Xilinx ML-Suite

Rahul Nimaiyar
Ashish Sirasao

© Copyright 2018 Xilinx

FPGA for Deep Learning

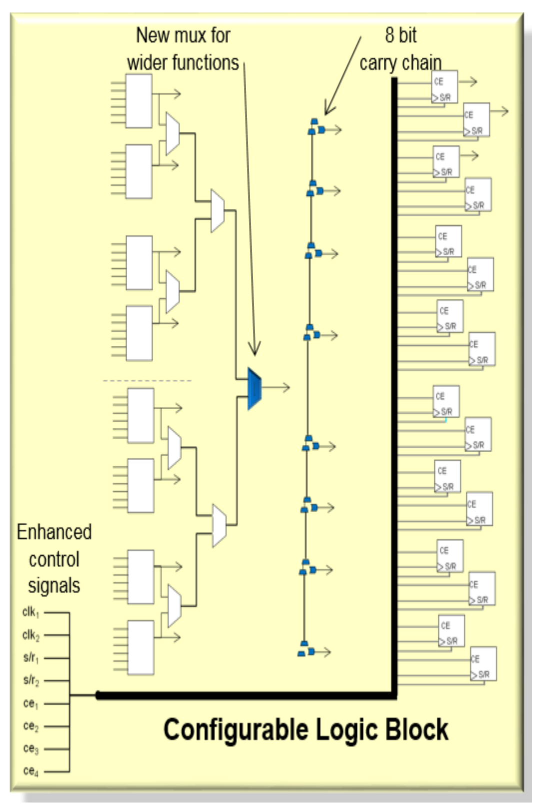
© Copyright 2018 Xilinx

Deep Learning on Xilinx Adaptable Devices

Data Parallel	• 2D Array of MACs • Flexible on-chip memory access • High Bandwidth, Multiple Access Ports
Custom Memory Hierarchy	• Near Memory Compute • Programmable routing for data & filter reuse
Compression & Sparsity	• Flexible Data Types • FP32/16, INT16/8/4/2, Binary/Ternary • Sparsity friendly compute
Broad Device Range	• Scalable device family for different applications • Built in System functions – Networking, Video, ARM

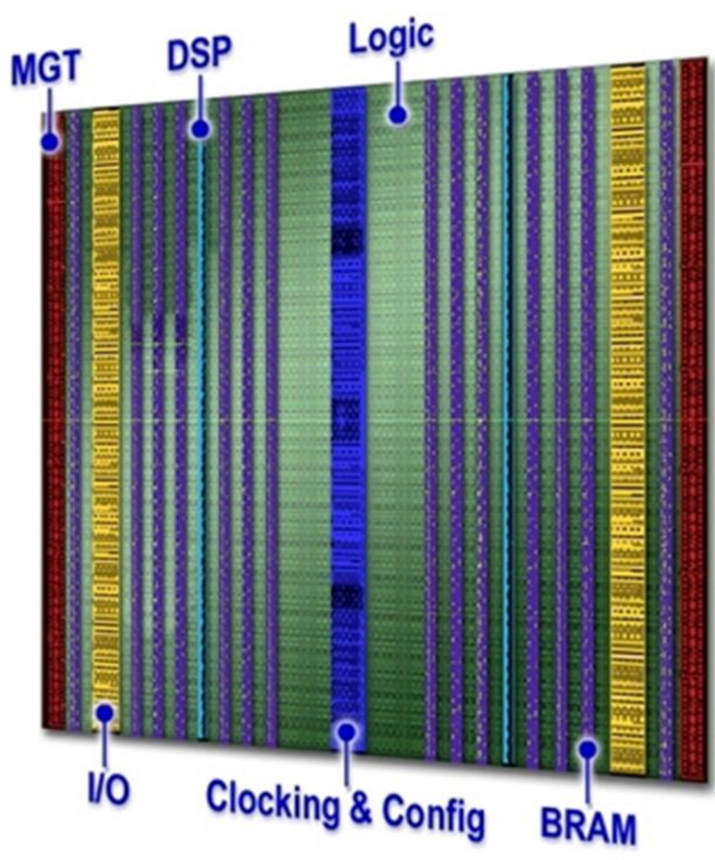
© Copyright 2018 Xilinx

Xilinx FPGA Building Blocks

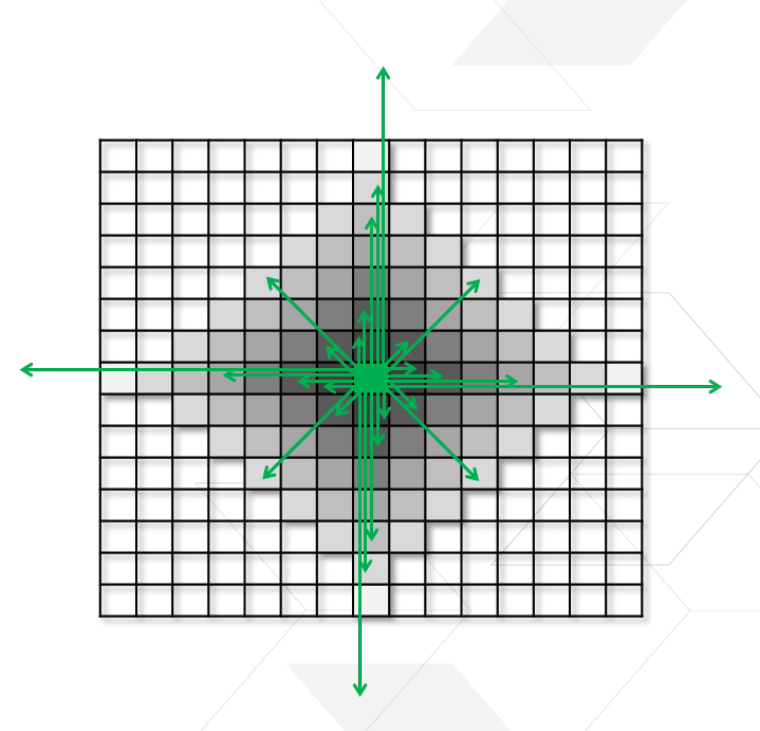


CLB Architecture

Eight six-input LUTs, Sixteen Flip Flops, Carry chain, Muxes for larger logic structures, Distributed memory, Shift register



Columnar Chip Layout



Programmable Routing:

Local routes spanning one, two, four or five tiles away

Long routes spanning twelve or sixteen tiles away

© Copyright 2018 Xilinx

DSP48E2 Slice In UltraScale

- 27x18-bit multiplier and 27-bit pre-adder
- Pre-adder MUX to select A or B input
- Squaring MUX on output of pre-adder
- Wide XOR (up to 96-bits)
- WMUX product feedback
- Cascading of Input and Outputs

Choose A or B as input to Pre-Adder

Squaring MUX

Wide XOR

B

A

D

C

Squaring MUX

Wide MUX

W

Wide XOR

XOR

48

ALU

27x18

X

+/-

27-bit Pre-Add

=

Pattern Detect

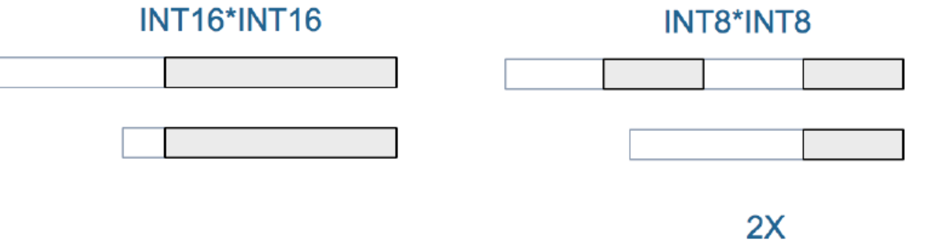
Output Registers

DSP48E2

27-bit pre-adder

27x18 Mult

W-MUX



INT4*INT4

INT2*INT2

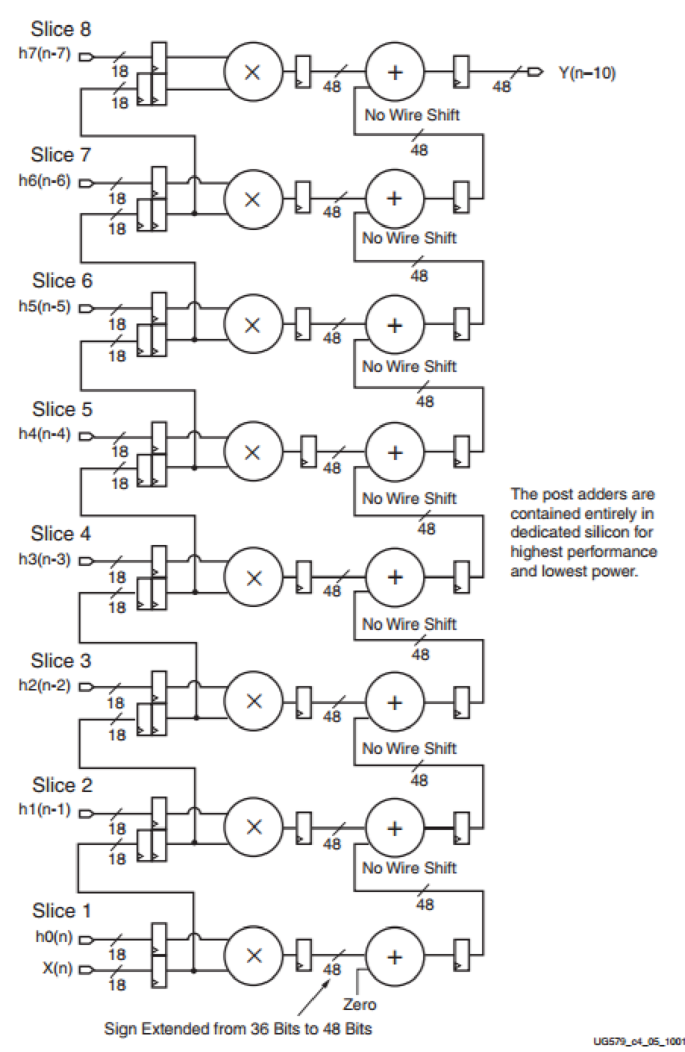
4X

8X

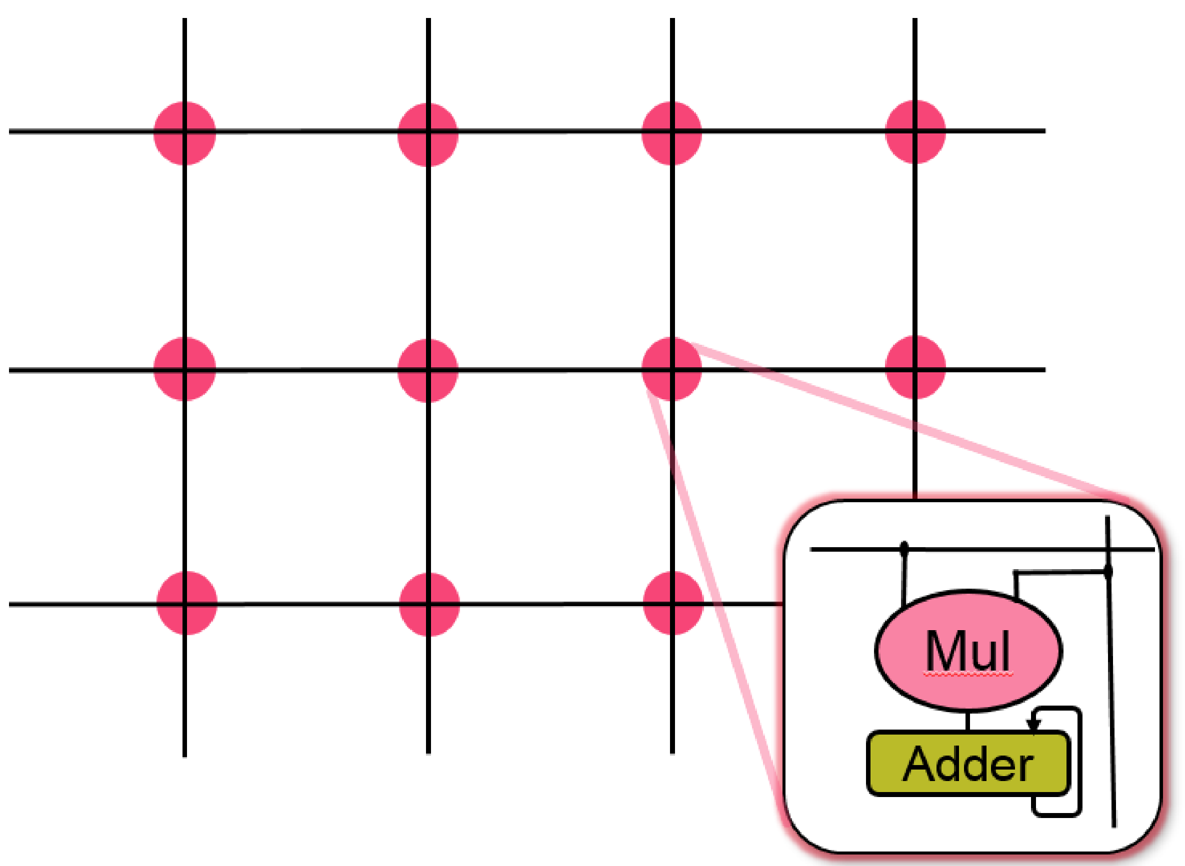
Xilinx White Paper: wp486, wp487

© Copyright 2018 Xilinx

DSP Cascade – Deep learning



Dot Product Engine – Using Output Cascade



Systolic Array of Accumulators– Using Input Cascade

© Copyright 2018 Xilinx

Variable Precision Compute Density – TOPs on VU9P

Weight/Activation	VU9P
MAX FLOAT (SP)/ FLOAT (SP)	2.18538
MAX FP16/FP16	5.64
MAX 8b/8b	17.48304
XFP8 (1,3,4)	23.60306
MAX 4b/4b - DSP	25.71035
XFP7 (1,3,3)	34.96608
MAX 4b/8b	41.72917
MAX 4b/4b - LUT	81.65384
MAX 2b/8b	92.10951
MAX T/8b	92.10951
MAX B/8b	92.10951
MAX B/4b	160.7017
MAX 2b/2b	314.7075
MAX B/B	686.6345

