A Methodology for FPGA to Structured-ASIC Synthesis and Verification

Mike Hutton, Richard Yuan, Jay Schleicher,
Gregg Baeckler, Sammy Cheung
Altera Corp. San Jose, CA, USA
{mhutton,ryuan}@altera.com

Kar Keng Chua, Hee Kong Phoon
Altera Corp. Penang, Malaysia

Abstract

Structured-ASIC design provides a mid-way point between FPGA and cell-based ASIC design for performance, area and power, but suffers from the same increasing verification burden associated with cell-based design. In this paper we address the verification issue with a methodology and fabric to directly tie FPGA prototype and functional in-system verification with a clean migration path to structured ASIC. The most important aspects of this methodology are the use of physically identical blocks for difficult-to-verify PLLs, I/O and RAM and a structured re-synthesis of FPGA logic blocks to target cells that guarantees anchor points for easy formal verification.

1. Introduction

Today's hardware designers are faced with difficult decisions arising from conflicting efficiency and time-to-market pressures. Cell-based ASICs offer the best density, performance and power but have long design times, high NRE costs, and increasingly difficult verification cycles. FPGAs offer zero NRE-cost but higher unit cost and poorer density, performance and power compared to ASIC.

A less-often cited benefit of FPGAs is methodology: Because physical issues, timing and power, SI, etc. are performed by the vendor and amortized across thousands of designs, the designer performs only functional verification and timing analysis, avoiding not just risk but many expensive tools. Most FPGA designers use re-programmability as an integral part of their test methodology and perform significant amounts of their test in-system. A mask re-spin to an ASIC designer is analogous to a software iteration for an FPGA designer.

Recently, structured ASICs or "neo-gate-arrays" have been proposed as a hardware model that bridges the gap between ASIC efficiency and FPGA flexibility. A structured ASIC consists of a base array of hard blocks (e.g. I/O and RAM) along with relatively simple logic structures in a regular fabric that is hard-wired for most processing layers, but can be targeted at a specific application by customizing only several processing steps (e.g. two to four metal or via layers).

The theoretical efficiency of structured-ASIC can be close to that of an ASIC while the NRE costs are minimized – at 90nm the NRE runs about 1/10 that of a full mask set for a cell-based ASIC. Recent keynotes and panel sessions [1-5] have dealt with the efficacy of structured-ASIC.

A number of architectural proposals exist. Patel [6] and Tong [7] described a via-programmed gate-array (VPGA), in which a fabric of LUT-like logic elements is programmed by via-connections between pre-defined routing layers. This is architecturally very close to a FPGA methodology in that the routing architecture mimics the logic-element and routing structures of generic FPGAs. Pileggi [8] discussed further tradeoffs in VPGA design, Kheterpal [9] explored the routing architecture issues, and Reed Taylor and Schmit [10][11] evaluated VPGA power consumption design and tradeoffs.

Hu [12] introduced the GLA or gain-based logic block array, which is also based on via-programming. The GLA logic cell consists of two NAND2 and AND3 and a NAND3 interconnected. They describe a design flow for synthesizing and technology mapping to a library of these 3 primitive gates, which are then packed into the base cell, and then a placement algorithm based on VPR [13]. Ran and Marek-Sadowska [14] examine transistor-level implementations of cell blocks for VPGAs and a design flow for via-programmable ports between fixed metal layers M1 and M2 (on a 2-metal process). Shenoy, Kawa and Camposano [15] further investigate mask-programmable fabrics and CAD tools.

Most studies to-date on structured ASIC concentrate only on the logic fabric. Much of the silicon area for both FPGAs and structured ASIC base arrays, however, is taken up by RAM and I/O buffers supporting high-speed serial and CDR transceivers, numerous I/O standards and DDR interfaces. (See Figure 1.) These blocks are non-trivial to design and characterize Further, the preferred design flow for many designers is to prototype and ship low-volume on FPGAs then migrate to an ASIC should volumes dictate. Conversion requires re-design, qualification, and complete re-verification.

A number of companies manufacture structured ASICs, including Altera, AMI, ChipExpress, eASIC, Faraday [16], Fujitsu, LSI, NEC [17] and Virage [18].

