FAST AND ACCURATE RESOURCE ESTIMATION OF RTL-BASED DESIGNS TARGETING FPGAS

Paul Schumacher and Pradip Jha

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Email: {paul.schumacher, pradip.jha}@xilinx.com

ABSTRACT

FPGAs have become complex, heterogeneous platforms targeting a multitude of different applications. Understanding how a design maps to them and consumes various FPGA resources can be difficult to predict, so typically designers are forced to run full synthesis on each iteration of the design. For complex designs that involve many iterations and optimizations, the run-time of synthesis can be quite prohibitive. In this paper, we describe a fast and accurate method of estimating the FPGA resources of any RTL-based design. We achieve run-times that are more than 60 times faster than synthesis and is on average within 22% of the actual mapped slices across a large benchmark suite targeting three different FPGA families. This resource estimator tool is first provided in Xilinx PlanAhead 10.1.

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) have become quite diverse in the types of applications that utilize them. These applications can include everything from real-time video processing and other complex digital signal processing (DSP) functions; to various forms of soft instruction set processors for control-based applications; to communication and networking designs. In order to target these diverse applications, the resources available on these FPGA platforms have also become quite diverse in their own right. Modern FPGAs contain not just basic, programmable building blocks including look-up tables (LUTs), flip-flops (FFs), and additional carry logic, but more complex, on-chip blocks as well, such as block memories (e.g., block RAMs), DSP elements (e.g., multipliers, DSP48s), and high-speed transceivers [2].

It is important for designers that use these diverse, heterogeneous FPGAs to understand the consequences of decisions made during the process of capturing the design in a synthesizable manner. As pointed out in [2], FPGA resource usage is an important measure of hardware cost (besides path delay and power consumption). The sooner the designer is aware of the hardware impact of coding decisions, the sooner he can make any necessary improvements and corrections before they become hidden in a large design implementation. Performing full synthesis at each design iteration to obtain hardware resource estimations can become quite time-consuming, especially for large, complex designs.

A fast method to obtain detailed hardware resource estimations is essential in modern FPGA design. A large amount of past work in FPGA resource estimation tends to focus on specific high-level languages and tools, including C and its variants [3], [4], [5] and Matlab/Simulink [2], [6], [7]. While highly-integrated with a specific tool flow, its usage cannot be expanded outside of the specific tool or language. These techniques also tend to focus on only one FPGA resource (e.g., LUTs). Others have targeted specific cores or applications [8], exploiting knowledge of these specific designs to obtain FPGA resource estimations. While quite accurate, these approaches also tend to be somewhat narrow in focus and are difficult to extend to other designs or applications. Some have used an RTL front-end [9], but run-times are expected to be prohibitive for large FPGAs. In this paper, we describe a fast FPGA resource estimation tool that can work with any RTL-based tool flow, including high-level synthesis flows that create RTL. The estimation techniques used in our approach are discussed in Section 2; results of the estimation tool are described in Section 3, including run-time and estimation accuracy across three FPGA families; while conclusions are discussed in Section 4.

2. ESTIMATION TECHNIQUES

The goal of our resource estimator was to provide fast and immediate feedback to the designer on the hardware impact of coding decisions *when the decisions are being made*. Note that this estimation was not meant to be full synthesis. We are not actually *mapping* to the FPGA hardware or

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

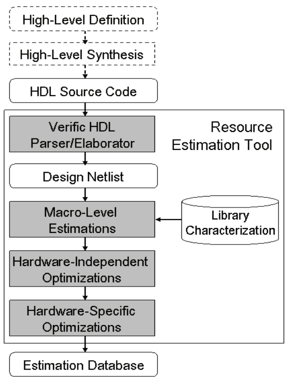


Fig. 1. Resource estimation tool flow diagram

performing low-level synthesis optimizations, we are only *modeling* the steps that synthesis is expected to take. As our goal was to create a tool that provides estimations within 30 seconds for typical designs, there are many low-level synthesis tasks that the tool cannot perform due to time constraints. We did, however, derive some fast modeling techniques and optimizations that work well for many designs.

Fig. 1 shows the basic tool flow within the resource estimator. The first step of the tool is to parse and elaborate the RTL-based design, which can optionally be provided by running high-level synthesis. In our tool, this step is performed by the Verific® parser/elaborator tool [10]. This tool was helpful in meeting our speed requirements of 30-second run-time, and it supports a number of HDL source types, including VHDL, Verilog, and SystemVerilog. Verific produces a design netlist targeting a library of parameterized primitives and operators (henceforth, inclusively called *macros).*

This library of macros targeted by Verific includes elements we categorize as follows: bitwise/binary logic, unary logic, arithmetic elements, comparators, multiplexers, shifters, and storage (including registers and memories). There are many benefits to describing the design in terms of these well-understood functional categories as it 'speaks the language' of the designer (see Section 3.3). In order to create this description and make relative comparisons between categories, we need to multiply each of these macros by weighting factors as not all macros are equal in size (e.g., a 64x64-bit multiplier vs. a 3-bit adder). Doing so is the basis of resource estimation.

