 4 个维度分别为 RSCK， RS 按时间展开， CK 按空间展开 , 如果 C 只有 3 （输入层的 RGB 图像）， RS 也在空间展开。
- 红色线表示的 input feature map 中 C 个 channel 的数据广播给 K 列。目前的实现中是一个周期广播给 8 列

卷积

- 以 INT16 为例， 1024 mac 每个周期需要 2048 个运算数据（ weight 和 feature data ） , 而实际上内部 buffer 能够提供的基本带宽是 256Byte/cycle ， weight 和 data 各 128byte ，相当于 64 个 weight element 和 64 个 feature element 。
- 基本思想是这样的：每次运算开始前，准备好 16 个 kernel ，每个 kernel 是 64 个 weight, 总共 1024 。然后进行一个 16 周期的运算，运算过程中保持着总共 1024 个 weight 不变，每个周期获取 64 个 feature data ，共享给 16 个 kernel ， add-tree 每个周期输出 16 个部分卷积结果，这 16 个周期中 feature data 源源不断取。这 16 个部分卷积结果将被累加到 CACC 的 RAM 中不同的 entry 上。
- 这 16 个周期中，也不断取 weight ，给下一轮运算做准备，
- 这种一轮运算叫做 stripe operation ， 一般是 16~32 周期，最大可以到 64 周期，极端情况下可以只有 1 个周期。

卷积模式

- 支持四种模式 Direct 、 Image-input 、 Winograd 、 Batching，彼此是互斥的
- If the NVDLA design specification has Atomic-C = 16 and Atomic-K = 64 (which would result in 1024 MAC instances), and one layer of the network has the input feature data channel number = 8 and output feature data kernel number = 16, then the MAC utilization will be only 1/8th (i.e., only 128 MACs will be utilized with the others being idle at all times).
- 所以，对于 Image-input 这种 C 特别少的，要有特殊处理，即把 RS 在空间展开（ folding ）
- Winograd 模式下， 64 个 K， 按 C 展开 4， 按 RS 展开 16， 算一个 Winograd 窗口中的点
- Batching 模式主要针对全连接层，最大到 32 。要注意 Support for large batching sizes means a large area cost for activation buffering. Maximum batching size is limited by the convolution buffer size, so the maximum batching number is a hardware limitation in the design specification.

卷积的一些细节

- add-tree 在 Direct 模式下做 64 个乘积的累加。
- Winograd 模式下做 POA

```
// MAC support Winograd post addition (POA).
// It's a 2-level matrix muliplication implemented by adders.
// Fomular of POA are:
//                    | pp_out_00, pp_out_01, pp_out_02, pp_out_03 |  | 1   0 |
// | 1,  1,  1,  0 |  | pp_out_04, pp_out_05, pp_out_06, pp_out_07 |  | 1,  1 |
// | 0,  1, -1, -1 |  | pp_out_08, pp_out_09, pp_out_10, pp_out_11 |  | 1, -1 |
//                    | pp_out_12, pp_out_13, pp_out_14, pp_out_15 |  | 0, -1 |
// MAC cell CSA tree level 0
// FP16 sign tag process
// 64(128) -> 16(32)
// MAC cell CSA tree level 1
// for DC: 16(32) -> 8(16)
// for WG: first step of 4x4(2*4x4) -> 4x2(2*4x2)
// MAC cell CSA tree level 2
// for DC: 8(16) -> 4(8)
// for WG: second step of 4x4(2*4x4) -> 4x2(2*4x2)
// MAC cell CSA tree level 3
// for DC: 4(8) -> 2(4)
// for WG: first step of 4x2(2*4x2) -> 2x2(2*2x2)
// MAC cell CSA tree level 4
// for DC: 2(4) -> 1(2)
// for WG: second step of 4x2(2*4x2) -> 2x2(2*2x2)
```

卷积的 CBUF

- CBUF 保存权重、 Feature ，如果用到了压缩，还有 Weight 的 Tag ，这三者都需要一读一写两个口
- 用多 Bank 来支持多个读写口。支持 2~32 个 bank ，每个 bank 容量为 4KB~32KB
- 权重和 Feature 都支持压缩
-
- CDMA 把 Input Image 写入 CBUF 时，会经过特殊的整理，以便支持 Image-input 模式下在空间展开 RS

Configuration Space Block

•非标准但非常简单的配置接口，可以轻易转接。 The CSB interface is intentionally extremely simple, and low-performance; as such, it should be simple to build an adapter between the CSB and any other system bus that may be supported on a platform.

• 用 Ping-pong 的机制隐藏配置的延迟。 NVDLA introduces the concept of ping-pong register programming for per-hardware-layer registers. For most NVDLA subunits, there are two groups of registers; when the subunit is executing using the configuration from the first register set, the CPU can program the second group in the background

一些有趣的功能

•CSC (convolution sequence controller) 里面，有 Weight Loader 和 Data Loader 两大模块，都是从 CBUF 中取数，然后往 CMAC 里面灌
•Data Loader 的主要代码是用 Mentor Catapult 这个 HLS 工具转的，里面有大量的浮点操作， data 的 Winograd 的预处理应该是在这里进行的。
•因为 Weight 在 Winograd 预处理时，只是从 3x3 膨胀到 4x4 ，还算可以接受，因此很可能没有在芯片上 on-the-fly 地做预处理

Weight Loader 中，有一些注释，表明 Weight 的压缩是在这里做的

```
// Decoder of compressed weight
//          data_mask              input_data     mac_sel
//                 |                    |                 |
//          sums_for_sel            register     register
//                 |                    |                 |
//                 --------------->  mux           register
//                                       |                |
//                                output_data  output_sel
// phase I: calculate sums for mux
// phase I: registers
// phase II: mux
// phase II: registers and assertion
// phase III: registers
```

一些有趣的功能

为低精度数据准备的 Precision Scaling. Control memory bandwidth throughout the full inference process; feature data can be scaled to its full range before chunking into lower precision and writing to memory . Scale key resources (e.g., MAC array) to support full range for best inference result (other linear functions may be applied). Revert input data before any of the non-linear functions (i.e., keep input data of non-linear functions as original data).