3-9810801-0-6/DATE06 © 2006 EDAA

We propose a structured ASIC fabric and design methodology closely tying the ease of design and verification to that of FPGAs, without sacrificing efficiency. Our thesis is that verification and DSM design issues will force the majority of these designs to be conversions from FPGA prototypes, and thus the key methodology is in facilitating this flow. We argue that hard blocks and clocking need to be similar or even identical between FPGA and ASIC, a seamless conversion of generic logic with common register boundaries needs to be maintained to ensure ease of verification, and a common tool flow must integrate the two design styles.

Figure 1 illustrates the correspondence between an FPGA floorplan (top) and a compatible structured ASIC base array (bottom right). In the proposed methodology there is a 1:1 layout-level correspondence between RAM, PLL, transceiver and I/O blocks, while soft-logic DSP multipliers and logic cell fabric of the FPGA are re-synthesized to the structured ASIC fabric. The key elements of the paper are the design and software flow for this re-synthesis and the tie-in to verification.

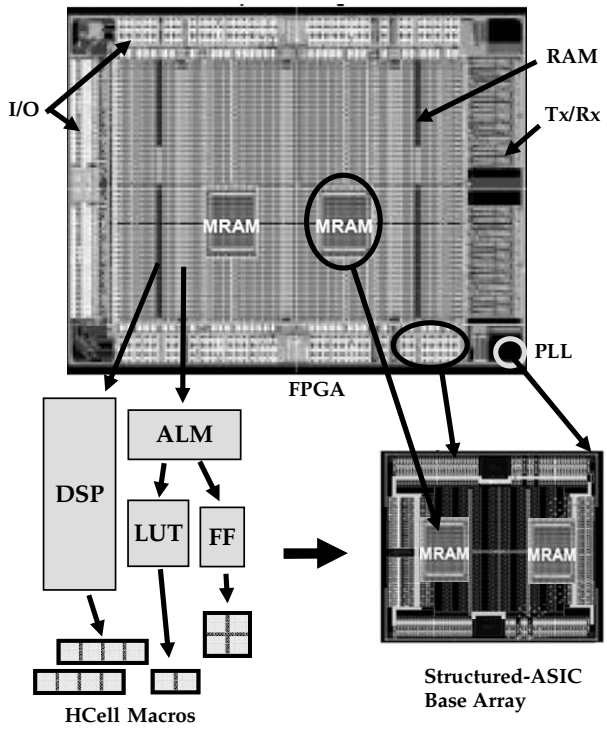

Figure 1. FPGA/ASIC Correspondence

The remainder of the text is organized as follows. In Section 2 we introduce the FPGA logic cell and structured-ASIC HCell and HCell macro and base array which are the targets of our combined synthesis flow. Section 3 describes macro libraries and the synthesis flow for converting logic based on a commercial FPGA into this base array. Section 4 shows the verification process for showing equivalence of an FPGA prototype to its corresponding ASIC device, and we conclude in Section 5.

2. FPGA Logic Cell, HCell and HCell-Macros

SRAM-based FPGA logic cells normally consist of k-input LUTs and flip-flops. The basic functionality of a logic cell is to implement k-input combinational functions and optionally register the function's output. For example, Figure 2 shows the basic logic unit in Altera's Stratix II device [19], [20], called an ALM ("Adaptive Logic Module"). An ALM can be configured to be a 6-LUT or two 5-LUT or smaller logic cells through appropriate LUT-mask programming, and contains by-passable register elements. We will use this ALM as an example, but the underlying methodology is applicable to other 4-LUT architectures such as Xilinx Virtex or Altera Stratix.

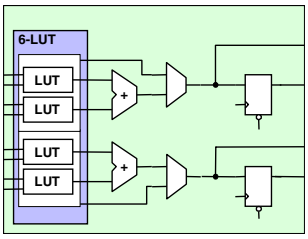

Figure 2. Simplified FPGA Logic Cell

We will use the term HCell to represent the fine-grained basic logic unit which forms the logic fabric in the structured ASIC base array. It is similar to the FPGA logic cell in the sense that the fabric consists of a regular pattern formed by tiling one or more basic cells in a two-dimensional array. The difference is that the HCell has no overhead for configuration and, for the examples used in this paper, is assumed to be finer granularity.

