An FPGA-Based Solution for a Graph Neural Network Accelerator (WP024)

Achronix Data Acceleration

February 18, 2021 **White Paper**

Thanks to the rise of big data and the rapid increase in computing power, machine learning technology has experienced revolutionary development in recent years. Machine learning tasks such as image classification, speech recognition, and natural language processing, operate on Euclidean data with a certain size, dimension, and an orderly arrangement. However, in many realistic scenarios, data is represented by complex non-Euclidean data such as graphs. These graphs not only contains data, but also the dependencies between data, such as social networks, protein molecular structure, e-commerce platform customer data, and so on. The increase in data complexity poses a severe challenge to traditional machine learning algorithm design and its implementation technology. In this context, many new graph-based machine learning algorithm, or graph neural networks (GNNs), are constantly emerging in academia and industry.

GNN has very high requirements for computing power and memory, and the software implementation of its algorithm is very inefficient. As a result, the industry has a very urgent need for hardware-based GNN acceleration. While traditional convolutional neural network (CNN) hardware acceleration has many solutions, the hardware acceleration of GNN has not been fully discussed and researched. At the time of writing this white paper, Google and Baidu were unable to search Chinese research on GNN hardware acceleration. The motivation for this white paper is to combine the latest foreign GNN algorithm, acceleration technology research and a discussion of GNN FPGA-based acceleration technology, and present it to readers in the form of a overview.

Introduction to GNN

The architecture of GNN has many similarities to traditional CNN at a macro level, such as convolutional layer, polling, activation function, MLP, FC layer and other modules — all of which can be applied to GNN. The figure below shows a relatively simple GNN architecture.

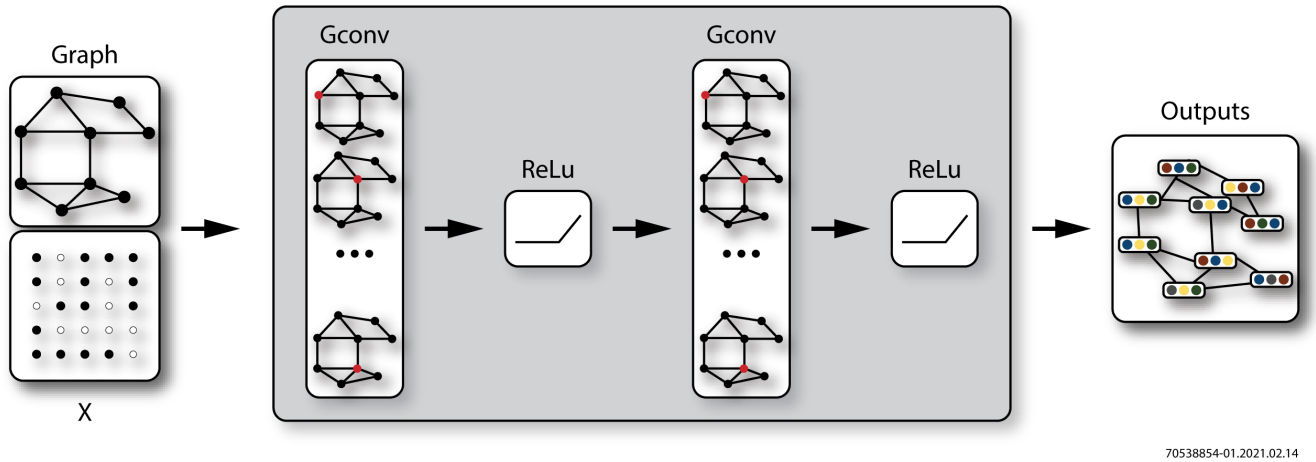

Figure 1: ***Typical GNN Architecture (Source:*** *https://arxiv.org/abs/1901.00596**)***

However, the graph data convolution calculation in GNN is different from the 2D convolution calculation in traditional CNN. Taking the figure below as an example, the process of convolution calculation for the red target node is as follows:

1. Graph convolution – Use the neighbor function to sample the features of surrounding nodes and calculate the average value. The number of neighboring nodes is uncertain and disordered (non-Euclidean data)
2. 2D convolution – Use the convolution kernel to sample the features of the surrounding nodes and calculate the weighted average. The number of neighboring nodes is determined and ordered (Euclidean data)