MAX TOPs Estimates at 700 MHz FMAX

© Copyright 2018 Xilinx

Virtex® UltraScale+™ Full Spectrum of Memory

Distributed RAM
(10s of megabits)

Block RAM
(10s of megabits)

UltraRAM
(100s of Megabits)

High Bandwidth Memory
(Multi-Gigabyte)

External Memory
2666-DDR4
(Multi-Gigabyte)

	Distributed RAM	Block RAM	UltraRAM	High Bandwidth Memory	External Memory
VU9P	36Mb *675Tb/s*	77Mb *216Tb/s*	270Mb *80Tb/s*	VU37P: 8GB, 460GB/s	64GB *85GB/s*

5 Tiers of Memory -> Build custom memory hierarchy.
500 Mb of On-chip Memory and Tb/s of On-chip Memory Bandwidth

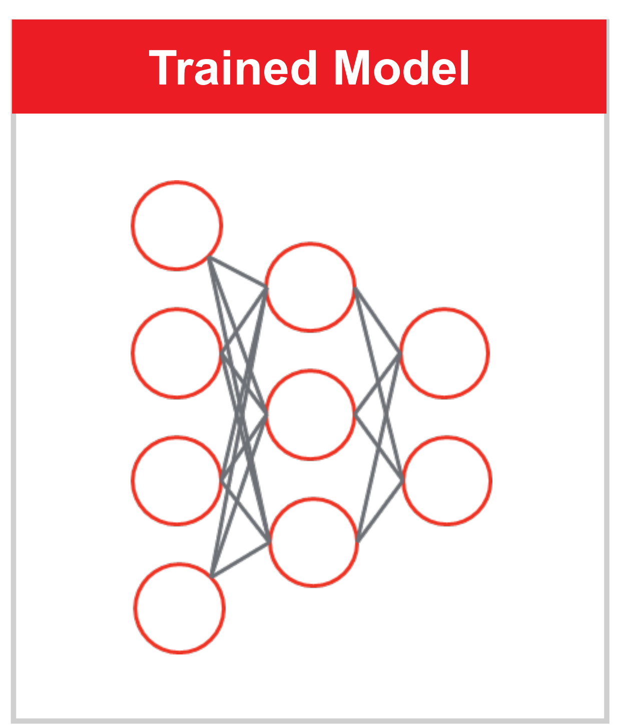
© Copyright 2018 Xilinx

XIinx ML-Suite – xDNN Inference Engine

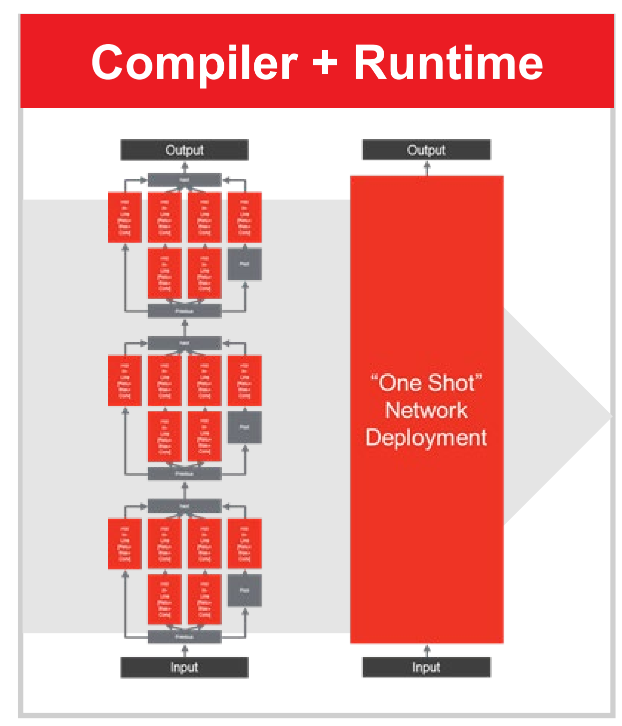
© Copyright 2018 Xilinx

Xilinx Inference Engine – DNN Processor + Compiler

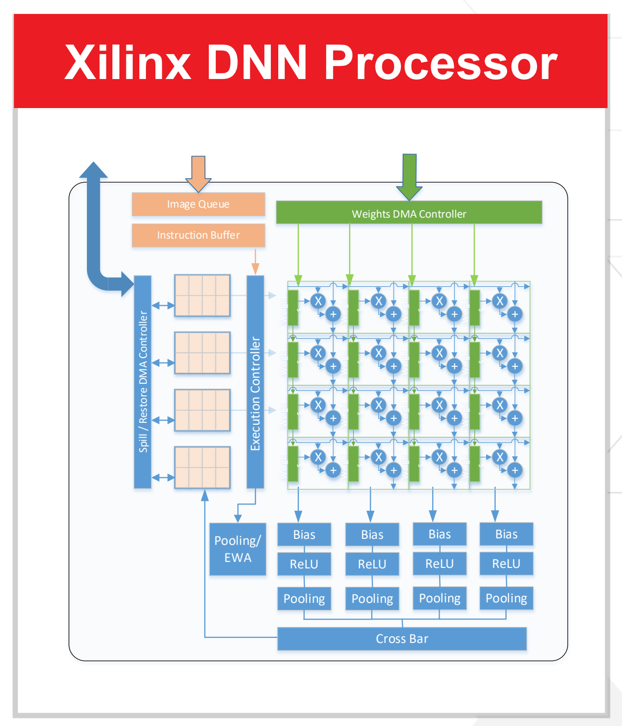
Trained Model



Compiler + Runtime



Xilinx DNN Processor



- Low Latency AND High Throughput at Batch =1
- No FPGA Expertise Needed

© Copyright 2018 Xilinx

Xilinx DNN Processor (xDNN)

Image Queue

Instruction Buffer

Weights DMA Controller

Spill / Restore DMA Controller

Execution Controller

Systolic Array

Bias | ReLU | Pooling (×4)

Pooling/ EWA

Cross Bar

- Low latency CNN Inference engine for Classification, Detection and Segmentation
- Overlay architecture – No need to reprogram FPGA
- Programmable Feature-set
- Tensor Level Instructions
- 700+MHz DSP Freq (VU9P)
- Custom Network Acceleration

© Copyright 2018 Xilinx

xDNN – Channel Parallel Systolic Array

Image Queue

Instruction Buffer

Weights DMA Controller

Spill / Restore DMA Controller

Execution Controller

2D Channel Parallel Datapath & Systolic Array Distributed Weight Buffers

Bias | ReLU | Pooling (×4)

Pooling/ EWA

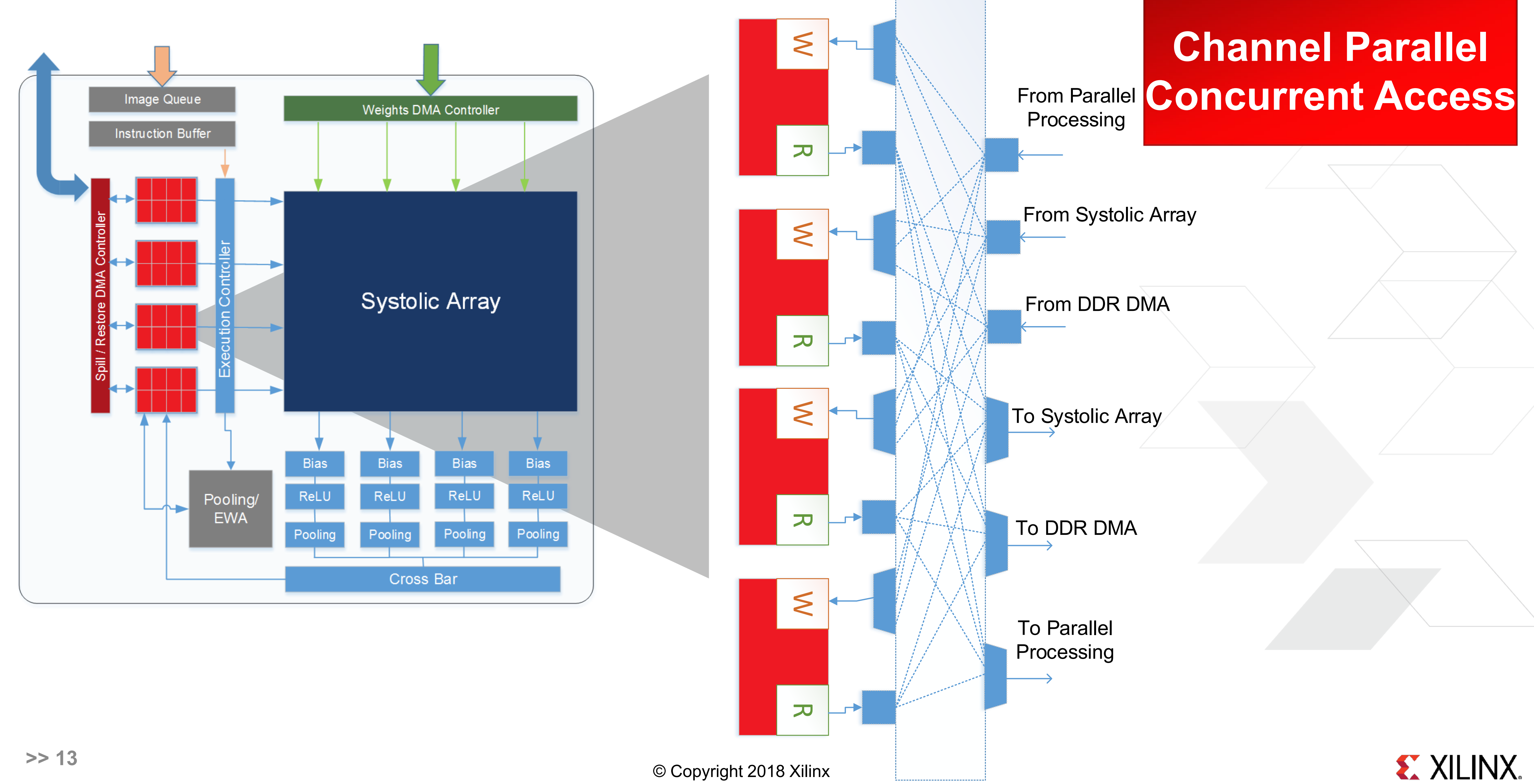
Cross Bar

Weights · X · + · Σ

Micro-Architecture Optimized for underlying Ultrascale+ FPGA Fabric

© Copyright 2018 Xilinx

xDNN Processor – Tensor Memory



© Copyright 2018 Xilinx

XILINX

Fusing of Operations

Conv
Batch Norm.
ReLU
Bias
Max Pool

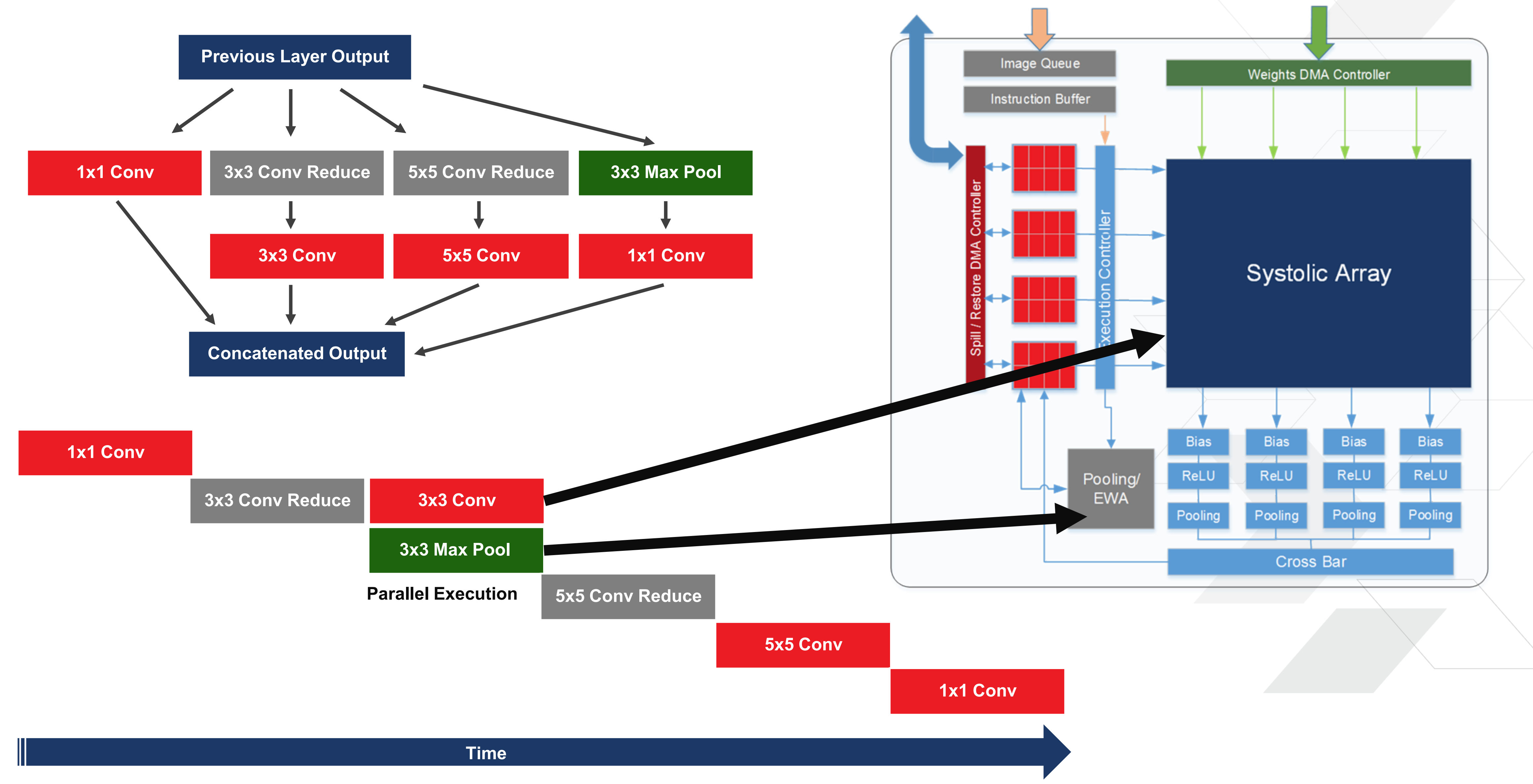
Conv.