A wide range of HCell candidates can be explored, from fine-grained NAND gates to multiplexers to coarse-grained LUTs. Samples of HCell candidates are shown in Figure 3. In a structured fabric an array of such cells and general purpose routing connecting them are laid down on the lower layers of the chip. Specific layers then form via-connections or metal lines to customize the generic array into specific functionality. Figure 4 shows how logic function macros comprising multiple HCells can be placed into the regular fabric.

We use a library of HCell macros to represent pre-optimized, pre-characterized ASIC library cells. Each HCell macro consists of one or more HCells arranged in one or two dimensional arrays. A group of HCells together emulate a given FPGA combinational logic cell, DFF, or DSP (multiplier) block. The macro defines the number of HCells needed, the configuration of each HCell used, and the connectivity among these HCells. The configuration

and the connectivity is done via lower level layers to minimize the impact on routing resources and achieve better density. Figure 5 illustrates the HCell macro concept.

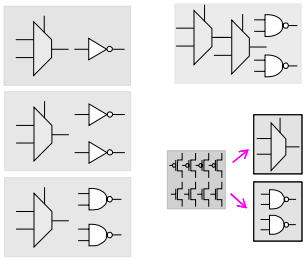

Figure 3. HCell Candidates

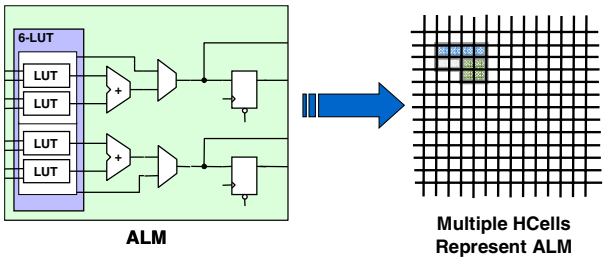

Figure 4 FPGA Logic Cells in HCell Fabric

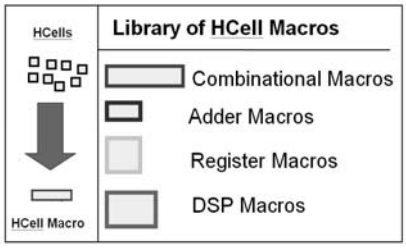

Figure 5 Sample HCell Macros

An important issue in the design of the HCell and HCell macros is the 1:1 correspondence between FPGA prototype and structured cells. Though it could be slightly more efficient to re-synthesize entire cones of logic and blocks into HCells, the benefits of maintaining all names and functional boundaries outweigh the area savings. Since the conversion of FPGA LEs and routing to HCells results in a 10:1 or more decrease in logic area the contribution to overall die area is dramatically smaller than I/O, RAM and other hard blocks in the base die. So, even a significant inefficiency in this structured conversion process will result in a negligible die-area cost.

2.1 Creating HCell Macros

Combinational HCell macros are created by re-synthesizing FPGA logic cells in terms of base HCells. For a given logic cell in the prototype FPGA netlist, optimization is done to compact the logic by removing parts of the FPGA logic cell that are not used for a given instance, thus reducing the area cost, the intra-logic cell delay, and power dissipation. More sophisticated optimization can also be done on each FPGA logic cell so that the resulting HCell macro meets the desired density, performance, and power cost metrics. Since the HCell macro library is pre-designed and the optimization problem is relatively small, we can improve results with longer runtime when necessary.

A complete example of these optimizations in the re-synthesis flow is shown in Figure 6: if the HCell architecture consists of a single MUX, then the 6-input LUT FPGA cell implementing the function

!D*C+D*E*!F*A+D*E*F*!B

is reduced to a 3 element HCell macro.

Relative to the combinational FPGA logic cells, we use a small number of registered HCell macros corresponding to all possible types of registered FPGA logic cells. These are built out of HCells to further reduce cost with respect to the FPGA logic fabric since we will only build the right type and right number of registers based on the FPGA design. Similarly for DSP blocks we use one type of HCell macro. Treating DSP blocks as "soft" logic enables us to build the blocks only on as-needed basis. It also allows us to only build the logic that is needed for a particular mode of the FPGA DSP block.