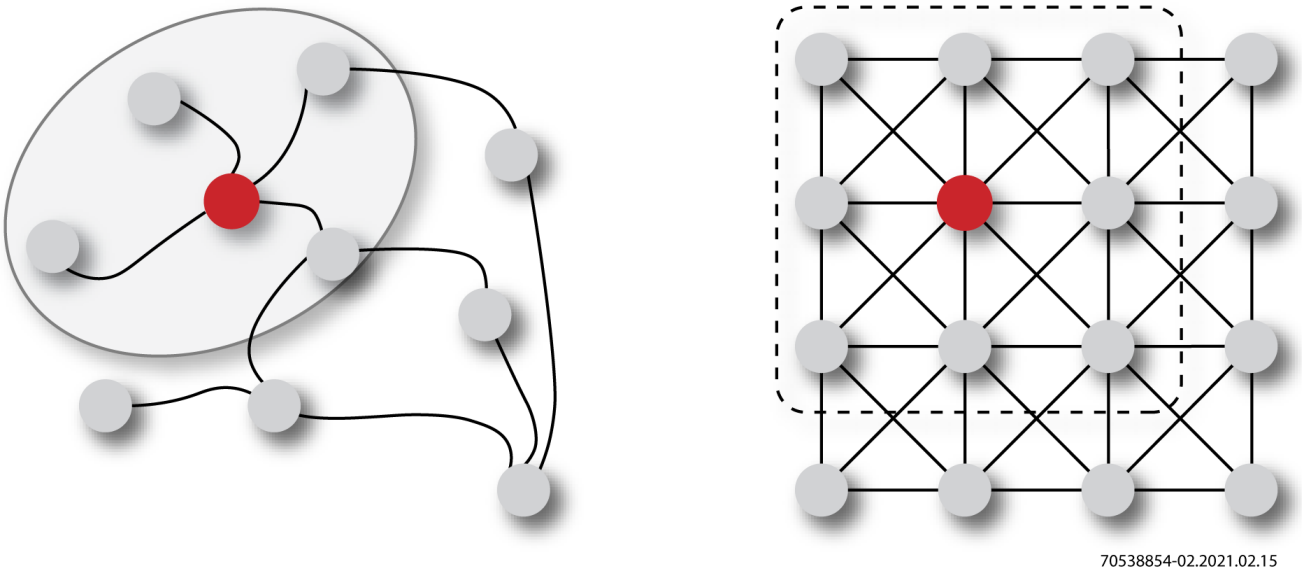

Figure 2: ***Graph and 2D Convolution (Source:*** *https://arxiv.org/abs/1901.00596****)***

Introduction to GraphSAGE Algorithm

Academia has conducted a lot of research and discussion on the GNN algorithm and proposed a considerable number of innovative implementation methods. Among them, GraphSAGE, proposed by Stanford University in 2017, is an inductive representation learning algorithm for predicting dynamic, new, unknown node types in large graphs, especially optimized for graphs with a huge number of nodes and rich node features. As shown in the figure below, the GraphSAGE calculation process can be divided into three main steps:

1. Adjacent node sampling – Used to reduce complexity, generally sampling two layers, each layer sampling several nodes.
2. Aggregation – Used to embed the target node,, i.e. the low-dimensional vector representation of the graph.
3. Prediction – Uses embedding as the input to the fully connected layer to predict the label of the target node d.

