Leveraging Programmability in Electronic System-Level Designs

You gain productivity and accuracy with platform-based design and programmable platforms.

by Douglas Densmore
Graduate Student Researcher
University of California, Berkeley
densmore@eecs.berkeley.edu

Alberto Sangiovanni-Vincentelli
Professor
University of California, Berkeley
alberto@eecs.berkeley.edu

Adam Donlin
Research Engineer
Xilinx Research Labs
adam.donlin@xilinx.com

Platform-based design (PBD) is an electronic system-level (ESL) design methodology forged in the inferno of ASIC design complexity: it tackles shrinking design time pressures, the growing complexity of applications, nanometer-era design effects, and heterogeneity between chip and system components. Individually, the challenges are formidable. Combined, they are almost insurmountable: the methods ASIC designers use to address one challenge are often at odds with the methods that solve another. Programmable platforms can utilize PBD and be a welcome new home to refugees fleeing ASIC design complexity.

In this article, we'll show you how to effectively combine the benefits of FPGAs with lessons learned from ASIC design. You will see how the unique properties of programmable platforms leverage innovative techniques to make PBD a powerful FPGA design methodology. We will demonstrate how programmable platforms and PBD allow designers to build complex systems not just faster, but also of a higher quality than non-programmable solutions using traditional design techniques.

Platform-Based Design

At its core, PBD (Figure 1) attempts to:

- Maximize the level of design re-use available to system architects and chip designers
- Enable design verification, analysis, and synthesis at all levels of abstraction
- Provide the basis for design chain integration
- Make architectural exploration, including implementation platform selection, a fast and rigorous process

The basic tenets of the methodology are:

1. Orthogonalization of concerns. Function (what the system does), architecture (how it does it), communication, and computation are modeled independently with a unified mathematical framework.
2. Design formalization. Incorporate a mathematical framework with mapping of functional components to architectural elements. This process can be made (partially) automatic by providing common semantics between two consecutive layers of abstraction, similar to what was achieved when logic synthesis was introduced in the traditional design flow.
3. Successive refinement. The design progresses through a refinement process that links the levels of abstraction defined with the help of the design formalization.

The structure of the platform-based design methodology is illustrated in Figure 1. However, these techniques are severely limited if, during the development of systems with PBD, the results of simulation to predict performance before system creation are inaccurate. Thus, a critical element of PBD is the characterization of architectural components in terms of physical quantities such as time, power, and size. The choice to map functional components to architectural elements is guided by constraints and cost functions expressed in terms of these quantities. To evaluate the quantities for the entire design in terms of the quantities associated with the components, you would use simulation.

Architectural components at high levels of abstraction are characterized in terms of these quantities (time, power, and size), also through estimation or abstraction. If the abstraction or estimation is inaccurate, the results of simulation to predict performance before system creation are inaccurate.