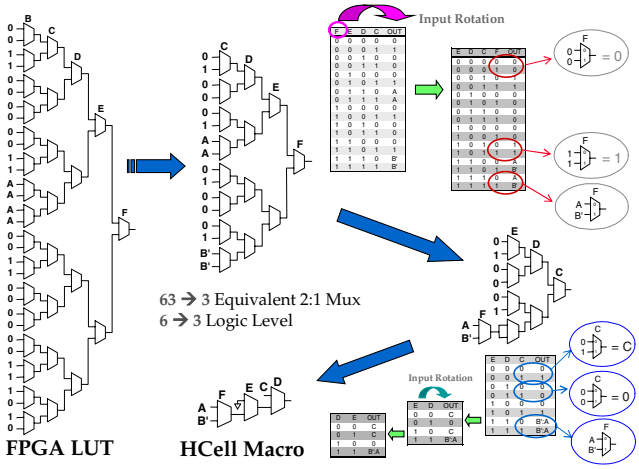

Figure 6. Optimization of a 6-input LUT

Figure 7 illustrates the logic cell to macro concept in more detail. Each of the logic portion and the register portion of the FPGA cell are transformed into an appropriate HCell macro in the library. Figure 8 shows specific examples: a 4-LUT and 6-LUT become macros with 2 and 3 HCells respectively, two example 2-LUTs use one HCell each, and a DFF with CLR is built from an HCell macro with 2 HCells. Template functions for such

re-synthesis functions form the HCell macro library to be discussed in Section 2.3.

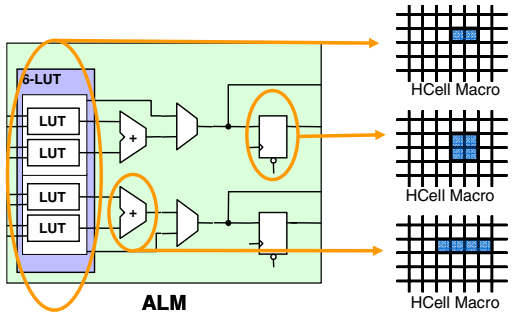

Figure 7 FPGA Logic Cell to HCell Macro Mapping

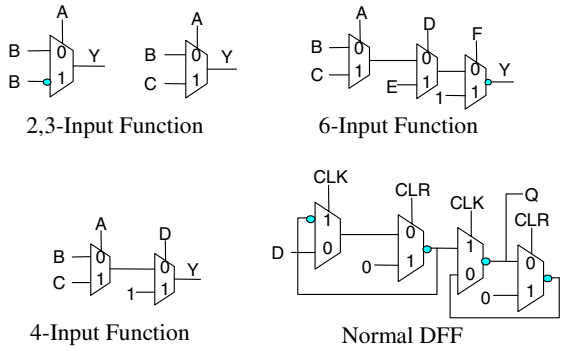

Figure 8. LUT & Flip-Flop generation with 2:1 MUX HCell

2.2 Architecture Experiments

We ran extensive architectural experiments to choose the best HCell architecture and HCell library.

Our experimental methodology is built by customizing an FPGA synthesis flow to allow re-synthesis of a given FPGA logic cell such as the Stratix II ALM into various base HCells. We used 100 commercial FPGA designs above 25,000 LUT4 logic elements in size. Each was re-implemented in HCell macros by the synthesis flow (described in Section 3) and then placed and routed using commercial ASIC placement tools.

Another logic fabric parameter is the number of HCells M for each HCell macro, which is dependent on the logic cell's complexity. Ideally we want to choose a threshold M large enough to cover the majority of combinational FPGA logic cells that we see in user designs without requiring an increase in HCell routing resources. The increased HCell size reduces the number of usable gates per square mm and makes the resulting device less cost-efficient.

Figure 9 shows the conversion of k-LUTs in the FPGA netlist into histograms of HCells required to implement them after re-synthesis. For this experiment, we continue to use the example 2:1 mux as the target HCell. We observe that nearly all combinational FPGA logic cells can be implemented in 6 or fewer HCells, and standardize on M=6. Thus, some LUTs are implemented with more than one HCell macro. Referring again to the 6-LUT of Figure 6, the logic reduces from 64 2:1 muxes and 64 LUT-mask SRAM bits to between 1 and 6 2:1 mux cells. In practice this is a 10:1 or more reduction in logic area.

2.3 HCell Macro Library

HCell macros can be pre-optimized and characterized for best area, performance and power. Pre-characterizing these macros and making them part of the library enables faster turn-around time since it avoids the need to optimize, verify, and characterize these cells on a per-design basis.