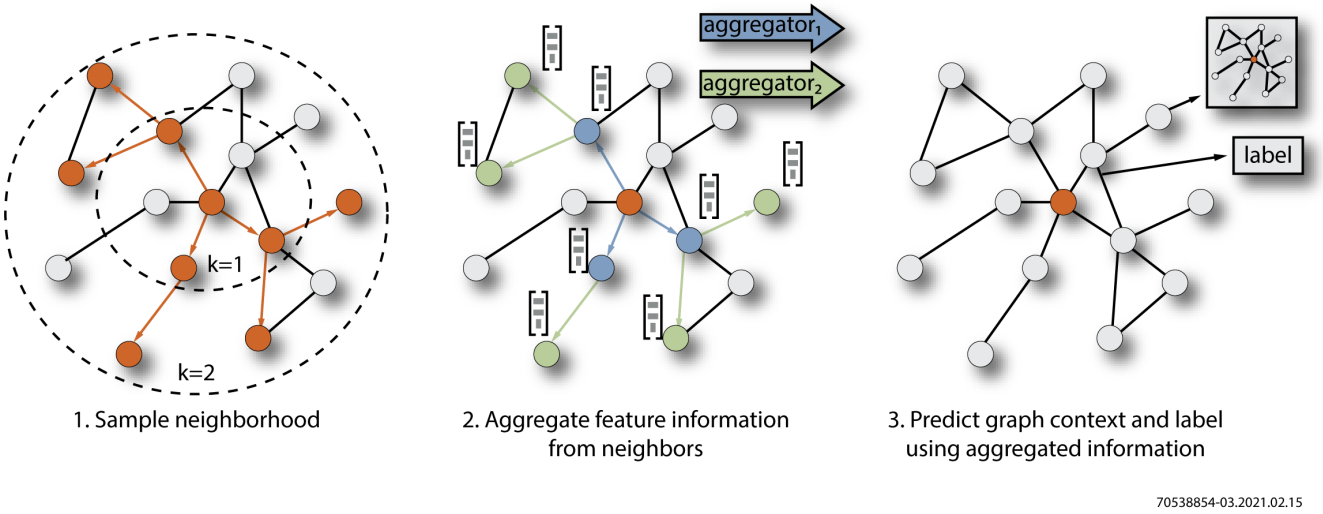

Figure 3: ***Visual Representation of the GraphSAGE Algorithm (Source:*** *http://snap.stanford.edu/graphsage****)***

In order to realize GraphSAGE algorithm acceleration in FPGA, its mathematical model must be understood in order to map the algorithm to different logic modules. The code shown in the figure below illustrates the mathematical process of this algorithm.

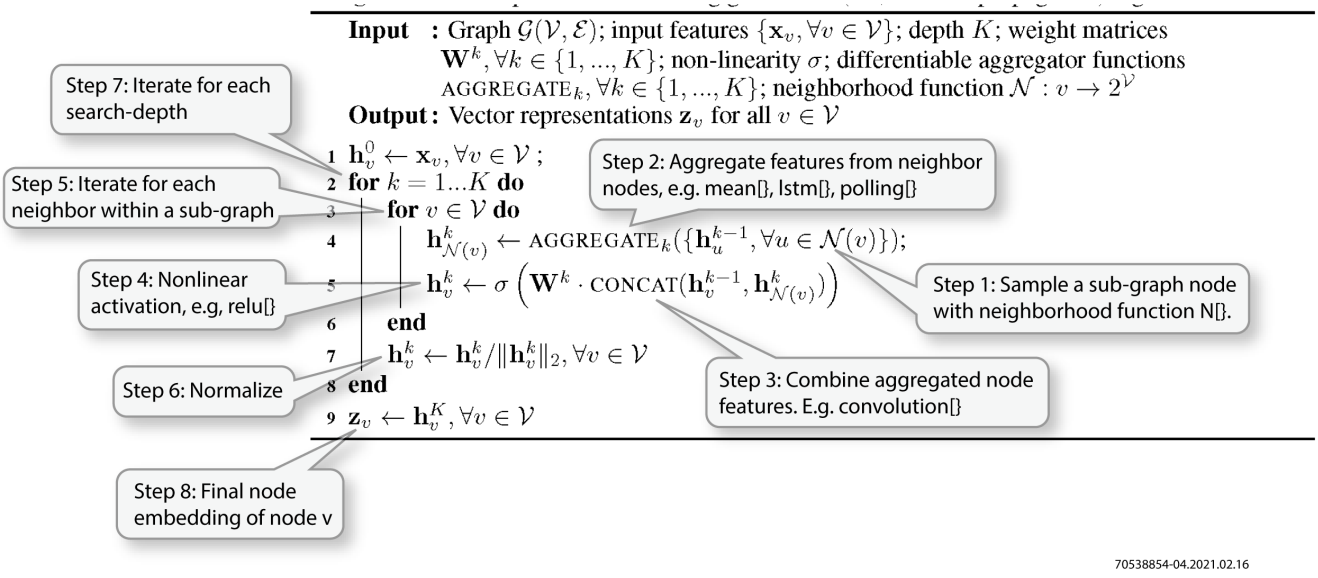

Figure 4: ***Mathematical Model of GraphSAGE Algorithm (Source:*** *http://snap.stanford.edu/graphsage****)***

For each target node xv to be processed, GraphSAGE performs the following operations:

1. Samples the nodes in the subgraph through the neighbor sampling function N(v).
2. Aggregates the features of the neighboring nodes to be sampled. The aggregation function can be mean(), lstm(), or polling(), etc.
3. Combines the aggregation result with the output representation of the previous iteration, and uses Wk for convolution.
4. Performs non-linear processing of convolution results.