Image Queue
Instruction Buffer
Weights DMA Controller
Spill / Restore DMA Controller
Execution Controller
Systolic Array
Bias
Pooling/ EWA
Pipelined Operators
(In-line, no RAM access)
Cross Bar

> Fuse operations as pipe-lined stages
> Data Flow – No memory access

© Copyright 2018 Xilinx

Instruction Level Parallelism

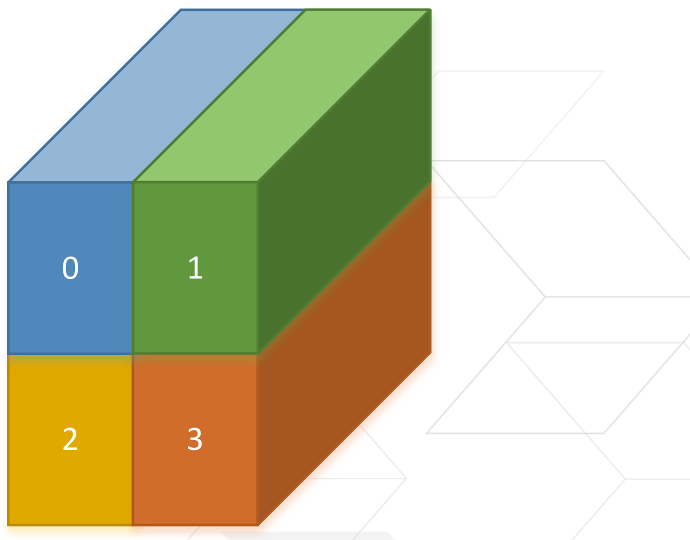


© Copyright 2018 Xilinx

XILINX

Automatic Intra Layer Tiling

- **Very important for High Resolution images**
- **Only when feature map size exceeds on-chip memory size**
- **Split each convolution**
  - Break IFM planes to tiles
  - Work on full feature map depth
  - Tiles fetched from external memory
  - Memory re-allocated so tiles overwrite previous tiles



© Copyright 2018 Xilinx

xDNN Feature support

Feature	Details
Convolution	NxM, Stride 1,2,4,8 N,M=1-15
Max Pool	NxM, Stride 1,2,4,8 N,M=1-15
Avg Pool	NxM, Stride 1,2,4,8 N,M=1-15
Dilated Convolution	Factor 1,2,4
De-Convolution	NxM, Stride 1,2,4,8 N,M=1-15
Up-sampling	Factor 2,4,8,16
Activation	ReLU, pReLU
Elementwise Addition	Any – Square, Rectangular
Precision	Int8, Int16
Network	Classification: e.g. ResNet Object Detection: e.g. YOLO v2, Segmentation: e.g. MaskRCNN

© Copyright 2018 Xilinx

Xilinx ML-Suite - xfDNN Software Stack

© Copyright 2018 Xilinx

https://github.com/Xilinx/ml-suite

Xilinx ML Suite

Branch: master ▾ | New pull request | Find file | Clone or download ▾

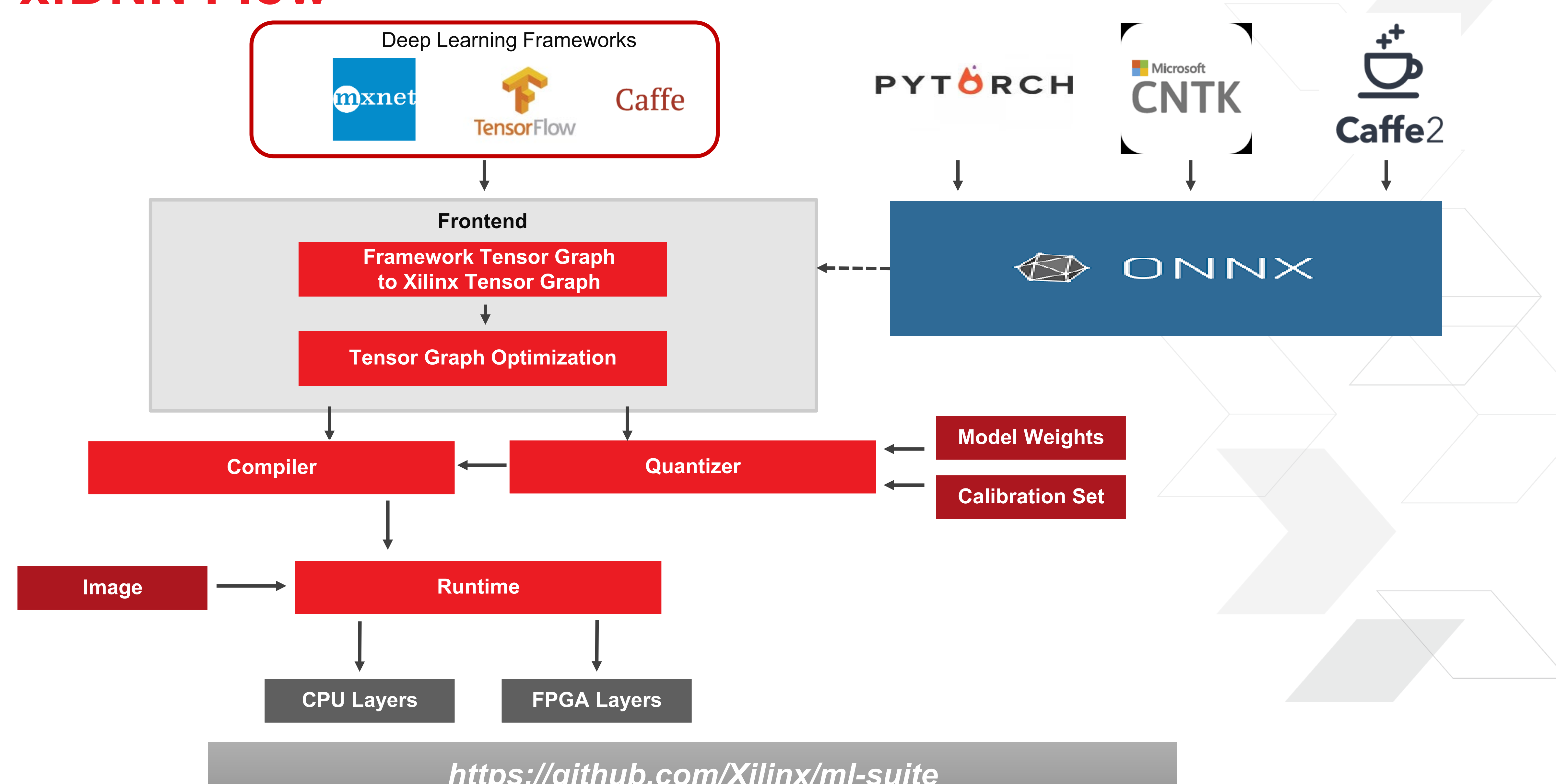
wilderfield Merge pull request #64 from Xilinx/v1.3 … Latest commit 5434529 2 days ago

apps	Adding darknet tools	4 days ago
docs	Staging 1.3	10 days ago
examples	Staging 1.3	10 days ago
ext	Staging 1.3	10 days ago
notebooks	Updating developer lab	4 days ago
overlaybins	Add XDNNv3 build for AWS	3 days ago
tests	Staging 1.3	10 days ago
xfdnn	Update xfdnn libraries	2 days ago
.gitignore	Update .gitignore	5 months ago
LICENSE	Release 1.1 to master	5 months ago
README.md	Updated docs	8 days ago
fix_caffe_opencv_symlink.sh	Staging 1.2	2 months ago

*Under Development

© Copyright 2018 Xilinx

xfDNN Flow



© Copyright 2018 Xilinx

XILINX

xfDNN Compiler

- Caffe - `xfdnn_compiler_caffe.pyc`
- Keras - `xfdnn_compiler_keras.pyc`
- MxNet - `xfdnn_compiler_mxnet.pyc`
- Tensforflow - `xfdnn_compiler_tensorflow.pyc`

Each of these tools contain largely the same arguments. For example, `xfdnn_compiler_caffe` has the following arguments:

```
usage: xfdnn_compiler_caffe.py [-h] [-n NETWORKFILE] [-g GENERATEFILE]
                               [-w WEIGHTS] [-o PNGFILE] [-c CONCATSTRATEGY]
                               [-s STRATEGY] [--schedulefile SCHEDULEFILE]
                               [-i DSP] [-v] [-m MEMORY] [-d DDR] [-p PHASE]
                               [-r RANKDIR]
```

- `-h, --help` - shows this help message and exit
- `-n, --networkfile` - NETWORKFILE Input prototxt for compiler
- `-g, --generatefile` - GENERATEFILE Output of Compiler, xDNN instructions
- `-w, --weights` - WEIGHTS Output of weights for use with deployment APIs
- `-o , --pngfile` - PNGFILE Outputs Optimized Graph as a PNG file, Requires dot executable
- `-c, --concatstrategy` - CONCATSTRATEGY
- `-s, --strategy` - STRATEGY Strategies for compiler (default: all)
- `--schedulefile` - SCHEDULEFILE Show layer by layer memory layout (optional)
- `-i, --dsp` DSP xdnn kernel dimension (28 or 56, default: 28) 28 is also known as "Med" and 56 as "Large"
- `-v, --verbose`
- `-m, --memory MEMORY` - On-chip Memory available in MB (default: 4). Needs to match intended overlaybin.
- `-d, --ddr DDR` - Users DDR memory along with on-chip memory. If the network is too large for the device, set this to `256` .
- `-p , --phase PHASE` - Caffe prototxt phase (TRAIN, TEST, ALL default: TEST)
- `-r, --rankdir RANKDIR`

© Copyright 2018 Xilinx

xfDNN Quantizer

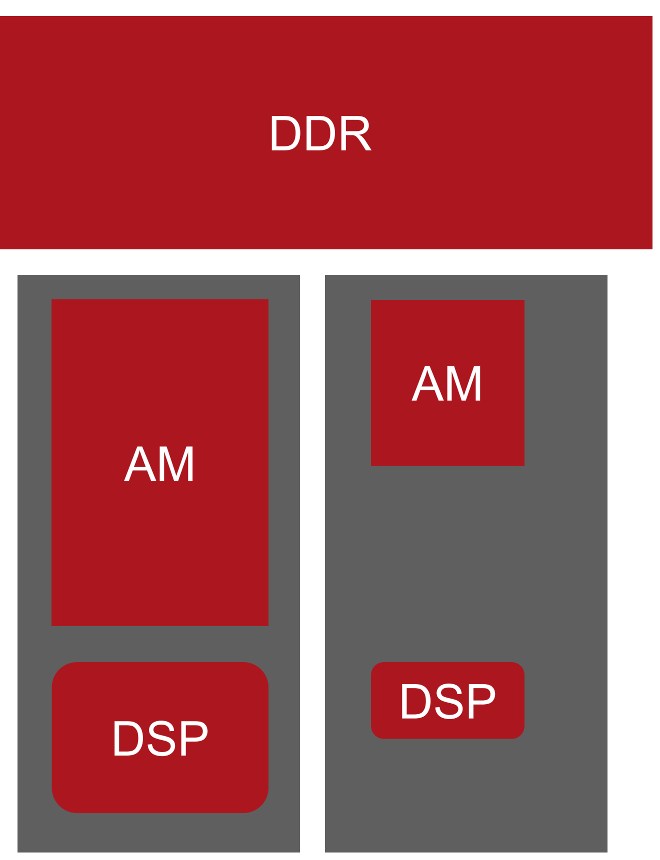
```
quantize.pyc [-h] [--deploy_model DEPLOY_MODEL]
                    [--output_json OUTPUT_JSON] [--weights WEIGHTS]
                    [--calibration_directory CALIBRATION_DIRECTORY]
                    [--calibration_size CALIBRATION_SIZE]
                    [--calibration_seed CALIBRATION_SEED]
                    [--calibration_indices CALIBRATION_INDICES]
                    [--bitwidths BITWIDTHS] [--dims DIMS]
                    [--transpose TRANSPOSE] [--channel_swap CHANNEL_SWAP]
                    [--raw_scale RAW_SCALE] [--mean_value MEAN_VALUE]
                    [--input_scale INPUT_SCALE]
```