To achieve faster turn-around-time and to minimize the impact on layer routing resources, it's desirable to make the library of HCell macros large enough so that the majority of the combinational logic cells encountered in the FPGA designs can be directly translated into a single macro in the library.

Although a theoretical design could use any arbitrary logic cell in terms of its functionality we find in practice that if we accumulate the library cells using a sufficiently large number of designs the library can cover a majority of new design structures. This is similar to conclusions drawn by Trevillyan some time ago [21] in a study of common FPGA LUT-masks.

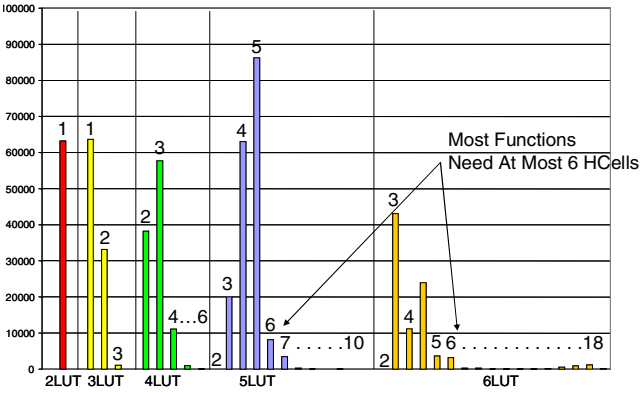

Figure 9. LUT to HCell Mapping Statistics

2.4 Expanding the HCell Library

The ALM allows LUTs of up-to k=7 inputs. Since it's unrealistic to enumerate all possible 7-input functions as HCell macros the HCell macro library is not exhaustive. On the other hand, there is a need to minimize the break up of one FPGA logic cell into multiple HCell macros since doing so uses more routing resources and thus negatively impacts routability by the back-end physical design tools.

We note that the initial library can be expanded as new designs are encountered. Previous work [21] has shown that though there are a doubly exponential number of LUT-masks in k only a small number are seen in practice and our experiments confirm this. The verification flow can deal with several thousands of macros.

The HCell macro accumulation flow consists first of synthesizing the design for an FPGA. Then, for each combinational logic cell, look it up in the HCell macro library. If there is no match, we optimize this logic cell using HCells and if it uses less than M HCells, then add it to the HCell macro library.

Figure 10 shows the decreasing benefits of library growth. The x-axis denotes additional new designs whereas the y-axis denotes the average logic portion, over the remaining designs in the design-set, that are not covered by the current library. We start out the library with arbitrary 50 designs. On average, less than 7% of the logic portion of a new design fails to be covered by a single macro in the pre-compiled library, and before all designs are used this decreases below 2%.

3. COMBINED SYNTHESIS FLOW

This design methodology enables easy migration from FPGA to a structured-ASIC, illustrated in Figure 11. For the FPGA prototype, we execute

1. Design source (HDL) extraction
2. RTL level optimization
3. Logic optimization, and
4. FPGA LUT and DFF technology mapping.

At this point, the ASIC design netlist is identical to the FPGA netlist. For the ASIC design, we perform an additional re-synthesis step before entering place and route wherein each of the FPGA logic cells in the FPGA netlist is converted into equivalent ASIC library cells, i.e. HCell macros. For each combinational and registered cell in the FPGA netlist there exists a corresponding HCell macro output cell which is functionally equivalent. In the case of combinational LUTs there could be several buried HCell macros. For DFF cells there is a 1:1 correspondence.

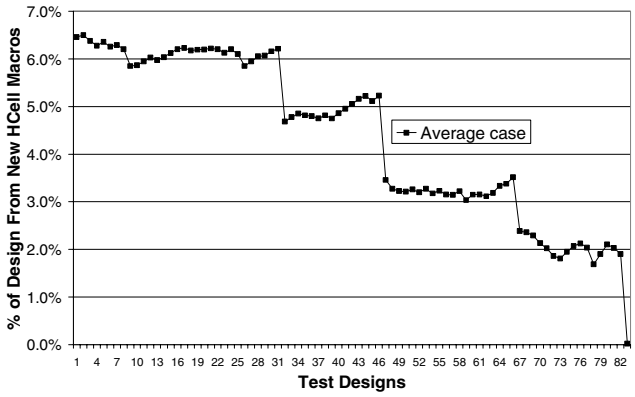

Figure 10. Effect of Fixed HCell Macro Library

Since we have designed the HCell macro library with a goal of covering most of the logic cells seen in the FPGA netlist, this translation process is fairly straight-forward and consists of looking up the proper HCell macro from the library using a binary-decision diagram (BDD) package.