5. Iterates several times to end the processing of all neighboring nodes of the current k-th layer.
6. Normalizes the results of the k-th layer iteration.
7. Iterates several times to end the processing of all K-layer sampling depths.
8. Embeds the final iteration result zv as the input node xv.

GNN Accelerator Design Challenges

The GNN algorithm involves a large number of matrix calculations and memory access operations. It is very inefficient to run this algorithm on a traditional x86 architecture server, which is manifested in slow speed and high energy consumption.

The application of new GPUs can bring significant benefits to the computing speed versus energy efficiency ratio of GNN. However, the shortcomings of GPU memory scalability make it incapable of processing the massive number of nodes of a graph. A GPU's instruction execution method also causes a calculation delay to be too large and uncertain; therefore, it is not suitable for scenarios that require real-time calculation of a graph.

The various design challenges mentioned above makes the industry urgently need a GNN acceleration solution that can support highly concurrent, real-time computing, huge memory capacity and bandwidth, scalableto the data center.

FPGA Design Scheme of GNN Accelerator

Achronix's Speedster®7t FPGA family (and the first device, AC7t1500) of high-performance FPGAs are optimized for data-center and machine learning workloads, eliminating several performance bottlenecks found in solutions based on CPUs, GPUs, and traditional FPGAs. Speedster7t FPGA family is based on TSMC's 7nm FinFET process, and its architecture uses a revolutionary new 2D network on chip (NoC), an original machine learning processor matrix (MLP), and utilizes a high-bandwidth GDDR6 controller, 400G Ethernet and PCI Express Gen5 interfaces , While ensuring ASIC-level performance, it provides users with flexible hardware programmability. The following figure shows the architecture of the Speedster7t1500 high-performance FPGA..

Figure 5: ***Achronix Speedster AC7t1500 High-Performance FPGA Architecture***

The features mentioned above make the Achronix Speedster7t1500 a perfect solution for the various challenges faced in the design of GNN accelerators.

Table 1: ***GNN Design Challenge and the Achronix Speedster7t1500 FPGA Solution***

GNN Design Challenges	Speedster AC7t1500 Solution
High-speed matrix operations	Machine learning processor (MLP).
High-bandwidth and low-latency storage	LRAM+BRAM+GDDR6+DDR4.
High-concurrency and low-latency calculation	An FPGA uses programmable logic circuits to ensure low- and high-concurrent delay calculations at the hardware level.
Memory expansion	Based on 4 × 400 Gbps RDMA ensures expansion of memory access with very low latency in the data center.
Algorithms continue to evolve	The programmable logic in an FPGA ensures that the algorithm can be upgraded and reconfigured at the hardware level.
Complex design	Abundant hard IP reduces development time and complexity, NoC simplifies interconnection between modules and improves timing

GNN Accelerator Top-Level Architecture

This GNN accelerator is architected for GraphSAGE, but its design can be applied to other similar GNN algorithm acceleration. Its top-level architecture is shown in the figure below.