© Copyright 2018 Xilinx

xfDNN Compiler

© Copyright 2018 Xilinx

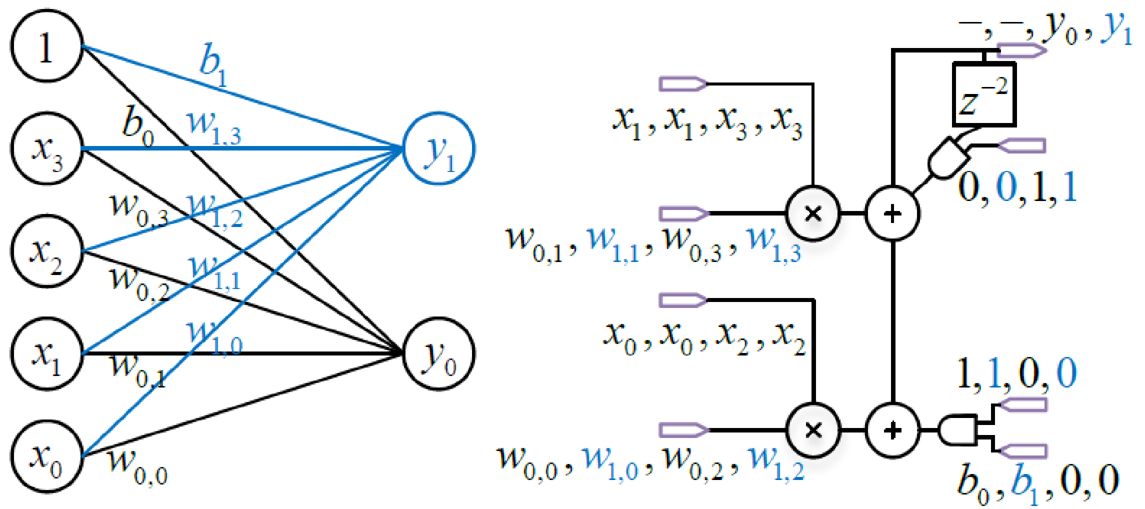
xDNN Logical View



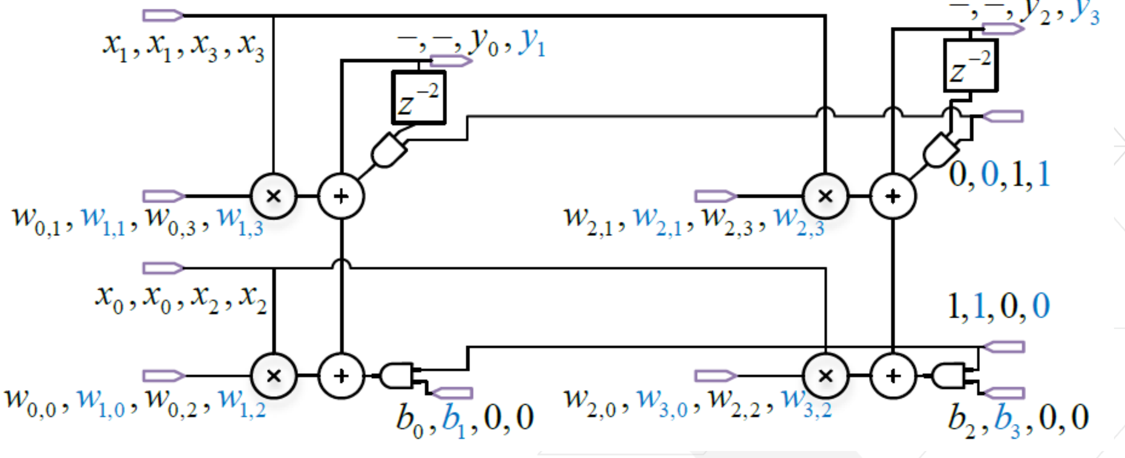
- **All instructions are high level instructions**
  - CONV, POOL, ELEMWISE, ReLU, Shift, Scale
  - No need to break down to lower level instructions
- **DSP array operates on Activation Memory**
  - Fast – On Chip SRAM
  - Slow – DDR
  - No need of explicit data movement during xDNN instructions(V3)
- **Weights are streamed automatically by xDNN – Formatted by software**
- **First Activation layer requires software formatting**

© Copyright 2018 Xilinx

DSP - Time Sliced Dot Product Computation



Single Column

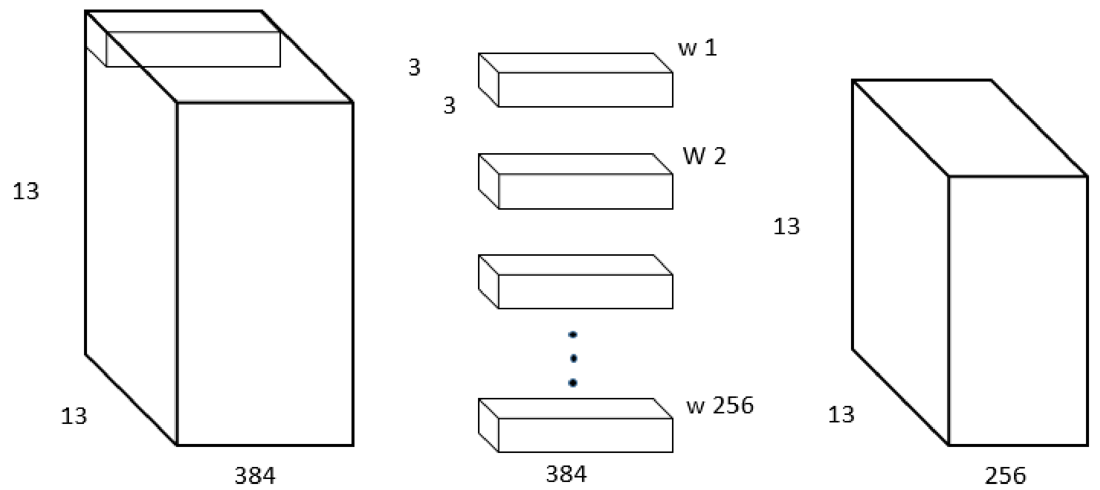


Multiple Column

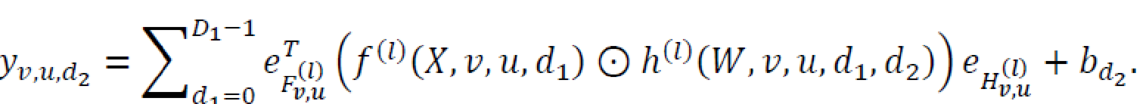
xDNN 96x16 array computes 4 outputs per column

© Copyright 2018 Xilinx

Convolution Operation on xDNN Systolic Array



	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7	Bank 8	Bank 9	Bank 10	Bank 11
	Ch 192 Ch 199	Ch 200 Ch 207	Ch 208 Ch 215	Ch 216 Ch 223	Ch 224 Ch 231	Ch 232 Ch 239	Ch 240 Ch 247	Ch 248 Ch 255				
	Ch 96 Ch 103	Ch 104 Ch 111	Ch 112 Ch 119	Ch 120 Ch 127	Ch 128 Ch 135	Ch 136 Ch 143	Ch 144 Ch 151	Ch 152 Ch 159	Ch 160 Ch 167	Ch 168 Ch 175	Ch 176 Ch 183	Ch 184 Ch 191
	Ch 0 Ch 7	Ch 8 Ch 15	Ch 16 Ch 23	Ch 24 Ch 31	Ch 32 Ch 39	Ch 40 Ch 47	Ch 48 Ch 55	Ch 56 Ch 63	Ch 64 Ch 71	Ch 72 Ch 79	Ch 80 Ch 87	Ch 88 Ch 95



$$y_{v,u,d_2} = \sum_{d_1=0}^{D_1-1} e^T_{F^{(l)}_{v,u}} \left(f^{(l)}(X,v,u,d_1) \odot h^{(l)}(W,v,u,d_1,d_2) \right) e_{H^{(l)}_{v,u}} + b_{d_2}.$$

Real Activation Memory Layout of 256 Channels with kernel size hxw

© Copyright 2018 Xilinx

Compiler – Scheduling for Optimized Memory Allocation

Previous Layer Output

1x1 Conv

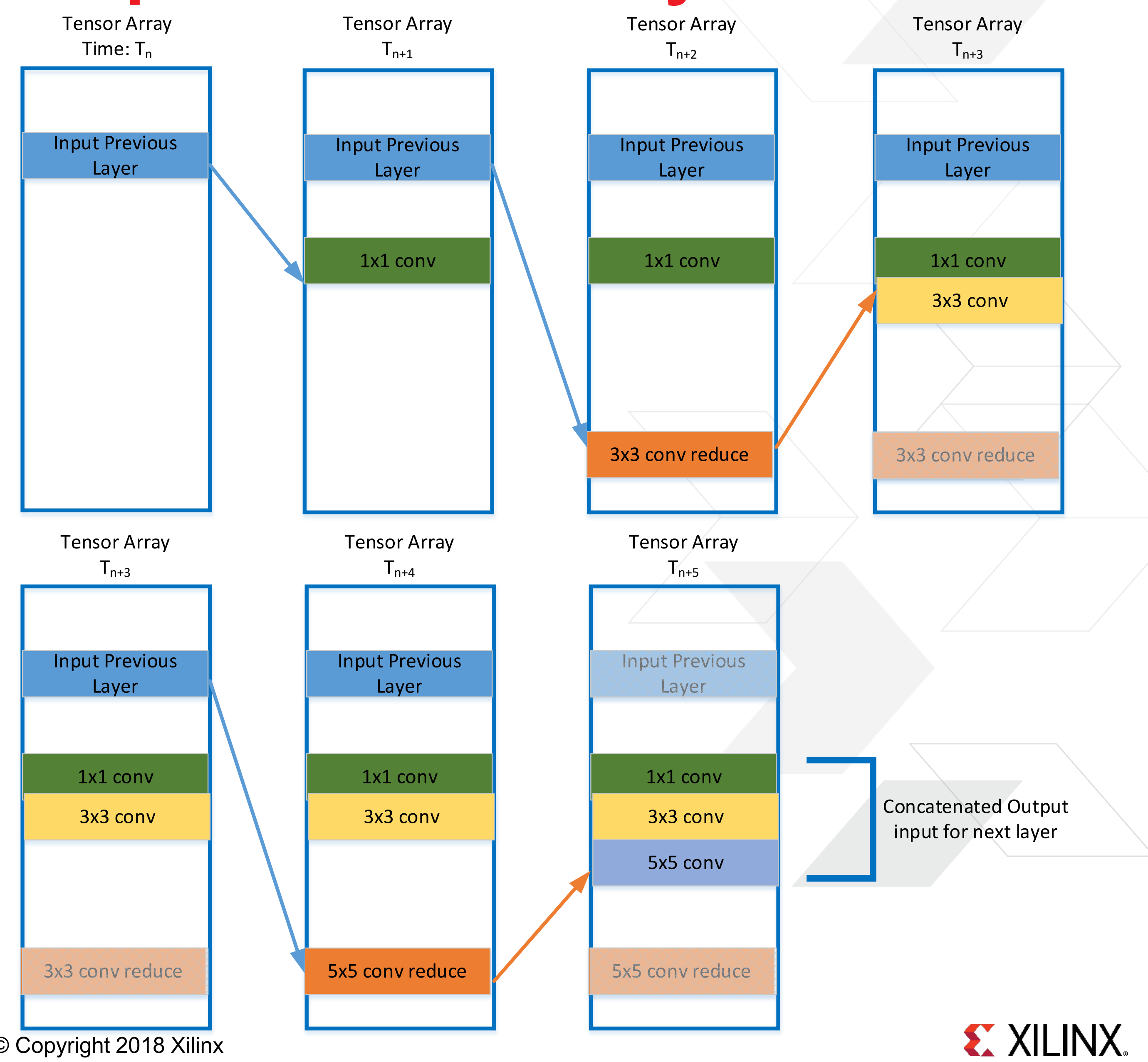
3x3 Conv Reduce

5x5 Conv Reduce

3x3 Conv

5x5 Conv

Concatenated Output



- **Address allocation - output of layer n -> input for layer n+1**
- **Maximize on-chip memory, minimize spill to external memory**
- **Concatenation through the address generation**

© Copyright 2018 Xilinx

Compiler – Optimize for ILP and Pipelining

Previous Layer Output

1x1 Conv
3x3 Conv Reduce
5x5 Conv Reduce
3x3 Max Pool

3x3 Conv
5x5 Conv
1x1 Conv

Concatenated Output

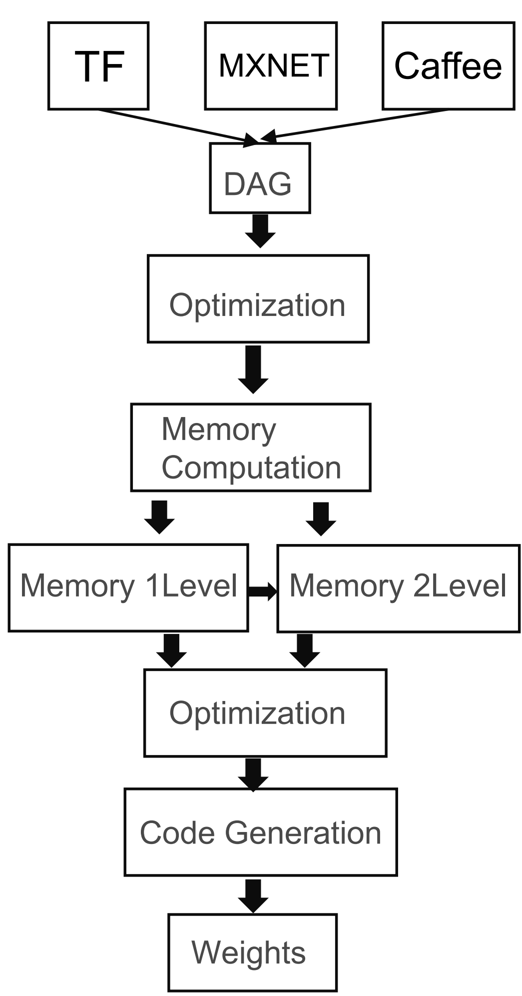
1x1 Conv
3x3 Conv Reduce
3x3 Conv
3x3 Max Pool
Parallel Execution
5x5 Conv Reduce
5x5 Conv
1x1 Conv

Time

Image Queue
Instruction Buffer
Weights DMA Controller
Spill / Restore DMA Controller
Execution Controller
Systolic Array
Pooling/ EWA
Bias
ReLU
Pooling
Cross Bar