We translate each cell in the FPGA synthesized netlist independently. Registers are translated directly into the corresponding HCell macro from the library. For combinational logic, we first perform a library lookup. This consists of searching the permutation group of the LUT-mask and checking correspondence in the library with a BDD. Matches are converted directly. Otherwise we cover the logic cell using the library cells. This can be done as follows: first we optimize the FPGA logic cell using a minimal number of HCells. Then in a recursive manner we select LUTs to cover the whole HCell network according to the following criteria – these LUTs have to be part of the ASIC library; in addition, these LUTs represent logic cones that do not have fanout outside the cone itself.

Note that the output of the original FPGA logic cell is still visible in the resulting ASIC netlist, which makes verification and debugging easy, as discussed in Section 4.

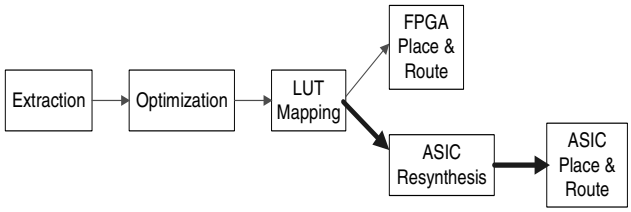

Figure 11. Combined Synthesis Flow

4. Verification Flows

Though the FPGA design can be tested in-system by programming the device, a replacement structured-ASIC cannot be tested in-system. Thus, the key aspect to an FPGA to ASIC migration is verification of identical functionality prior to mask fabrication.

When migrating a design from FPGA prototype to ASIC, the most difficult things to verify are the hard blocks such as PLLs, RAMs, DLLs, and IOs. With our proposed approach, these blocks are physically identical between an FPGA prototype and ASIC counter-part so that verification is reduced to the logic.

Logic verification is non-trivial in general, but with the structured synthesis flow of Section 3 we greatly simplify this by guaranteeing anchor-points at the output of every DFF and combinational LUT in the FPGA netlist.

Verification thus consists of:

1. Write fpga.*v* at the end of the FPGA compile.
2. Write asic.*v* at the end of the ASIC compile.
3. Compare *fpga.v* against *asic.v* using a standard formal verification tool.

The generally complicated issues of formal verification between an original and modified netlist – node matching, don't-care synthesis, duplicated or merged registers, etc. do

not exist. Cones with numerous anchor points are guaranteed by construction. For combinational FPGA cells that require more than one ASIC library cell to represent, the re-synthesis step asserts before and after the step that the functionality doesn't change. Since in general the FPGA cell has a small number of inputs, this checking is done efficiently using functional techniques such as 01-tables and BDDs.

For the complicated cells (PLLs, RAM, etc), we have a 1:1 physical match between FPGA to ASIC. Synthesized registers and clocking are a 1:1 functional match and are guaranteed anchor points for verification. Re-synthesized logic is functionally equivalent but not 1:1 and must be fully verified. For this we store the functionality as a bit string in the library cell name, and treat the library cell description (usually in Verilog or VHDL) as just another design, then synthesize it for FPGA, and compare the bit strings. This is summarized in Figure 12.

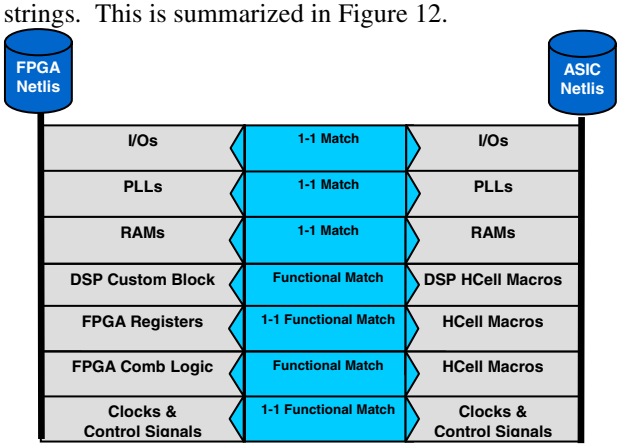

Figure 12. Block Level Matching Between FPGA and ASIC

5. Conclusions

In this paper, we have described a methodology for designing a structured-ASIC closely tied to a prototype FPGA. The key elements of the proposal are in the design of a structured-ASIC base array and re-synthesis and verification flows that greatly simplify migration.

