Empowering AI Implementation:
The Versatile SLAC Neural Network Library (SNL) for FPGA, eFPGA, ASIC

SLAC TID

Ryan Herbst, Ryan Coffee, J.J Russell, Abhilasha Dave, Dionisio Doering, Larry Ruckman

November 7 2023
Key points:

- Provides specialized set of libraries designed in High-Level Synthesis (HLS) for deploying ML inferences on FPGAs, eFPGAs and ASICs.
- At the edge of the data chain, SNL aims to create a high-performance, low-latency FPGA implementation for AI inference engines.
- Supports Keras like API for layer definition.
- Dynamic reloading of weights and biases to avoid re-synthesis.
- Supports 10s of thousands of parameters or more depending on latency requirements for the inference model.
- Total end to end latency of ~couple of usec to couple of millisecond.
- Streaming interface between layers.
FPGAs for Latency Critical Neural Network

In context of neural network inference run ASICs, FPGAs, CPUs, GPUs comes with their own strengths and weaknesses.

**CPUs**
- CPU offers the greatest programming flexibility.
- Lower compute throughput.
- Limited Parallelism: CPUs are optimized for single-threaded or limited multi-threaded performance, which is often not sufficient for highly parallel workloads like neural network inference.

**GPUs**
- GPU performance is typically much higher than a CPU and improved further when using a large batch number.
- Processing many queries in parallel.

**FPGAs**
- In latency critical real-time systems, it is not always possible to batch input data.
- This is where FPGAs are somewhat unique, allowing neural networks to be optimized for a single query.
- Still achieve a high-level compute resource utilization.
- When an ASIC does not exist, this makes FPGAs ideal for latency critical neural network processing.

**ASICs**
- ASICs offer the highest performance and lowest cost.
- Only for targeted algorithms.
- There is no flexibility.
On-the-fly Weights & Biases Loading

- SNL implementation is targeting **scientific instruments** which will continuously adapt to new data and **changing environments**
  - High speed training to support this goal
  - Bias and weight **updates in real time**

New Data Set

Real-time experiment

Streaming input data

FPGA based ML inference run

Reload the new weights and biases without having to resynthesize the network

Remote Retraining
Buildable Neural Networks and Future Support

- **Supported Libraries:**
  - **NN Layers:** Conv2D, MaxPooling2D, Average Pooling, Dense
  - **Activators:** LeakyRelu, Relu, SoftMax
  - **Data Types:** Fixed point, Integer, Floating Point

**CNNs/DNNs**

**Fully Connected style Deep Autoencoders**

**MLPs**

- SNL is a work in progress library
- SNL can provide future support for more libraries for variety of neural networks layer types.
  - e.g: Support to foundational Transformer Neural Network blocks
- It can also provide Novel weights loading such as binary or ternary.

Work in Progress
Challenges with Pruning

- Some AI-to-FPGA frameworks take the weights and biases and pruning portions of the network structure to save resources
  - Re-synthesis is required for each new training set
  - Risk of the FPGA implementation failing due to increased resources usage, timing failures or massive change in internal interconnect structure

Neural Network

New Data Set

Training & pruning

Resynthesis Place & Route

FPGA

Challenging, Risky Chance of failure

Time Consuming

Pruned Network for Retraining

Retraining & pruning

● Earlier pruned weights came back
● NW Became heavier
SNL Model Example: Tiny CNN - MNIST

- Parameters = 7312
- Latency = 1023 cycles
  ~5.115us

### NW Definition and Area Consumption KCU105

<table>
<thead>
<tr>
<th>Resource</th>
<th>Utilization</th>
<th>Available</th>
<th>Utilization %</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT</td>
<td>98953</td>
<td>242400</td>
<td>40.82</td>
</tr>
<tr>
<td>LUTRAM</td>
<td>10081</td>
<td>112800</td>
<td>8.94</td>
</tr>
<tr>
<td>FF</td>
<td>124674</td>
<td>484800</td>
<td>25.72</td>
</tr>
<tr>
<td>BRAM</td>
<td>100.50</td>
<td>600</td>
<td>16.75</td>
</tr>
<tr>
<td>DSP</td>
<td>448</td>
<td>1920</td>
<td>23.33</td>
</tr>
<tr>
<td>IO</td>
<td>7</td>
<td>520</td>
<td>1.35</td>
</tr>
<tr>
<td>GT</td>
<td>8</td>
<td>20</td>
<td>40.00</td>
</tr>
<tr>
<td>BUFG</td>
<td>25</td>
<td>480</td>
<td>5.21</td>
</tr>
<tr>
<td>MMCM</td>
<td>1</td>
<td>10</td>
<td>10.00</td>
</tr>
</tbody>
</table>

**Modules and Loops**

- processNetwork
- process_0_s
- process_3
- prefilt
- prefilt Pipeline conv2d window 50

<table>
<thead>
<tr>
<th>Type</th>
<th>Issue</th>
<th>Slack</th>
<th>Latency (cycles)</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0.17</td>
<td>1023</td>
<td>5.115e+03</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0.17</td>
<td>1007</td>
<td>5.055e+03</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0.17</td>
<td>1066</td>
<td>5.830e+03</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1.97</td>
<td>125</td>
<td>625.000</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>3.15</td>
<td>35</td>
<td>175.000</td>
<td></td>
</tr>
</tbody>
</table>
SNL Model Example: MLP-MNIST

NW Definition and Area Consumption for KCU105

- Parameters = 42310
- Latency = 3212 cycles
  ~16.06us
Further Testing with SNL: Power Comparison

• We went further and did power comparison.
• Table presents the power measurements obtained from actual KCU105 board for both CNN and MLP examples.

**Observations:**
• There is a clear difference between power estimated by Vivado and the actual power measured on HW.
• Specifically, Vivado’s power estimate for the Tiny CNN network is differs by 1.4645W when compared to the physical hardware measurement.
• Similarly, Vivado’s power estimate for the MLP network differs by 1.0645W.
• For the MLP NW consumes the higher power, because it has a higher no. parameters necessitates increased memory mapping and more extensive computation between memory and logic cells, thereby leading to escalated power usage.

<table>
<thead>
<tr>
<th>Measurement Type</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny CNN</td>
<td></td>
</tr>
<tr>
<td>HW Excluding NN IP</td>
<td>2.2745</td>
</tr>
<tr>
<td>HW Including NN IP</td>
<td>2.5025</td>
</tr>
<tr>
<td>Vivado Excluding NN IP</td>
<td>3.289</td>
</tr>
<tr>
<td>Vivado Including NN IP</td>
<td>3.967</td>
</tr>
<tr>
<td>MLP</td>
<td></td>
</tr>
<tr>
<td>HW Excluding NN IP</td>
<td>2.2745</td>
</tr>
<tr>
<td>HW Including NN IP</td>
<td>2.6545</td>
</tr>
<tr>
<td>Vivado Excluding NN IP</td>
<td>3.294</td>
</tr>
<tr>
<td>Vivado Including NN IP</td>
<td>3.719</td>
</tr>
</tbody>
</table>
Challenges...

- How can we prune the model without having resynthesize the entire design?
- To target transformer style huge networks on FPGA is a complex challenging task given the FPGA resource constraints.
- It requires **Domain scientists and FPGA designers** to work closely together to optimize design for FPGA deployment.
- It also requires a significant modification in transformer architecture and training processes.