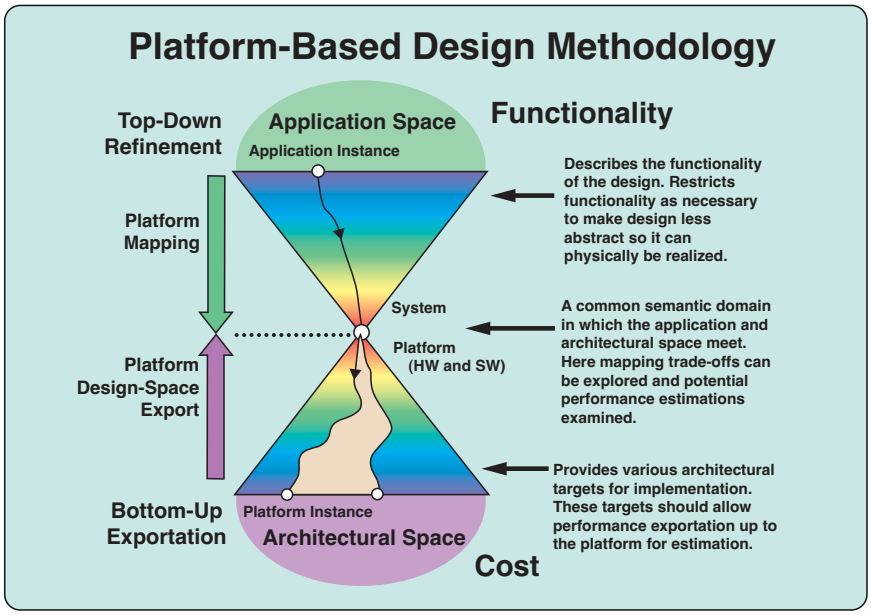

Figure 1 – Platform-based design

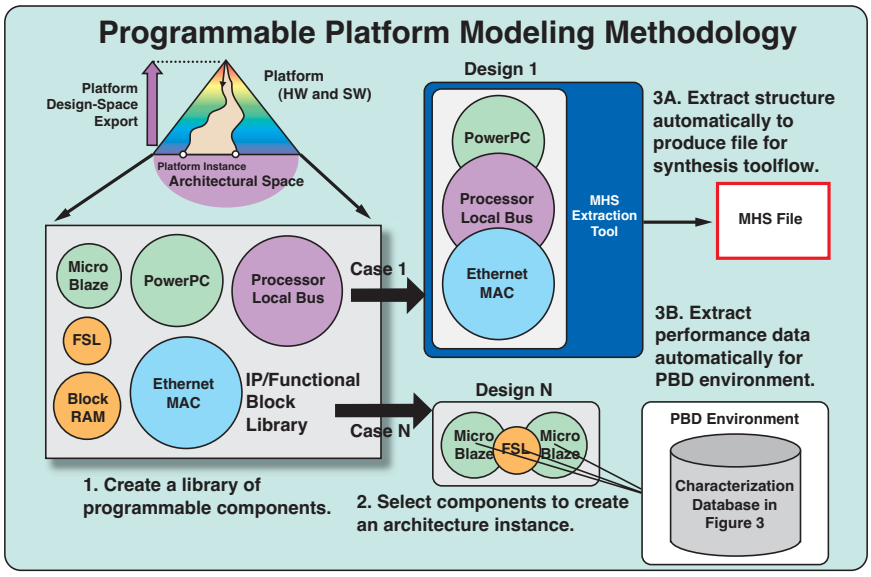

Figure 2 – Programmable platform modeling

Designers require performance models that are abstract and modular while still having a close correlation (accuracy) to actual devices. In addition, these models need to be convenient to use and efficient to simulate. Although this is certainly possible for all design styles and is one of the essential values of the methodology, it is easier to carry out if the implementation platform has fewer degrees of freedom.

Programmable platform FPGAs such as the Xilinx® Virtex™-4 device are excellent substrates for PBD: they are easily reconfigurable using the FPGA fabric and contain powerful high-performance pre-designed processors whose performance can be accurately predicted before simulation.

Programmable Platform Modeling

In our approach, a platform is a collection of IPs that you can use to implement your designs. Each IP has to be given a functional model (what it can do) and a performance model (the cost of doing it). In the Virtex-4 platform, a functional model of an element corresponds to a number of different implementations regarding actual selection of the basic building blocks. Each of these implementations is characterized by a different performance model.

Functional Modeling

The architectural space in Figure 2 contains functional models of FPGA blocks that you can combine to form a programmable system. The key requirement of the models is abstraction to increase designer productivity; however, they must still remain efficient in their ability to yield a good design.

There are three aspects of programmable platforms that facilitate this task as compared to other design styles (such as ASICs):

1. Programmable platforms allow one device to represent many possible architectural topologies. Therefore, one model set for a device allows many potential designs tailored to various levels of concurrency and application specificity.
2. Programmable platforms are naturally partitioned into functional or IP-based blocks. This is the case with the Xilinx implementation of the IBM CoreConnect system in EDK. You can create models at this granularity with a transaction-level set of models.
3. Programmable models are a collection of configured elements. This configuration can be explicitly captured by the models and provided directly to the tool flow. In the case of Xilinx FPGAs, this could be the microprocessor hardware specification (MHS) file.

Figure 2 shows an example of the entire flow. Notice that these three techniques increase efficiency by facilitating library creation, transaction-level modeling, and design synthesis, all while maintaining the required abstraction level.

Performance Models

Accurate programmable platform performance models are easier to obtain than other implementation styles such as ASICs because the physical characteristics of the implementation fabric are known at design time. Figure 3 illustrates a flow based on a pre-characterization process.