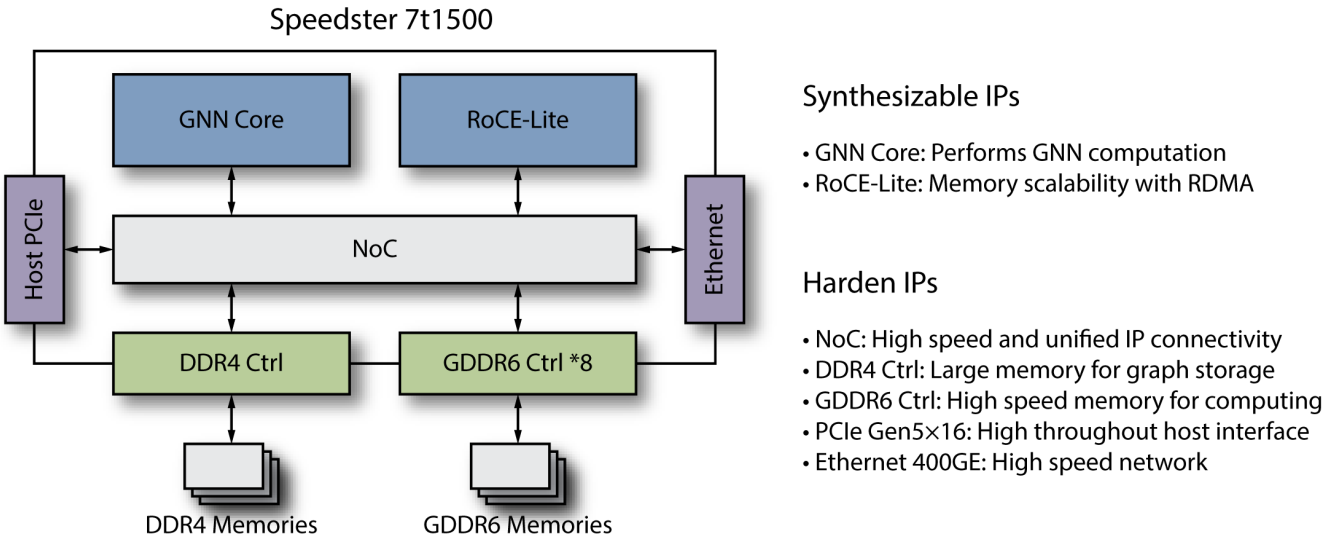

Figure 6: ***GNN Accelerator Top-Level Architecture***

The architecture consists of the following blocks:

- A GNN core in the figure is the central part of the algorithm implementation (details below).
- A RoCE-Lite is a lightweight version of the RDMA protocol, used for remote memory access via high-speed Ethernet to support graph computing for massive nodes.

- A 400GE Ethernet controller is used to carry the RoCE-Lite protocol.
- GDDR6 memory is used to store the high-speed access data required for the GNN processing process (DDR4 is used as a spare high-capacity memory). This memory is used to store relatively low frequency data, such as graph data to be pre-processed.
- A PCIe Gen5 ×16 interface provides a high-speed host interface for data exchange with server software.

All of the above modules are interconnected through the high-bandwidth NoC.

GNN Core Microarchitecture

Before starting to discuss the GNN core's microarchitecture, a review of the GraphSAGE algorithm is in order. The aggregation and merging (including convolution) of the inner loop account for most of the calculation and memory access of the algorithm. Through research, we get the characteristics of these two steps as follows.

Table 2: *Comparison of Aggregation and Merging Operations in the GNN Algorithm (source: https://arxiv.org/abs/1908.10834)*

Step	Aggregation Operation	Merge Operation
Memory access mode	Indirect access, irregular	Direct access, rules
Data reuse	Low	High
Calculation mode	Dynamic, irregular	Static, rule
Calculation	Low	High
Performance bottleneck	Storage	Calculation

It can be seen that the aggregation operation and the merge operation have completely different requirements from the calculation and memory access. The aggregation operation involves sampling of neighboring nodes. However, a graph is a non-Euclidean data type — its size and dimension are uncertain and disorderly, the matrix is sparse, and the node position is random. Therefore, the memory access is irregular and makes it difficult to reuse data.

In the merge operation, the input data is the aggregation result (low-dimensional representation of the node) and the weight matrix. Its size and dimension are fixed, with a linear storage location. There is no challenge to memory access, but the calculation of the matrix is very large.

Based on the above analysis, a choice was made to use two different hardware structures in the GNN core accelerator design to handle aggregation and merging operations separately (figure below):

- Aggregator – Through the single-instruction. multiple-data (SIMD) processor array, graph neighboring nodes are sampled and aggregated. The single instruction can be predefined as mean() mean value calculation, or other applicable aggregation functions; multiple data means that a single mean() mean value calculation requires the feature data of multiple neighboring nodes as input, and this data comes from the subgraph sampler. The SIMD processor array is load balanced through the scheduler, Agg Scheduler. The adjacency matrix and node characteristic data h0v read back from GDDR6 or DDR4 by the subgraph sampler through NoC are cached in the Adjacent List Buffer and the Node Feature Buffer, respectively. The result of aggregation hkN(v) is stored in Aggregation Buffer.
- Combinator – Perform the convolution operation of the aggregation result through the pulsation matrix PE. The convolution kernel is the Wk weight matrix. The convolution result is non-linearly processed by the ReLU activation function, and it is also stored in the Partial Sum Buffer for the next Round iteration.