In the proposed fabric a single HCell forms the basis for all logic re-synthesized from the FPGA netlist. HCells are grouped into HCell macros which implement functionality directly corresponding to DSP blocks, FPGA combinational logic cells and FPGA registers, in which all anchor points are 1:1 and directly visible to verification.

Overall this fabric and methodology allows for a direct correspondence and simple flow from FPGA to ASIC. The methodology is independent of the FPGA fabric and exact HCell characteristics used.

References

[1] R. Camposano, "Will the ASIC Survive", Keynote Address at SBCCI 2004, p5.

[2] W. Dally and A. Chang, "The Role of Custom Design in ASIC Chips", in Proc. DAC 2000, pp. 643-647.

[3] A.B. Kahng, "Design Technology Productivity in the DSM Era", in Proc ASPDAC, 2001.

[4] A. El-Gamal, J. Cohn, A. Kahng, I. Bolsens, A. Broom, C. Hamlin, P. Magarshack, Z. Or-Bach and L. Pileggi, "Fast, Cheap and Under Control: The Next Implementation Fabric", in Proc. DAC 2003, pp. 354-355.

[5] Dataquest; "ASIC Design Times Spiral Out of Control"; Gary Smith; January 2002.

[6] C. Patel, A. Cozzie, H. Schmit and L. Pileggi, "An Architectural Exploration of Via Patterned Gate Arrays", in Proc. ISPD 2003, pp. 184-189.

[7] K. Tong, V. Kheterpal, V. Rovner and L. Pileggi, "Regular logic fabrics for a via patterned gate array (VPGA)", in Proc. CICC 2003.

[8] L. Pileggi *et.al.*, "Exploring Regular Fabrics to Optimize the Performance-Cost Trade-Off", in Proc. DAC 2003, pp. 782-787.

[9] V. Kheterpal, A.J. Strojwas and L. Pileggi, "Routing Architecture Ex07.

[10] R. Reed Taylor and H. Schmit, "Enabling Energy Efficiency in Via-Patterned Gate Array Devices", in Proc. DAC 2004, pp. 874-877.

[11] R. Reed Taylor and H. Schmit, "Creating a Power-aware Structured ASIC", in Proc. ISLPED 2004, pp. 74-77.

[12] B. Hu, H. Jiang, Q. Liu and M. Marek-Sadowska, "Synthesis and Placement Flow for Gain-Based Programmable Regular Fabrics", in Proc. ISPD 2003, pp. 197-203.

[13] V. Betz, J. Rose and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer, 1999.

[14] Y. Ran and M. Marek-Sadowska, "On Designing Via-Configurable Cell Blocks for Regular Fabrics", in Proc. DAC 2004, pp. 198-203.

[15] N. Shenoy, J. Kawa and R. Camposano, "Design Automation for Mask Programmable Fabrics", in Proc. DAC 2004, pp. 192-197.

[16] K-C Wu and Y-W Tsai, "Structured ASIC, Evolution or Revolution", in Proc. ISPD 2004, pp. 103-106.

[17] T. Okamoto, T. Kimoto and N. Maeda, "Design Methodology and Tools for NEC Electronics' Structured ASIC ISSP", in Proc. ISPD 2004, pp. 90-96.

[18] D. Sherlekar, "Design Considerations for Regular Fabrics", in Proc. ISPD 2004, pp. 97-102.

[19] M. Hutton *et.al.*, "Improving FPGA Performance and Area Using an Adaptive Logic Module", in Proc. FPL 2004, pp. 135-144.

[20] D. Lewis *et.al,* "The Stratix II Logic and Routing Architecture", in Proc. FPGA 2005, pp. 14-20.

[21] L. Trevillyan, "An Experiment in Technology Mapping for FPGAs Using a Fixed Library", in Proc. IWLS 1993