NMAX: Fast, Modular, Low Latency, Low Cost/Power Neural Inferencing
Leader in eFPGA

• TSMC IP Alliance Member
• eFPGA working Silicon TSMC 40/28/16/12
• eFPGA in design for GF14 and TSMC 7/7+
• >>10 customer deals/design/fab/silicon: 16-180nm; more in negotiation
• First 6 announced agreements
NMAX Value Proposition

- Inferencing: 8x8 or 16x8 natively; 16x16 at ½ throughput
- High MAC utilization: more throughput out of less silicon
- Low DRAM bandwidth: more throughput at less system cost and power
- Low latency: high performance, low latency at batch = 1
- Modular: can respond quickly to customer needs
- Scalable: doubling MACs means doubling throughput
- Flexible: run any NN or multiple NN
- Tensorflow/Caffe: easy to program
NMAX Inferencing Applications

- **Edge**
  - Automotive
  - Aerospace
  - Cameras
  - Cell Phones
  - Drones
  - PCs
  - Retail
  - Robotics
  - Speech
  - Surveillance

- **Data Centers**
Comparing Neural Engine Alternatives
Neural Inferencing Challenges & Terminology

- An inferencing chip with 1000 MACs @ 1GHz has 1 TMACs/sec = 2 TOPS
  - This is a peak number: no one uses all of the MACs
- Challenge: move data at low power where needed to keep MACs utilized
- Challenge: maintain high performance at batch=1 for lowest latency
How do you get the NN throughput you need?

1. Determine how many OPS/MACs are needed for each image
   - YOLOv3 2MP = 400 Billion MACs/image = 800 Billion Operations/image

2. Determine how many images/second you need to process
   - YOLOv3 2MP autonomous driving: 30 image/sec = 24 TOPS throughput

3. How many MACs you need is determined by this formula:
   - Y TOPS Peak = X TOPS Throughput \( \div \) MAC utilization
   - MAC utilization will vary based on NN, image size, batch size
   - Batch=1 is what you need at the edge
   - Number of MACs required = Y TOPS Peak \( \div \) Frequency of MAC Completion
   - NOTE: no short cuts in the above for pruning model, Winograd, compression
MAC Utilization / MAC Efficiency

• A MAC can only do a useful calculation if both the activation and the weight are available on the inputs; if not it stalls
• MAC Utilization = (# of useful MAC calculations) ÷ (# of MACs Available)
• Example:
  ▪ Nvidia Tesla T4 claims 3920 images/second on ResNet-50 @ Batch=28
  ▪ Each ResNet-50 image takes 7 Billion Operations (3.5 Billion MACs)
  ▪ So T4 is doing 3920 * 7 Billion Ops = 27.44 Trillion Ops/sec = 27.44 TOPS
  ▪ T4 data sheet claims 130 TOPS (int8)
  ▪ So T4 MAC utilization, for Resnet-50 @ Batch=28, is 21%
Microsoft BrainWave Slide from HotChips 2018:

Batching improves HW utilization but increases latency

Ideally want high HW utilization at low batch sizes
ResNet-50 Int8

- Image classification
- 224 x 224 pixels
- 50 stage neural network
- 22.7 Million weights
- 3.5 Billion MACs per image = 7 Billion Operations per image
ResNet-50 Images/Second vs Batch Size: NMAX utilization high at batch=1
Real Time Object Recognition: YOLOv3
YOLOv3 Int8

- Real time object recognition
- 2 Megapixel images
- >100 stage neural network
- 62 Million weights
- 400 Billion MACs per image = 800 Billion Operations per image
**NMAX: YOLOv3, 2048x1024, Batch=1 using 2 x 4Gbit LPDDR4 DRAM**

10x reduction in DRAM BW requirements vs competing solutions (<25 vs >300GB/s)

<table>
<thead>
<tr>
<th>NMAX array size</th>
<th>12x12</th>
<th>12x6</th>
<th>6x6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM Size</td>
<td>64MB</td>
<td>64MB</td>
<td>32MB</td>
</tr>
<tr>
<td>TOPS Peak</td>
<td>147</td>
<td>73</td>
<td>37</td>
</tr>
<tr>
<td>Throughput (1GHz)</td>
<td>124 fps</td>
<td>72 fps</td>
<td>27 fps</td>
</tr>
<tr>
<td>Latency</td>
<td>8 ms</td>
<td>14 ms</td>
<td>37 ms</td>
</tr>
<tr>
<td>Avg. DRAM BW</td>
<td>12 GB/s</td>
<td>14 GB/s</td>
<td>10 GB/s</td>
</tr>
<tr>
<td>Avg. SRAM BW</td>
<td>177 GB/s</td>
<td>103 GB/s</td>
<td>34 GB/s</td>
</tr>
<tr>
<td>XFLX &amp; ArrayLINX BW</td>
<td>18 TB/s</td>
<td>10 TB/s</td>
<td>4 TB/s</td>
</tr>
<tr>
<td>MAC Efficiency</td>
<td>67% 98 TOPS Throughput</td>
<td>78% 58 TOPS Throughput</td>
<td>58% 22 TOPS Throughput</td>
</tr>
</tbody>
</table>

No NDA Required – Public Information
Why is NMAX the right solution for performance inferencing

• The most efficient implementation of any neural network is a hardwired ASIC
• But customers want reconfigurability
• NMAX is the closest reconfigurable architecture to hardwired ASIC
  ▪ Each stage when configured executes just like an ASIC
• NMAX can run any neural network running Tensorflow/Caffe
NMAX512 Tile Microarchitecture: 1 TOPS @ <2mm² in TSMC16FFC/12FFC

Features

• 8 NMAX clusters achieves 50-90% MAC efficiency

• Local eFPGA logic (EFLX) for:
  ▪ Control logic & management
  ▪ Reconfigurable data flow
  ▪ Additional signal processing (e.g. ReLU, Sigmoid, Tanh)

• Local L1 SRAM for weights & activations

• L2 SRAM (via RAMLINX)

• L3 storage through DDR/PCIe

• High speed XFLX interconnects all blocks within the tile

• High speed ArrayLINX connects to adjacent NMAX tiles to create larger NMAX arrays by abutment

*architectural diagram, not to scale
Every Tile is Reconfigured (quickly & differently) every stage

NMAX512 Tile*

This example does a matrix multiply of a 512 activation vector from the prior stage times a weight matrix which is then activated to produce the activation vector for the next stage.

Input Activation from L2 SRAM

Output Activation to L2 SRAM

No NDA Required – Public Information

*architectural diagram, not to scale
NMAX Clusters Systolically Multiply the Activation Vector by the Weights

- Example of a 4 input vector multiplying by a 4x4 weight matrix

Modular NMAX arrays: easily scales from 1 to >100 TOPS

Features

- NMAX tiles form arrays by abutment
- ArrayLINX interconnect on all 4 sides of NMAX tile automatically connect to provide high bandwidth array-wide interconnect
- Shared L2 SRAM:
  - Local, high-capacity SRAMs placed in between NMAX tiles
  - Holds weights for each layer, as well as activations from one layer to the next
  - EFLX place-and-route algorithms minimizes interconnect distances between SRAM and NMAX

*architectural diagram, not to scale*
NMAX is dataflow: NMAX Compiler maps from Caffe/TensorFlow

- Mapped NN automatically “unrolls” onto the NMAX hardware
- Control logic & data operators maps to EFLX reconfigurable logic