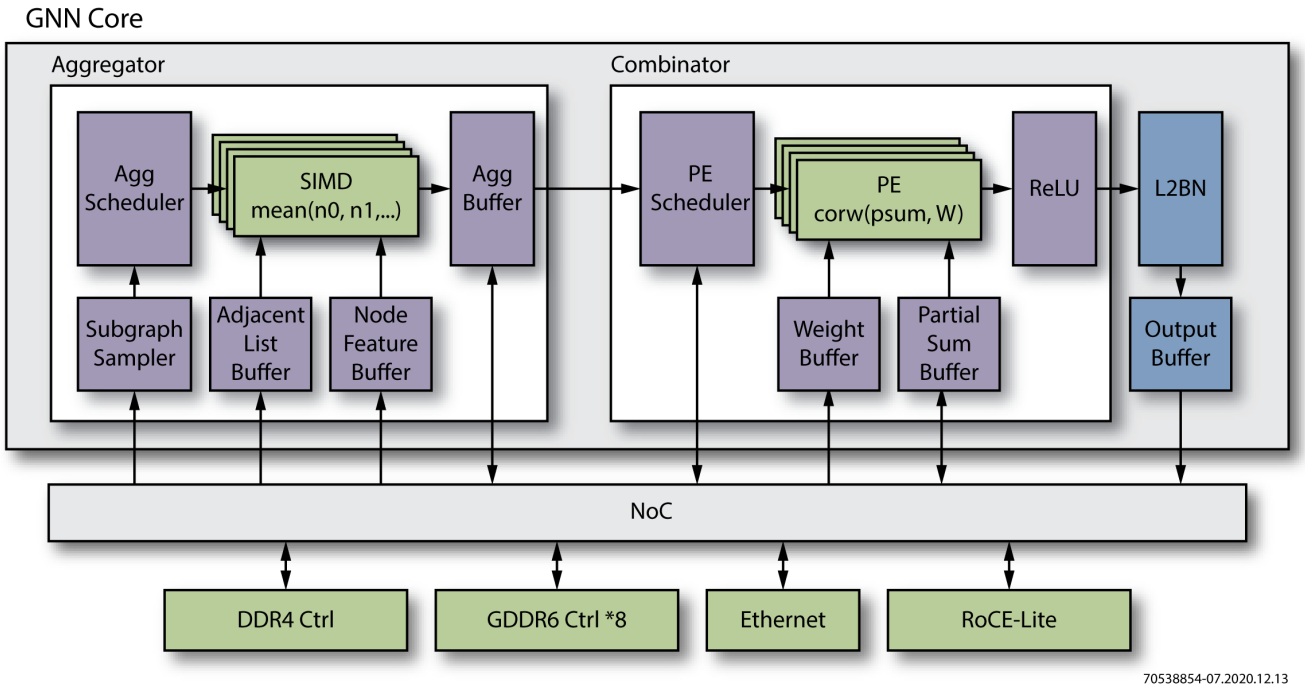

Figure 7: ***GNN Core Functional Block Diagram***

After the merged result is normalized by L2BN, it is the final node representation hkv. In a typical node classification prediction application, the node representation hkv can pass a fully connected layer (FC) to obtain the node's classification label. This process is one of the traditional machine learning processing methods, which is not reflected in the GraphSAGE paper, and this function is not included in this architecture.

Conclusion

This white paper discusses the mathematical principles of the GraphSAGE GNN algorithm, and analyzes the technical challenges of a GNN accelerator design from multiple angles. By analyzing the problems and solving them one by one at the architecture level, an architecture is proposed that uses the competitive advantages provided by the Achronix Speedster7t AC7t1500 FPGA to create a highly scalable GNN acceleration solution that can deliver excellent performance.

For more information about the Speedster7t FPGA family, visit www.achronix.com.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Notice of Disclaimer