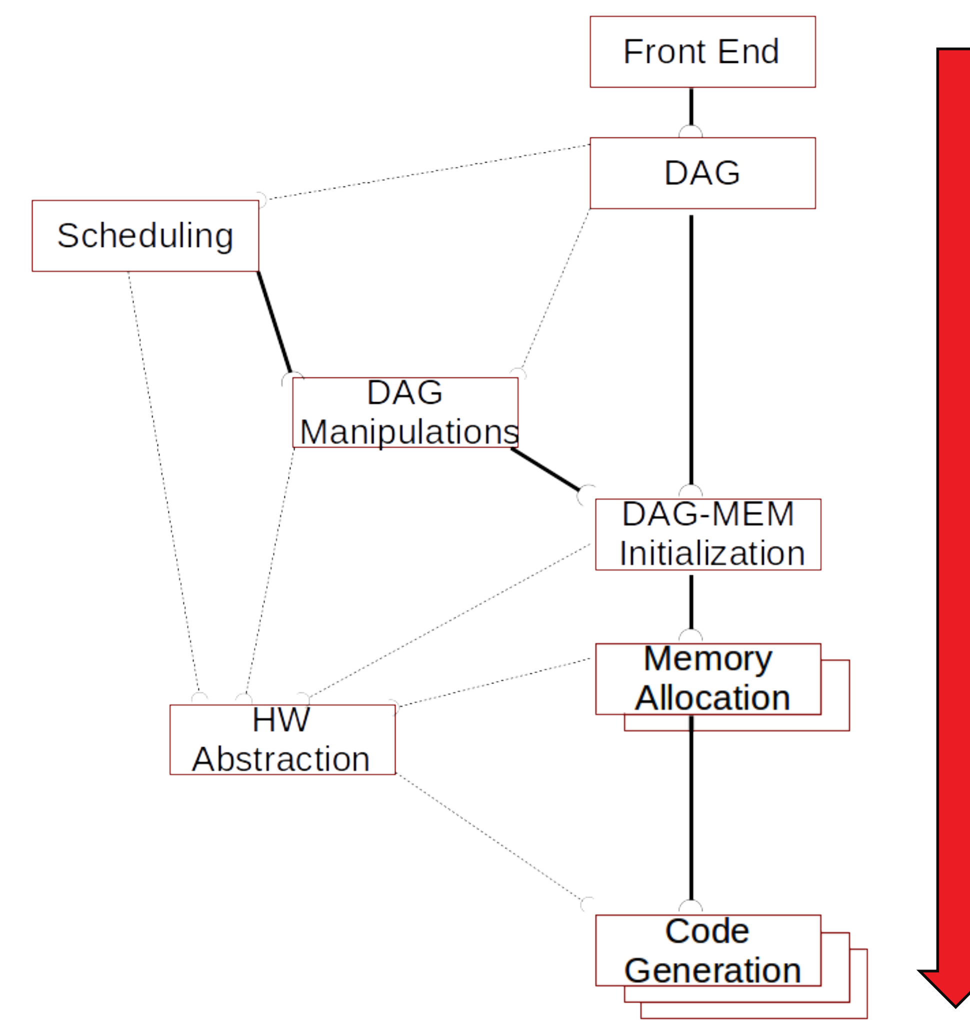
© Copyright 2018 Xilinx

xfDNN Compiler Features



- Support for multiple frameworks
  - MXNET, TF, Caffe
- Weight extraction and formatting
- DAG optimization and Operator Fusing
  - Conv = Conv + BN + Scale
  - Relu Merging
  - BatchNorm into Scale
  - conv_1x1_s2 = conv_1x1 + maxpool
- Slicing support for Conv, Elemwise, Concat
- Support for Scatter and Gather for URAM to DDR data movement and back
- Memory Computation
  - Minimum memory and graph initialization
- Memory allocation (Single and 2level)
  - URAM – User selectable LRU Cache
  - DDR – 256MB
- Code generation
  - Data movement + Split Code

© Copyright 2018 Xilinx



The Compiler Organization

Front-end → Code generation (Obviously)

There is interdependency among modules

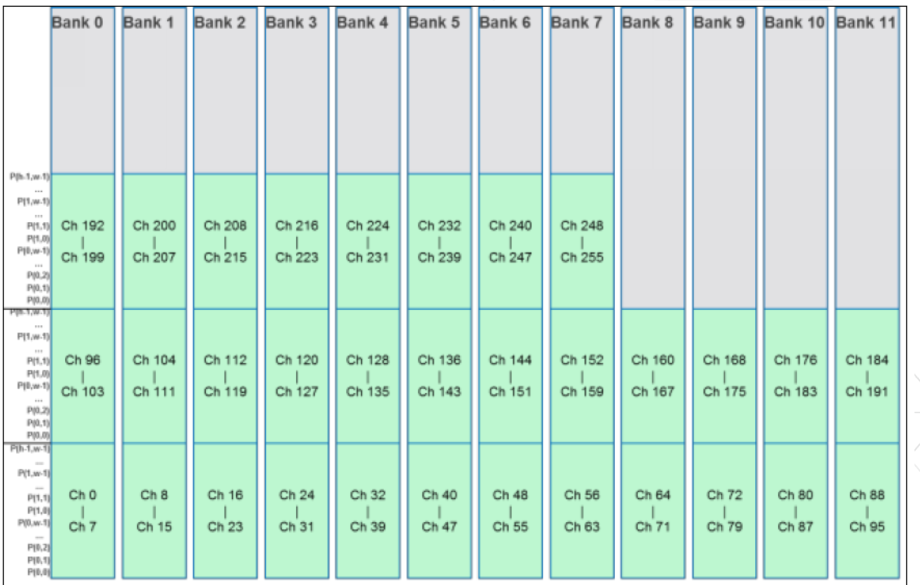
- Schedules ask for tensors space
- So do DAG manipulations
- Only after Memory allocation
  - we know Tensors space
- At Code generation
  - AM compute or DDR
- Key Optimizations
  - Replication
  - Pooling Around
  - Tiling
  - Parallel Schedule

© Copyright 2018 Xilinx

Replication – Memory Centric and Rule Based

- **What is Replication?**
  - The input tensor of a Convolution must satisfy alignment requirements
  - *R = Channels % DSP*
    - *R* is extra space
  - We can use R to replicate the input tensor in order to exploit parallelism
    - We can double down on R if necessary
- **Replication is a manipulation of Tensor layout (space)**
- **Replication improves the throughput of the convolution unit (time)**
  - If the convolution is not 1x1
- **Replication is an example of space-time**

Note: Replication is a tensor property. It affects its shapes and its memory space.



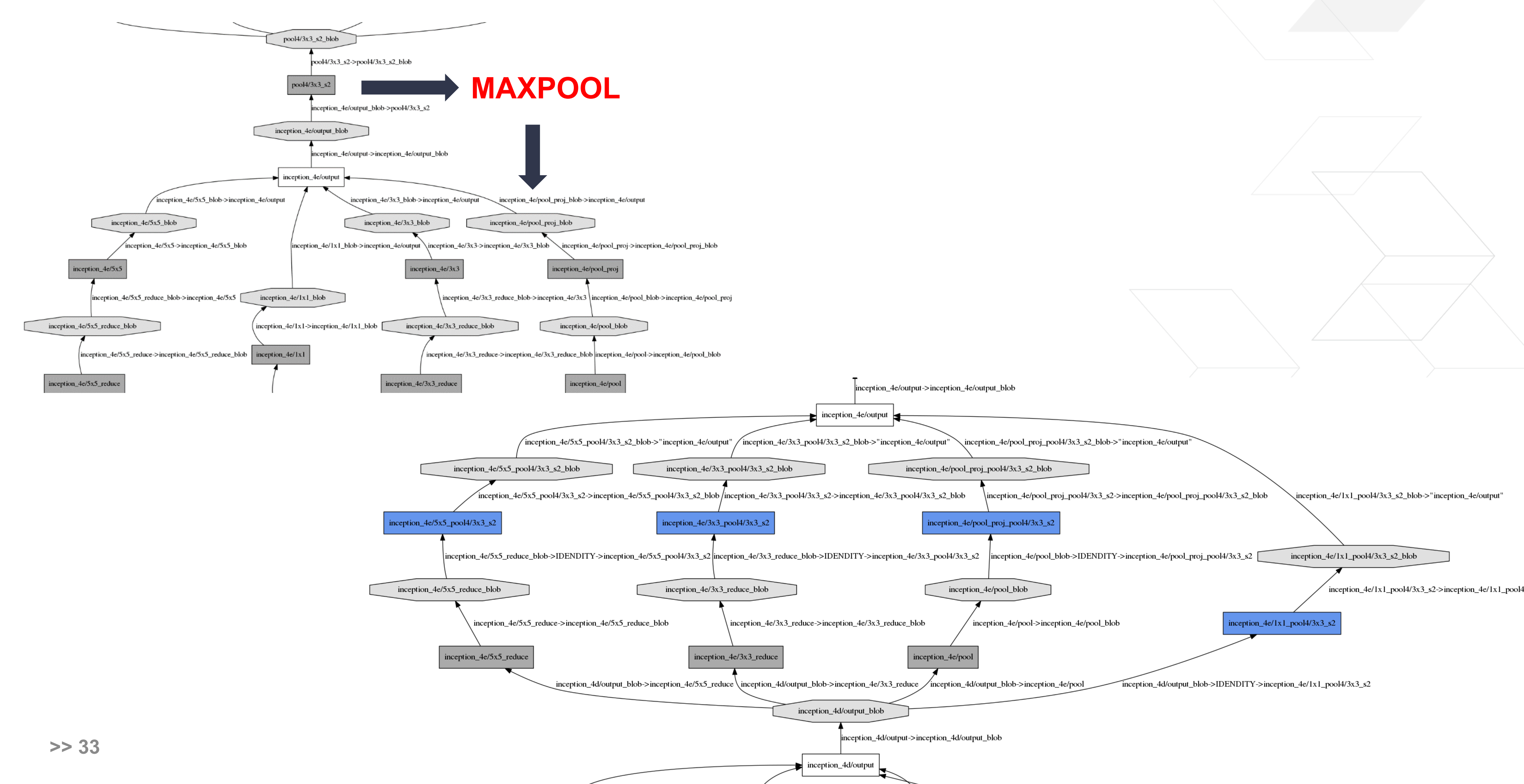
© Copyright 2018 Xilinx

Compiler and Manual Replication

- **Replication is ON by default**
  - If a convolution is not 1x1 it is eligible for Replication
- **The selection of the details about replication**
  - It is hardware dependent
  - The compiler uses abstraction
    - And code provided by HW
- **Replication is a property of the tensors thus:**
  - it is applied before memory computation
  - And everything looks like any other tensor
- **We know Replication is important and we allow external inputs**
  - Users can overwrite any tensor replication
  - By specifying the layer and the input replication

© Copyright 2018 Xilinx

Pooling around … opportunity and pipelining



Pooling around … Rule-based graph manipulations

- We can move (Max)Pooling across concatenation
  - --poolingaround
- We can pipeline Conv + Pooling
  - --pipelineconvmaxpool
- This is a two step graph optimization
  - Rule based == always beneficial

© Copyright 2018 Xilinx

Tiling: from DDR to AM and back to DDR

- **Compiler Memory-Allocation Rules:**
  - Either the inputs *and* the outputs are in AM
  - Or they are in DDR
- **Compiler Applies Heuristics**
  - Where to allocate: By size, tops, bottoms ..
  - How to de-allocate: Least Recently Used policy
- **Tiling: The data is in DDR**
  - V2 the compiler breaks the computation in software tiles
    - AM constraints
  - V3 the compiler estimates the largest tile that
    - Minimizes the overlay space/communication
    - Fit in AM
  - We do tiling at Code Generation time (last stage)

© Copyright 2018 Xilinx

Tiling Examples

V2 can split only the **Height** of the input/output volume

- Channels and width are fixed
    - The minimum tile can be larger than the AM (stop)
    - Multiple instructions

V2 Tiling

```
# spLiT conv1/7x7_s2 2097152 ;
4 XNGather 0x0 0x1c0000 224 224 3 0 1 1 0 113 # #  Conv conv1/7x7_s2 # SPLIT Code :)
5 XNConv conv1/7x7_s2#0 7 7 2 2 3 3 1 1 16 26 2 1 1 0x0 224 114 3 0x39000 112 56 64 0 # SPLIT Code :)
6 XNScatter 0x39000 0x0 112 112 64 0 1 1 0 55 # # Conv conv1/7x7_s2 # SPLIT Code :)
7 XNGather 0x0 0x1c0000 224 224 3 0 1 1 109 223 # #  Conv conv1/7x7_s2 # SPLIT Code :)
8 XNConv conv1/7x7_s2#1 7 7 2 2 3 0 1 1 16 26 2 1 1 0x0 224 115 3 0x39800 112 56 64 0 # SPLIT Code :)
9 XNScatter 0x39800 0x0 112 112 64 0 1 1 56 111 # # Conv conv1/7x7_s2 # SPLIT Code :)
```

V3 Tiling

```
3 XNConv conv1/7x7_s2 7 7 2 2 3 3 1 1 16 26 2 1 1 0x0 224 224 3 0x126000 112 112 64 0 0 1 3 32 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 1 112 38 1 0 24576 # V3 SPLIT Code :)
6 XNGather 0xc40 0x126000 112 112 64 0 1 1 0 111 0 # conv1/7x7_s2_blob
7 XNMaxPool pool1/3x3_s2 3 3 2 2 0 0 0xc40 112 112 64 0x0 56 56 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0
```

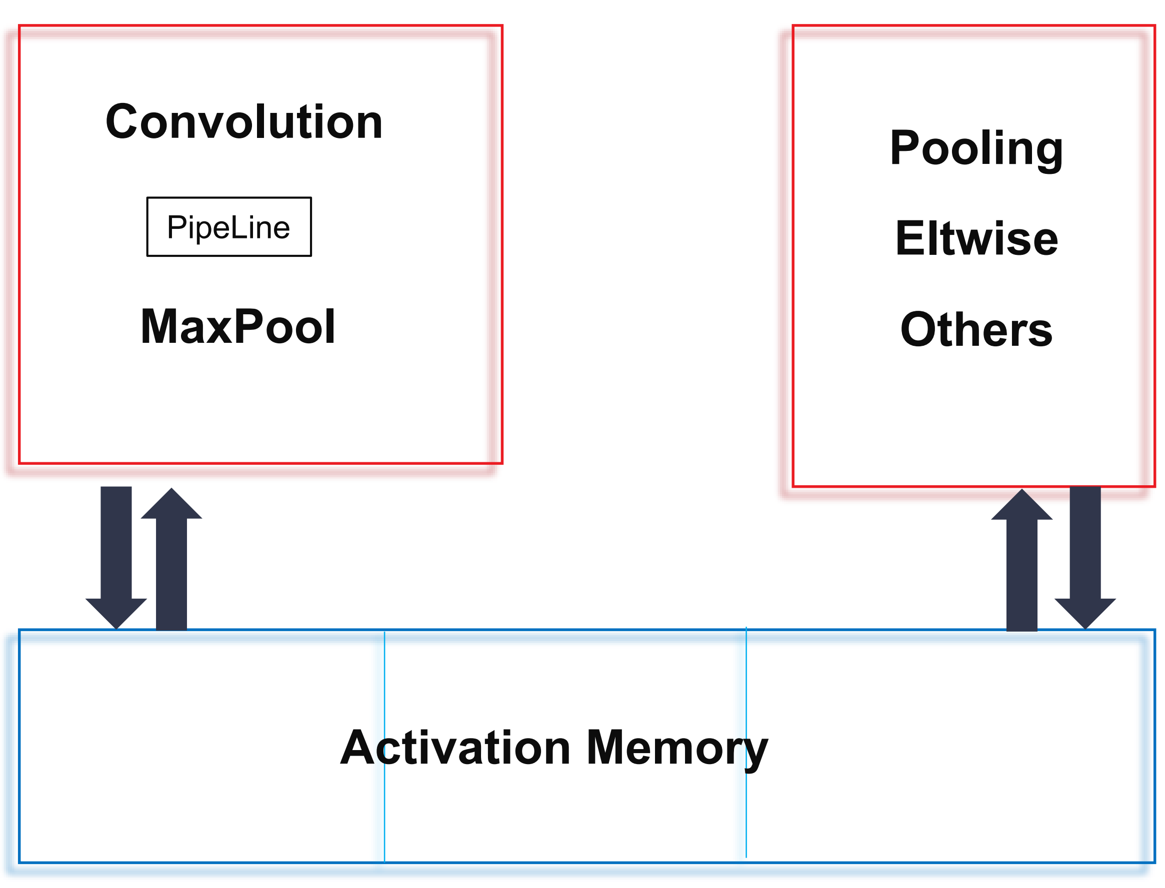
V3 can split **Height and Width** of the input/output volume