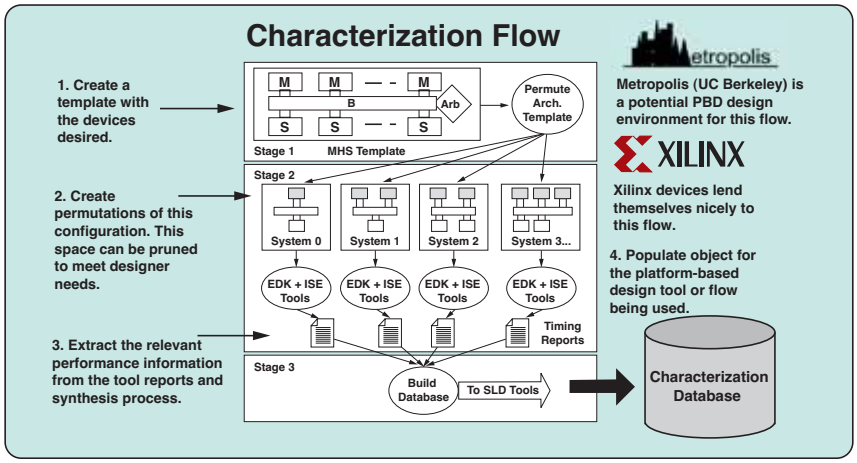

Figure 3 – Programmable platform characterization

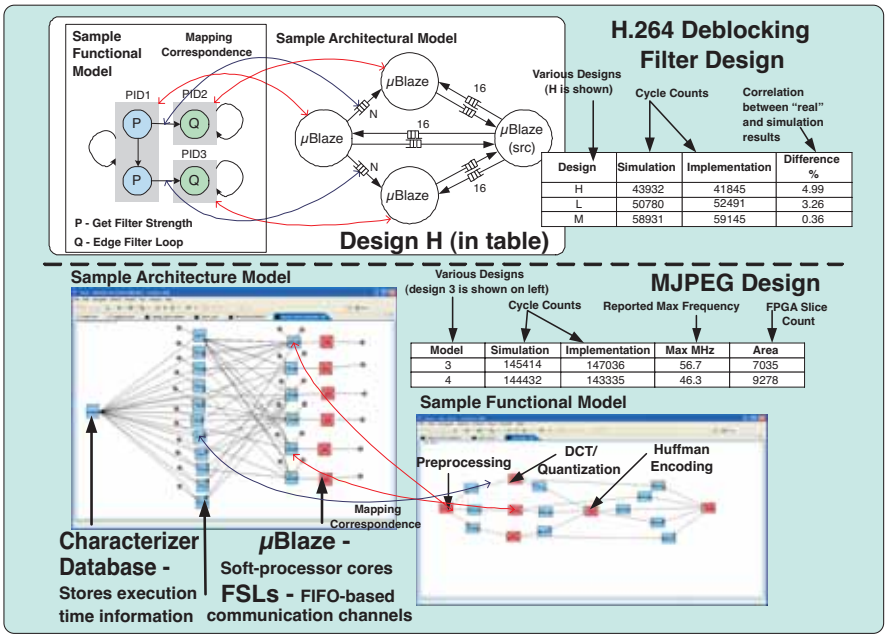

Figure 4 – Example methodology results

- Permutations of various architecture topologies are created in the first stages. These permutations result from an initial "seed" topology tailored to your specific needs.
- Permutation generation is followed by actual synthesis, place and route of these designs for the device being characterized. You use the exact tool flow that the device will eventually use for programming. This is not easily available in an ASIC flow.
- The data from synthesis, place and route is captured in a database structure that can be used by a tool supporting PBD during simulation. Instead of relying on static estimations, simulation can annotate transactions directly with costs that have been realized on an actual system corresponding to the simulation. This database is independent of the actual system, thus yielding a great deal of modularity. You can reuse the database for other designs or add to it on an independent, individual basis. And because the models are created to have a very specific topology, you can use the topology information to index the database during simulation.

Example Designs Using this Approach

We have two example designs using our approach: a motion JPEG encoder and an H.264 video standard deblocking filter. Both of these designs were first specified in the Metropolis Design Environment from the University of California, Berkeley. (Metropolis is a design environment that fully supports the PBD methodology.)

The Metropolis models were of Xilinx Virtex-II Pro CoreConnect components. Specifically, these designs were of MicroBlaze™ and Fast Simplex Link (FSL) networks. Figure 4 shows the designs and a sample of the results we obtained. Key insights include:

- The accuracy of simulation versus actual implementation performance is very high. In the cases shown, the worst case is a mere 5% difference.
- Characterization allowed us to observe the actual system clock speed and design area values. This is important because not only does clock speed combine with the clock cycle requirements to form an actual execution time, but it also demonstrates the potentially detrimental effect of increasing the size of the design. This effect would be more difficult to obtain quickly in ASIC-based design flows.

Conclusion

Platform-based design is an ideal methodology to convert system design ideas into implementation realities, especially when coupled with pre-characterized programmable platforms. High-level abstract and modular models supported by the methodology allow fast, accurate, and efficient design space exploration and early validation.

The link to implementation offered by a pre-design fabric, as offered by the Xilinx Virtex family, yields rapid design turns. You can accurately predict the performance of your designs during the early design phases, further reducing the need to re-design.

We believe that the fast and efficient mapping of systems into programmable platforms offered by PBD can ease the pain that today's designers must endure to carry out their tasks. For more information about PBD and supporting tools, see *http://embedded.eecs.berkeley.edu/metropolis/index.html.*