Reconfigurable Computing and LLC markets

Rahul Razdan

Previous articles have introduced the idea of LLC markets, their issues in the current electronic ecosystem, and their importance relative to future mega-trends such as AI/IOT. The fundamental issue is the intense churn of the supply chain generated by the consumer dominated semiconductor marketplace for LLC markets such as Aerospace and Defense. The churn certainly causes issues of reliability, obsolescence, and many cases the need for managing future requirements changes.

Meanwhile, in an entirely different technical universe, reconfigurable computing has been percolating since the mid 1990s. The fundamental notion of reconfigurable computing is to dynamically build hardware based on the specifics of the application with the intention of optimizing performance or power. Reconfigurable computing got its start with the PRISM systems which offered a board-level programmable interface. This interface had the advantage of performance, but the disadvantage of usability. In 1996, the PRISC architecture was introduced with a programmable instruction model with an integrated compiler flow. This had the great advantage of simplicity, but limited performance due to the 2-input/1-output operand format. Finally, GARP, a coprocessor model similar to graphics co-processors, extended the model. Reconfigurable computing has seen success in networking and cryptography applications. Also, many of the embedded process applications range from MicroBlaze to Tensilica. Perhaps the most direct usage of these concepts is in the RISC-V open source project.

What does all of this have to do with LLC supply chain issues ? That link was made in a keynote talk at FCCM 2019. The critical observation was that the massive machinery of reconfiguration can be used not just for performance/power optimization, but also for addressing issues of semiconductor obsolescence, reliability, and absorbing future requirements volatility.

In this architecture, the fabric is composed of a large number of programmable parts (cpus, fpgas, neural networks/analog signal chains). The only real issue becomes how much to commit into the final implementation. The amount to be committed is a function of the expected lifetime of the system design and the expected reliability of the underlying parts. Let us consider how one addresses critical LLC issues:

1. **Obsolescence:** The mapping of system function to programmable parts is dynamic. In general, programmable parts have a much lower rate of obsolescence. Xilinix claims to have never EOLed an FPGA. Further, if that ever happens, a suitable replacement can

be made in the fabric and the function recompiled. Finally, one can even use a large stocking strategy because the same programmable parts are reused in a large number of applications.

2. **Reliability**: If the basic components are built for the consumer marketplace, they will have a component reliability designed for five years. With a programmable fabric, one can gain natural redundancy and dynamically move function upon detection of fault.
3. **Future Requirements**: For a product which is going to last 30 years, invariably there will be a requirement to update the core function and having a programmable fabric with capacity is one way to add the function without a rip-up and replace cycle. This is especially interesting in applications such as smart transportation or cities.

A very interesting byproduct of this methodology is **security**. Today, due to the manufacturing of large amounts of hardware outside of the United states, security in the form of discovery of system function and even worst back-door viruses are a concern. In the world of reconfigurable computing, both of these issues can be addressed in a positive direction for three reasons:

1. **Unique Part Count:** Since the number of unique parts is limited, their security characteristics can be monitored very closely.
2. **Programmable Location:** Back doors or other snooping methods can be minimized by dynamically moving function in the programmable fabric.
3. **System Function Isolation:** In this world, system functionality never leaves protected conditions. Even the final device can be built with failsafes which erase the system programming.

The critical capability which pulls all of this together is software (largely EDA in nature) in combination with runtime maintenance operating systems. The core concepts are not too dissimilar to the NASA design methodologies for custom highly redundant programmable space systems, but the required EDA capabilities are not yet available in the commercial marketplace.