- Channels are fixed
- The minimum tile is about the size of the channels + overlay
- One instruction


```
"xdnn_instr": "XNConv conv1_1 3 3 1 1 1 1 1 1 16 26 2 1 1 0x0 480 960 3 0xa8c000 480 960 64 \
0 1 0 0 0 0 2 3 32 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 480 49 1 0 24576 # V3 SPLIT Code :)",
"xdnn_kv": {
      "HAT": "1", "SRCAM-Buffer_0": "0", "SRCAM-Buffer_1": "24576",
      "XNOp": "XNConv", "bias": "1", "concat_full_sect_num": "0", "concat_i_am_your_father": "0", "concat_repl_sect_num": "0",
      "concat_repl_unit_num": "0", "concat_repl_unit_width": "0", "concat_starting_ch": "0", "dilation_h": "1", "dilation_w": "1",
      "dst_full_sect_num": "0", "dst_repl_sect_num": "2", "dst_repl_unit_num": "3", "dst_repl_unit_width": "32", "en_halfrate_mode": "0",
      "en_inlinemaxpool": "0", "en_pingpong_weight": "1", "id": "2", "inaddr": "0x0", "inchan": "3", "insize_h": "960", "insize_w": "480",
      "kernel_h": "3", "kernel_w": "3", "name": "conv1_1", "outaddr": "0xa8c000", "outchan": "64", "outsize_h": "960", "outsize_w": "480",
      "padding_h": "1", "padding_w": "1", "parallel_read": "0", "postshift": "2", "prerelu": "0", "preshift": "16", "relu": "1",
      "scale": "26", "slice": "0", "src_full_sect_num": "1", "src_repl_sect_num": "0", "src_repl_unit_num": "0", "src_repl_unit_width": "0",
      "strides_h": "1", "strides_w": "1", "tile_height": "49", "tile_width": "480", "wait_conv": "1", "wait_download": "1", "wait_ew": "1",
      "wait_pool": "1", "wait_upload": "1", "wait_upsmpl": "1"
},
```


© Copyright 2018 Xilinx

Convolution || Pooling … We have the technology



Obviously,

- Two separated kernels
- Parallel access to AM

***Translate to*,**

- Two *close-by* instructions
- No shared Tensors
- And ***far apart*** Memory Locations

© Copyright 2018 Xilinx

Convolution || Pooling … Opportunity

inception_4c/output
inception_4c/5x5_blob->inception_4c/output
inception_4c/pool_proj_blob->inception_4c/output
inception_4c/3x3_blob->inception_4c/output
inception_4c/5x5_blob
inception_4c/pool_proj_blob
inception_4c/3x3_blob
inception_4c/5x5->inception_4c/5x5_blob
inception_4c/pool_proj->inception_4c/pool_proj_blob
inception_4c/1x1_blob->inception_4c/output
inception_4c/3x3->inception_4c/3x3_blob
inception_4c/5x5
inception_4c/pool_proj
inception_4c/3x3
inception_4c/5x5_reduce_blob->inception_4c/5x5
inception_4c/pool_blob->inception_4c/pool_proj
inception_4c/1x1_blob
inception_4c/3x3_reduce_blob->incept
inception_4c/5x5_reduce_blob
inception_4c/pool_blob
inception_4c/1x1->inception_4c/1x1_blob
inception_4c/3x3_reduce_blob
inception_4c/5x5_reduce->inception_4c/5x5_reduce_blob
inception_4c/pool->inception_4c/pool_blob
inception_4c/1x1
inception_4c/3x3_reduce->inception_
inception_4c/5x5_reduce
inception_4c/pool
inception_4c/3x3_reduce
inception_4b/output_blob->inception_4c/5x5_reduce
inception_4b/output_blob->inception_4c/pool
inception_4b/output_blob->inception_4c/1x1
inception_4b/output_blob->inception_4c/
inception_4b/output_blob
inception_4b/output->inception_4b/output_blob
inception_4b/output

© Copyright 2018 Xilinx

Convolution || Pooling … Codes (V2 simplified)

Regular Schedule: **NOT** parallel and **NOT** optimal

```
41 XNConv inception_4b/pool_proj 1 1 1 1 0 0 1 1 16 26 2 1 1 0x0 14 14 512 0x504000 14 14 64 0
# 42 XNConcat inception_4b/output  0x440000 0x520000 917504
43 XNConv inception_4c/1x1 1 1 1 1 0 0 1 1 16 26 2 1 1 0x440000 14 14 512 0x520000 14 14 128 0
44 XNConv inception_4c/3x3_reduce 1 1 1 1 0 0 1 1 16 26 2 1 1 0x440000 14 14 512 0x0 14 14 128 0
45 XNConv inception_4c/3x3 3 3 1 1 1 1 1 1 16 26 2 1 1 0x0 14 14 128 0x558000 14 14 256 0
46 XNConv inception_4c/5x5_reduce 1 1 1 1 0 0 1 1 16 26 2 1 1 0x440000 14 14 512 0x0 14 14 24 0
47 XNConv inception_4c/5x5 5 5 1 1 2 2 1 1 16 26 2 1 1 0x0 14 14 24 0x5c8000 14 14 64 0
48 XNMaxPool inception_4c/pool 3 3 1 1 1 1 0x440000 14 14 512 0x0 14 14 0
49 XNConv inception_4c/pool_proj 1 1 1 1 0 0 1 1 16 26 2 1 1 0x0 14 14 512 0x5e4000 14 14 64 0
# 50 XNConcat inception_4c/output  0x520000 0x600000 917504
51 XNConv inception_4d/1x1 1 1 1 1 0 0 1 1 16 26 2 1 1 0x520000 14 14 512 0x439000 14 14 112 0
```

Optimality (later)
Data parallelism (short)

Pool
- IN address **High**
- OUT address **Low**

Conv
- IN address **Low**
- OUT address **High**

Optimal Schedule: ☺

```
34 XNConv inception_4b/1x1 1 1 1 1 0 0 1 1 16 26 2 1 1 0x36e000 14 14 512 0x520000 14 14 160 0
# 35 XNConcat inception_4b/output  0x520000 0x600000 917504
36 XNConv inception_4c/5x5_reduce 1 1 1 1 0 0 1 1 16 26 2 1 1 0x520000 14 14 512 0x0 14 14 24 0
37 XNConv inception_4c/5x5 5 5 1 1 2 2 1 1 16 26 2 1 1 0x0 14 14 24 0x4e8000 14 14 64 0
38 XNConv inception_4c/3x3_reduce 1 1 1 1 0 0 1 1 16 26 2 1 1 0x520000 14 14 512 0x0 14 14 128 0
39 XNMaxPool inception_4c/pool 3 3 1 1 1 1 0x520000 14 14 512 0x38000 14 14 0
39 XNConv inception_4c/3x3 3 3 1 1 1 1 1 1 16 26 2 1 1 0x0 14 14 128 0x478000 14 14 256 0
40 XNConv inception_4c/pool_proj 1 1 1 1 0 0 1 1 16 26 2 1 1 0x38000 14 14 512 0x504000 14 14 64 0
41 XNConv inception_4c/1x1 1 1 1 1 0 0 1 1 16 26 2 1 1 0x520000 14 14 512 0x440000 14 14 128 0
# 42 XNConcat inception_4c/output  0x440000 0x520000 917504
43 XNConv inception_4d/5x5_reduce 1 1 1 1 0 0 1 1 16 26 2 1 1 0x440000 14 14 512 0x520000 14 14 32 0
```


© Copyright 2018 Xilinx

Convolution || Pooling … Inceptionality ?

An **Inception** is a sub graph

- N(x) specific topological order
- An inception is specified by N
    - Graph property

Inception is self contained

- We explore 6,000 schedules
- Only 600 are valid
    - Up to a max 10,000

Each optimal schedule glued together into a single schedule

```
inception inception_4b/output - inception_4c/output
depth tfs dfs name [ inputs name']
#  0 18 18 42 inception_4b/output ops 0
#        INPUT [u'inception_4b/5x5', u'inception_4b/1x1', u'inception_4b/pool_proj', u'inception_4b/3x3']
#        OUTPUT [u'inception_4c/1x1', u'inception_4c/pool', u'inception_4c/5x5_reduce', u'inception_4c/3x3_reduce']
#  1 19 19 44 inception_4c/pool ops 1806336
#        INPUT [u'inception_4b/output']
#        OUTPUT [u'inception_4c/pool_proj']
#  2 19 19 46 inception_4c/5x5_reduce ops 9408
#        INPUT [u'inception_4b/output']
#        OUTPUT [u'inception_4c/5x5']
#  3 19 19 43 inception_4c/1x1 ops 50176
#        INPUT [u'inception_4b/output']
#        OUTPUT [u'inception_4c/output']
#  4 19 19 48 inception_4c/3x3_reduce ops 50176
#        INPUT [u'inception_4b/output']
#        OUTPUT [u'inception_4c/3x3']
#  5 20 20 49 inception_4c/3x3 ops 903168
#        INPUT [u'inception_4c/3x3_reduce']
#        OUTPUT [u'inception_4c/output']
#  6 20 20 47 inception_4c/5x5 ops 627200
#        INPUT [u'inception_4c/5x5_reduce']
#        OUTPUT [u'inception_4c/output']
#  7 20 20 45 inception_4c/pool_proj ops 25088
#        INPUT [u'inception_4c/pool']
#        OUTPUT [u'inception_4c/output']
sorted path by memory use
(114688.0, [u'inception_4b/output', u'inception_4c/5x5_reduce', u'inception_4c/5x5', u'inception_4c/output'])
(229376.0, [u'inception_4b/output', u'inception_4c/1x1', u'inception_4c/output'])
(458752.0, [u'inception_4b/output', u'inception_4c/3x3_reduce', u'inception_4c/3x3', u'inception_4c/output'])
(917504.0, [u'inception_4b/output', u'inception_4c/pool', u'inception_4c/pool_proj', u'inception_4c/output'])
5040
tick search 0 of 5040
Number of valid schedules 630
```


© Copyright 2018 Xilinx

Convolution || Pooling … optimality by Pareto search

```
The best schedule,  with min average space, and min average ops 8 1745408.0 3.75 2568384
0 inception_4b/output
1 inception_4c/5x5_reduce
2 inception_4c/5x5
3 inception_4c/3x3_reduce
4 inception_4c/pool
5 inception_4c/3x3
6 inception_4c/pool_proj
7 inception_4c/1x1
8 inception_4c/output
Committed Ins # 8
Concat  ▲  I:size  917504  POOLING:None,CONVOLUTION:None,_ANY_:inception_4b/output
           live:[u'inception_4b/output'] ops:0
        ▲  I:size  960512  POOLING:None,CONVOLUTION:inception_4c/5x5_reduce,_ANY_:None
           live:[u'inception_4c/5x5_reduce', u'inception_4b/output'] ops:9408
        ▲  I:size 1075200  POOLING:None,CONVOLUTION:inception_4c/5x5,_ANY_:None
           live:[u'inception_4c/5x5_reduce', u'inception_4b/output', u'inception_4c/5x5'] ops:627200
        ▲  I:size 1261568  POOLING:None,CONVOLUTION:inception_4c/3x3_reduce,_ANY_:None
           live:[u'inception_4c/3x3_reduce', u'inception_4b/output', u'inception_4c/5x5'] ops:50176
BINGO   ▲  I:size 2637824  POOLING:inception_4c/pool,CONVOLUTION:inception_4c/3x3,_ANY_:None
           live:[u'inception_4c/3x3_reduce', u'inception_4b/output', u'inception_4c/5x5', u'inception_4c/3x3', u'inception_4c/pool'] ops:1806336
        ▲  I:size 2523136  POOLING:None,CONVOLUTION:inception_4c/pool_proj,_ANY_:None
           live:[u'inception_4c/3x3', u'inception_4c/pool', u'inception_4c/pool_proj', u'inception_4b/output', u'inception_4c/5x5'] ops:25088
        ▲  I:size 1835008  POOLING:None,CONVOLUTION:inception_4c/1x1,_ANY_:None
           live:[u'inception_4c/3x3', u'inception_4c/pool_proj', u'inception_4c/1x1', u'inception_4b/output', u'inception_4c/5x5'] ops:50176
Concat  ▲  I:size 2752512  POOLING:None,CONVOLUTION:None,_ANY_:inception_4c/output
           live:[u'inception_4c/3x3', u'inception_4c/pool_proj', u'inception_4c/1x1', u'inception_4b/output', u'inception_4c/5x5', u'inception_4c/output'] ops:0
The best live 9 1640675.5555555555 3.6666666666666665
```

Simplified schedule

Dimensions to minimize:

1. Number of parallel steps
2. Number of live tensors

This is a full search

© Copyright 2018 Xilinx

Convolution || Pooling … Code generation (V3)

```
PARALLEL >
 inception_4c/pool inception_4c/3x3 Wait(Wait_Download=1, Wait_Upload=1, Wait_Conv=1, Wait_Pool=0, Wait_EW=1, Wait_Upsmpl=1, ParalleRead=0)
  inception_4c/3x3 IN 9361920-9437184 OUT 313600-413952
 inception_4c/pool IN 37632-263424 OUT 9136128-9361920
```

- **AS the last step**
  - We reach the convolution instruction
  - We know that there is a possible parallel Pool
  - We check the data parallelism
- **Parallel**
  - Parallel: do not wait for the completion of the Pooling

NOTES: V3 allows explicit parallelism

- **scheduling, inst-parallelism, and data-parallelism are architecture independent.**

© Copyright 2018 Xilinx

Improvements Due to Compiler Optimizations

Optimization	
No opt	100.00%
Replication	57.50%
Replication + Conv/Pool pipeline	54.20%
Replication + Conv/Pool pipeline + Parallel	53.97%
Replication + Conv/Pool pipeline + Pooling around	53.18%
Parallel	51.92%
All auto-opts	46.71%
Conv/Pool pipeline + Parallel + dedicated 1st block	38.68%
All auto-opts + dedicated 1st block	36.85%
Conv/Pool pipeline + Parallel + “manual” replication	36.69%
All auto-opts + “manual” replication	34.84%

Optimization	
No opt	100.00%
Replication	73.49%
Replication + Conv/Pool pipeline	72.00%
Replication + Conv/Pool pipeline + Parallel	68.88%
Replication + Conv/Pool pipeline + Parallel + “manual” replication	61.01%
All auto-opts + “manual” replication	55.13%

Resnet50

© Copyright 2018 Xilinx

ResNet50 Layer-wise Efficiency

EFFICIENCY (%)

100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%

res2a_branch1
res2a_branch2a
res2a_branch2b
res2a_branch2c
res2b_branch2a
res2b_branch2b
res2b_branch2c
res2c_branch2a
res2c_branch2b
res2c_branch2c
res3a_branch1
res3a_branch2a
res3a_branch2b
res3a_branch2c
res3b_branch2a
res3b_branch2b
res3b_branch2c
res3c_branch2a
res3c_branch2b
res3c_branch2c
res3d_branch2a
res3d_branch2b
res3d_branch2c
res4a_branch1
res4a_branch2a
res4a_branch2b
res4a_branch2c
res4b_branch2a
res4b_branch2b
res4b_branch2c
res4c_branch2a
res4c_branch2b
res4c_branch2c
res4d_branch2a
res4d_branch2b
res4d_branch2c
res4e_branch2a
res4e_branch2b
res4e_branch2c
res4f_branch2a
res4f_branch2b
res4f_branch2c
res5a_branch1
res5a_branch2a
res5a_branch2b
res5a_branch2c
res5b_branch2a
res5b_branch2b
res5b_branch2c
res5c_branch2a
res5c_branch2b
res5c_branch2c

RESNET50 LAYERS

© Copyright 2018 Xilinx

Graph Partitioning

> Automatically Partition and Code generation of Tensorgraph
> Data Flow Execution using multiple xDNN/FPGAs

FPGA or CPU | FPGA | CPU | FPGA | CPU

Pre-Processing | Subgraph 1 | Parallel Subgraphs | Post-Processing

1 Core -> Multi-Core -> Multi-Chip

© Copyright 2018 Xilinx

xfDNN Quantizer

© Copyright 2018 Xilinx

Quantization

- Quantization：
  - Represent parameter and activation with lower bit width
- Advantages：
  - Smaller
  - Faster
  - Power efficiency
  - Hardware Friendly
- Challenge：
  - Accuracy

	Dynamic Range	Min Positive Value
FP32	-3.4×10^{38} ~ $+3.4 \times 10^{38}$	1.4×10^{-45}
FP16	-65504 ~ +65504	5.96×10^{-8}
INT8	-128 ~ +127	1

© Copyright 2018 Xilinx

Quantization Strategy

E.g.： FP32->INT3

- 8 discrete values（-4,-3,…,3）

- Optimization Target:
  - E.g. Non-overflow
  - E.g. Minimize L2 loss: $\Sigma (X-X')^2$
- Methods：
  - Uniform vs. Nonuniform
  - Symmetry vs. Asymmetry ：
    - Zero-Shift(*TensorFlow*)
  - Float Scale vs. Int Scale

a）Uniform Quantization

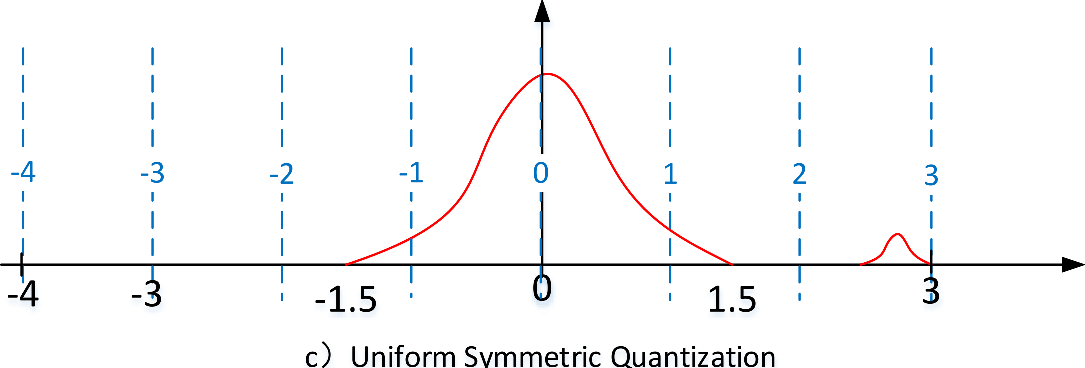
b）Ununiform Quantization

c）Uniform Symmetric Quantization

© Copyright 2018 Xilinx

Our Strategy

- Quantization Strategy
  - Uniform Symmetric Quantization → 8Bit for DeePhi DPU
  - Scale = 2^N



c) Uniform Symmetric Quantization

- Advantages
  - Simple hardware design
  - High efficiency: all fix-point calculation
  - Make use of redundancy of CNN models(especially with BatchNorm Layers)

© Copyright 2018 Xilinx

Quantize, Dequantize, Quantize position

- Uniform Symmetric Quantization：
  - Scale: 2^(-p)
  - Quantize:
    - $X_q = F(X_f) = round(X_f / scale) = round(2^{(p)} * X_f)$
  - Dequantize:
    - $X_d = F'(X_q) = X_q * Scale = 2^{(-p)} * X_q$
  - Range of dequantized value：
    - [-128*2^(-p) , 127*2^(-p)]
  - Quantize Pos:
    - $p = \log_2(1 / AbsMax(X_{min}/128, X_{max}/127))$

p	scale	min	max
-2	4	-512	508
-1	2	-256	254
0	1	-128	127
1	0.5	-64	63.5
2	0.25	-32	31.75
3	0.125	-16	15.875
4	0.0625	-8	7.9375

…… | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | -1 | -2 | ……

p: the decimal position of quantized value

© Copyright 2018 Xilinx

Example

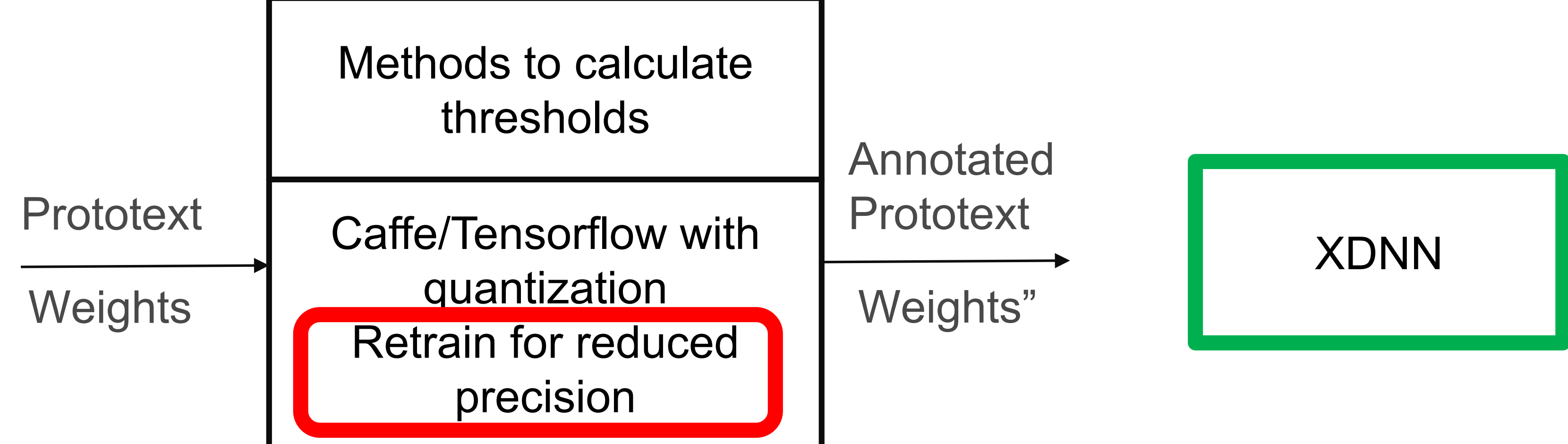
- Example: Float→Int8
  - Float value: {-6.0625, -8.013, 4.438}
- Steps：
  - Float range: [-8.013~4.438]
  - Quantize pos: p = 3
  - Scale: 2^(-3) = 0.125
  - Quantize: x_q = round(x / scale) = round(x * 8)
  - Dequantize: $x_d = x_q$ * scale = x_q * 0.125

p	scale	min	max
-2	4	-512	508
-1	2	-256	254
0	1	-128	127
1	0.5	-64	63.5
2	0.25	-32	31.75
3	0.125	-16	15.875
4	0.0625	-8	7.9375

X	X_q(p=3)	X_d
-6.0625	-49	-6.125
-8.013	-64	-8
4.438	36	4.5

© Copyright 2018 Xilinx

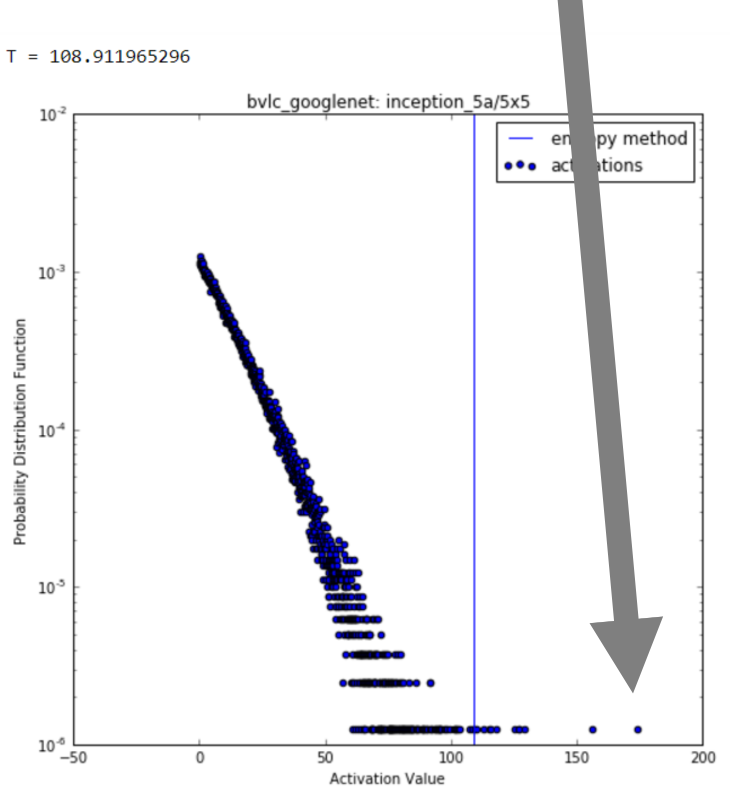
Overall Quantization Flow

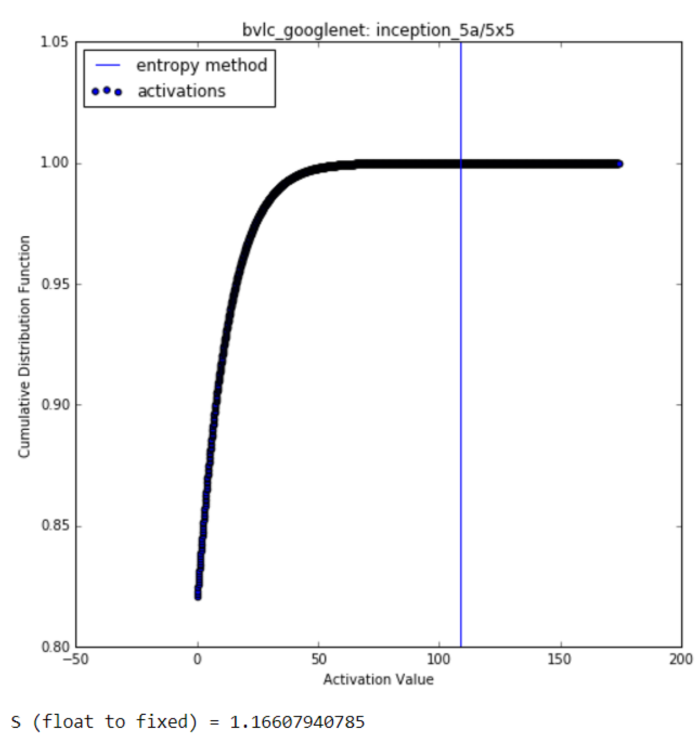


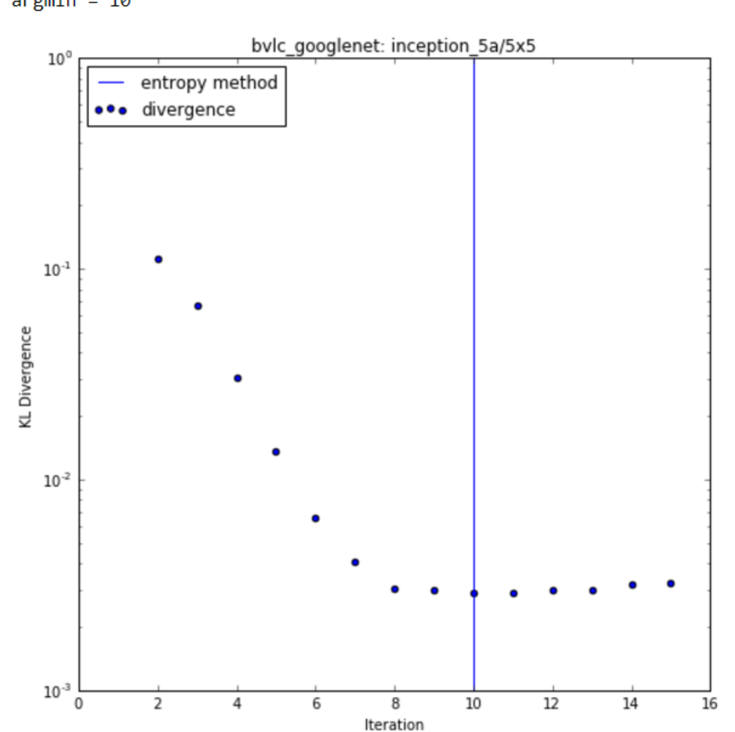
© Copyright 2018 Xilinx

Offline Quantization Procedure

- Walk the network graph, analyze distributions, and statistically determine thresholds for weights and activations using a calibration set
  - Where is the information in your deep learning models?







© Copyright 2018 Xilinx

Example Results

Network	Dataset	Values	weights	Top1 %	diff	Top5 acc	diff
Resnet-50	ImageNet	Floating P.	Original	72.97	0.00	91.13	0.00
		8 bits	Quantized	70.67	-2.30	89.79	-1.34
		8 bits	Retrained	**73.39**	0.43	**91.47**	0.34
	Flowers102	Floating P.	Original	85.38	0.00	97.38	0.00
		8 bits	Quantized	81.57	-3.81	96.37	-1.00
		8 bits	Retrained	**85.10**	-0.28	**97.06**	-0.32
	Places365	Floating P.	Original	52.38	0.00	83.00	0.00
		8 bits	Quantized	50.99	-1.39	82.39	-0.61
		8 bits	Retrained	**53.80**	1.43	**84.28**	1.28

Retraining recovers accuracy, and uses initial weights from offline quantization
Retraining time is significantly lower than training from scratch.

© Copyright 2018 Xilinx

xfDNN Runtime

© Copyright 2018 Xilinx

xfDNN Runtime Engine

> Lightweight and portable with no dependency on ML frameworks
  >> Uses JSON exported by Compiler and Quantizer
  >> Generate binary for xDNN

> Extensive C++/Python API with simplified use model

> Asynchronous XDNN execution

> Streaming/pipeline/Dataflow support for large imageset

> Multiple CNN models running on single FPGA

> Multiple FPGA support

> Support for both 16 and 8 bits

> CPU accelerated functions

© Copyright 2018 Xilinx

Simplify xdnn usage

```
import xdnn, xdnn_io

args = xdnn_io.processCommandLine()
xdnn.createHandle(args['xclbin'], "kernelSxdnn_0", args['xlnxlib'])

(weightsBlob, fcWeight, fcBias) = xdnn_io.loadWeights(args)

(fpgaInputs, batch_sz) = xdnn_io.prepareInput(args)
fpgaOutput = xdnn_io.prepareOutput(args['fpgaoutsz'], batch_sz)

xdnn.execute(args['netcfg'],
  weightsBlob, fpgaInputs, fpgaOutput,
  batch_sz, args['quantizecfg'])

fcOut = xdnn.computeFC(fcWeight, fcBias, fpgaOutput)

softmaxOut = xdnn.computeSoftmax(fcOut)

xdnn_io.printClassification(softmaxOut, args);

xdnn.closeHandle()
```

Blocking

```
# exec_async() enqueues FPGA task and returns immediately
xdnn.exec_async(args['netcfg'],
  weightsBlob, fpgaInputs, fpgaOutput,
  batch_sz, args['quantizecfg'], args['PE'])

# get_result() blocks for FPGA result
xdnn.get_result(args['PE'])
```

```
numFPGAs = 8
args = xdnn_io.processCommandLine()
xdnn.createHandle(args['xclbin'], "kernelSxdnn_0", args['xlnxlib'], numFPGAs)

(weightsBlob, fcWeight, fcBias) = xdnn_io.loadWeights(args)
```

8-FPGA

© Copyright 2018 Xilinx

XILINX

Streaming/pipeline/dataflow

```
def main():
  qPrep = Queue(maxsize=1)
  qFpga = Queue(maxsize=1)

  prepProc = Process(target=prep_process, args=(qPrep,))
  xdnnProc = Process(target=xdnn_process, args=(qPrep, qFpga))
  postProc = Process(target=post_process, args=(qFpga,))

  prepProc.start()
  xdnnProc.start()
  postProc.start()
```

```
def prep_process(q):
  ret = xdnn.createManager(g_xdnnLib)

  cInputBuffer = None
  cFpgaInputBuffer = None
  while True:
    (inputs, inputImageFiles) = prepareImages()

    fpgaInputs = xdnn.quantizeInputs(g_firstFpgaLayerName,
      inputs, cInputBuffer, cFpgaInputBuffer, g_fpgaCfgFile)

    q.put((fpgaInputs, inputImageFiles))
```

```
def xdnn_process(qFrom, qTo):
    xdnn.createHandle(g_xclbin, "kernelSxdnn_0", g_xdnnLib)
    weightsBlob = loadWeightsBiasQuant()

    fpgaOutput = None
    while True:
        (inputs, inputImageFiles) = qFrom.get()
        fpgaInputs = prepareFpgaInputs(inputs)
        if not fpgaOutput:
          fpgaOutput = prepareOutput(g_batchSize)

        xdnn.execute(g_netFile,
          weightsBlob, fpgaInputs, fpgaOutput,
          g_batchSize, g_fpgaCfgFile)

        np_out = np.frombuffer(fpgaOutput, np.float32)
        qTo.put((np_out, inputImageFiles))
```

```
def post_process(qFpga):
  (fcWeight, fcBias) = loadFCWeightsBias()

  while True:
    (fpgaOutput, inputImageFiles) = qFpga.get()

    fcOutput = xdnn.computeFC(fcWeight, fcBias, fpgaOutput)
    smaxOutput = xdnn.computeSoftmax(fcOutput)

    printClassification(smaxOutput, inputImageFiles, g_labelFile)
```

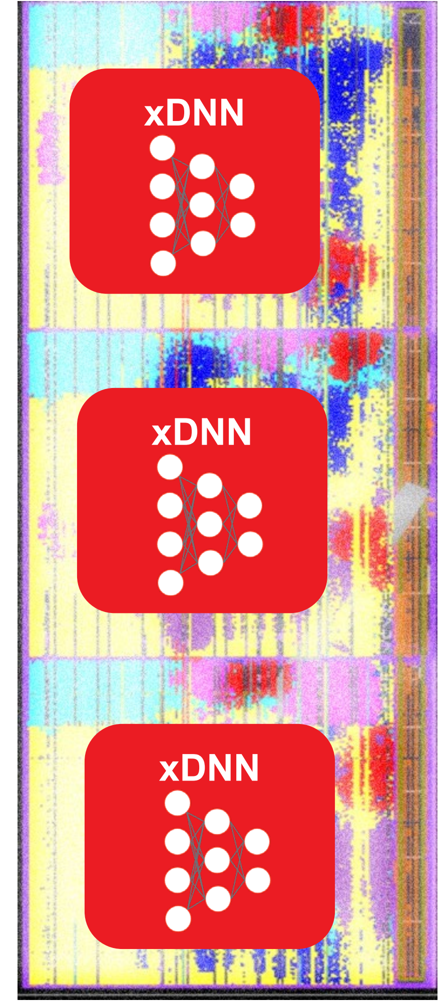
© Copyright 2018 Xilinx

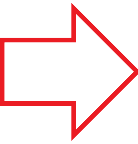
Multiple CNN models running on single FPGA

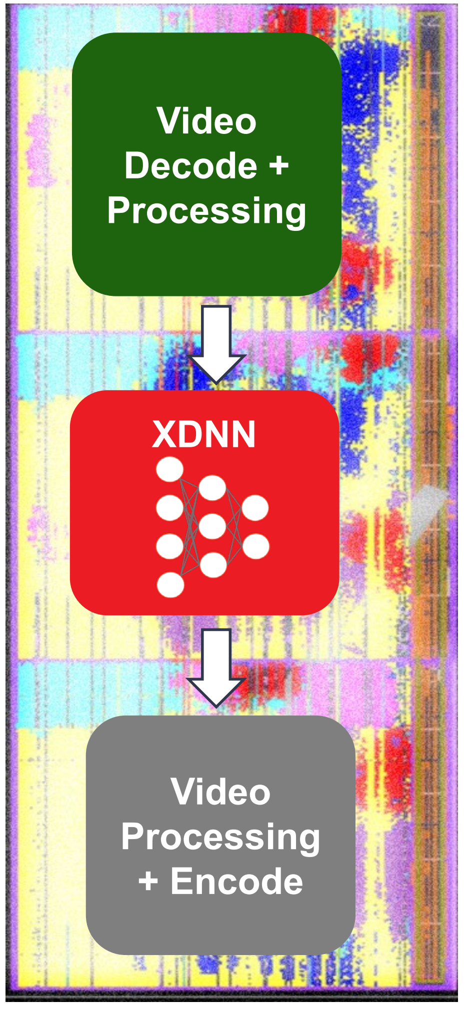
```
args = xdnn_io.processCommandLine()
xdnn.createHandle(args['xclbin'], "kernelSxdnn_0", args['xlnxlib'])

# setup/pre-process
for netCfg in args['fpgacfg']:
  netID = netCfg['name']
  PE = [int(x) for x in netCfg['PE'].split()]
  PEs[netID] = PE

  (weightsBlobs[netID], fcWeights[netID], fcBiases[netID]) = xdnn_io.loadWeights(netCfg)
  (fpgaInputs[netID], batchSize) = xdnn_io.prepareInput(netCfg)
  batchSizes[netID] = batchSize
  fpgaOutputs[netID] = xdnn_io.prepareOutput(int(netCfg['fpgaoutsz']), batchSize)
  netFiles[netID] = netCfg['netcfg']

# exec different networks on different PEs (async)
for netCfg in args['fpgacfg']:
  netID = netCfg['name']
  xdnn.exec_async(netFiles[netID], weightsBlobs[netID], fpgaInputs[netID],
    fpgaOutputs[netID], batchSizes[netID], 1, netCfg['quantizecfg'], PEs[netID])

# collect results
for netID, pe in PEs.iteritems():
  xdnn.get_result(pe)

# post-process
for netCfg in args['fpgacfg']:
  netID = netCfg['name']
  fcOut = xdnn.computeFC(fcWeights[netID], fcBiases[netID], fpgaOutputs[netID])
  softmaxOut = xdnn.computeSoftmax(fcOut)
  xdnn_io.printClassification(softmaxOut, netCfg);
```

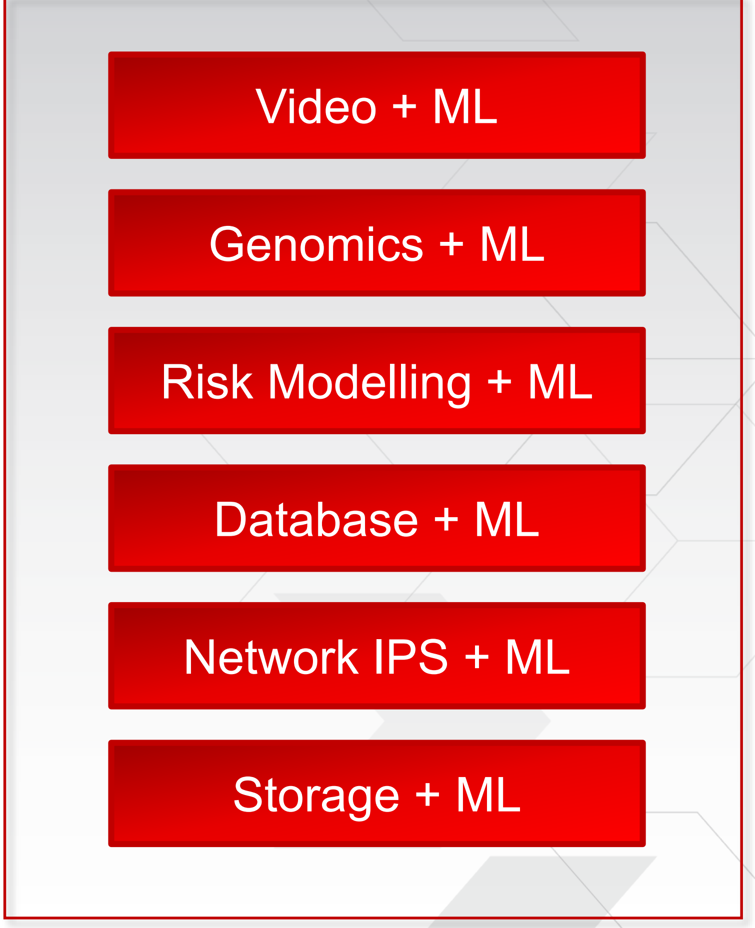
```
fpgacfg:
[
  {
    "name":"googlenet_1",
    "net":"googlenet",
    "datadir":"data",
    "netcfg":"googlenet.fpgaaddr.64.txt",
    "PE":"1",
    "firstFpgaLayerName":"conv1/7x7_s2",
    "fpgaoutsz":"1024"
    },
  {
    "name":"resnet_2",
    "net":"resnet",
    "datadir":"data_resnet",
    "netcfg":"resnet.fpgaaddr.64.txt",
    "PE":"2",
    "firstFpgaLayerName":"conv1/7x7_s2",
    "fpgaoutsz":"2048"
  }
]
```

Deep Learning Infusion









Integrate Custom Applications with xDNN. Lower end-to-end latency

© Copyright 2018 Xilinx

Adaptable.
Intelligent.

© Copyright 2018 Xilinx