As shown in Fig. 1, our estimation tool first creates estimations on each of these macros based on their parameters and pre-characterized results (e.g., an 8-bit adder uses eight LUTs). An automatically-generated database of curve-fitted equations is used in this step, covering all library macros and targeting all desired FPGA families (see Section 3). Taken out of context, these macro-level estimations are indeed accurate. However, as these macro-level estimations ignore a number of structural optimizations performed by synthesis, simply summing these estimations would typically give a gross over-estimation for the entire design.

To model this synthesis behavior, the estimator performs a number of second-tier optimizations on the Verific design netlist (see Fig. 1). These optimizations can be separated into three distinct categories: hardware-independent optimizations, which are performed across all target FPGA families and architectures; hardware-specific optimizations, which are run based on the availability and specific characteristics of a number of hardware elements in the target architecture; and estimated slice packing, which can be difficult to do without fully understanding the low level implementation details of the design.

2.1. Hardware-Independent Optimizations

The hardware-independent optimizations that are performed include but are not limited to the following:

Resource Sharing: The estimator finds common element types with common inputs and assumes shared resources. For example, if two adders are found, both *A+B*, then it is assumed the adder logic is shared. A limit is set on the number of elements merged so as to limit fan-out.

Comparators with Constants: A special case of resource sharing involves comparators with one input common and the other constant. When found, the tool assumes limited merging of resources. This is a common occurrence in some designs, especially those with data- and action-based schedulers.

Dangling/Disabled Logic: The estimator finds logic with either no load or is *disabled* (e.g., logic driving the first selection on a multiplexer when the select line is stuck high) and assumes zero resources are consumed. Once one such element is found, all driving logic is also removed, taking precautions to ensure that other logic is not also being driven by this element.

ROM Implementation: The estimator identifies large multiplexers with all inputs constant and assumes a ROM implementation. This type of circuit typically comes from ROMs described as large *switch/case* statements defined in the HDL.

2.2. Hardware-Specific Optimizations

There are a number of optimizations performed in the resource estimator tool that are dependent upon exploiting specific FPGA hardware elements. Performing these optimizations becomes a form of template matching, looking for specific functional patterns and structures. This set of optimizations includes but is not limited to the following:

Block RAM Memories: Memories and their supporting hardware are identified (e.g., read enable multiplexers, port registers) to target on-chip FPGA block memories. Note that the characteristics that are taken into account include: memory depth, bit width, number of write/read ports, and asynchronous/synchronous nature of ports.

Multiplier/Arithmetic Functions: The tool identifies arithmetic functions and their supporting hardware (e.g., load multiplexers, input/output registers) that target on-chip multipliers and/or DSP blocks (e.g., XtremeDSP^{TM} slices or DSP48s on Xilinx Virtex^{TM}-4 and Virtex-5 FPGA families). This includes all functionality expected to be found by the synthesis tool, including multiplier-accumulators (MACCs).

Shift Registers: The estimator finds chains of registers, verifies their controls (e.g., common clock enable), and if possible, optimizes their implementations using SRL16s (or SRL32s on Virtex-5).

Register Controls: There are a number of programmable options for slice flip-flops on Xilinx FPGAs (e.g., synchronous or asynchronous set/reset, clock enable). The tool identifies this functionality – often expressed in the form of multiplexers and registers – and optimizes for the target FPGA architecture.

IOB Registers: Registers that are connected to inputs, outputs, or bi-directionals (IOBs) in the design are consumed by the IOBs when appropriate.

Multiplexer Trees: Sequences of multiplexers are found by the tool and their implementations are optimized to the LUT-based FPGA hardware.

Binary Logic Trees: The tool finds sequences of binary logic functions that will be implemented with LUTs, and optimizes them with regards to both the number of unique inputs of the tree as well as the number of outputs. As we are unable to perform complete logic optimization in the limited time frame given the tool, we also include a factor to model this behavior by increasing the number of inputs for very large binary trees.

2.3. Estimated Slice Packing

One difficult task in estimating FPGA resources is determining how the LUTs and flip-flops get packed into slices on Xilinx FPGAs [1]. In past work, either only configurable logic blocks (CLBs) or slices is estimated [4], [6], [9]; or only LUTs and/or flip-flops is reported [3], [5], [7], but not both. In order to estimate the number of slices consumed by a design, we used a set of 90 representative design (see Section 3), each targeting three different FPGA families. We used the post-map results from these designs and performed curve fitting using the following equation:

$$Slices = a_0 \cdot LUTs + a_1 \cdot Flops + a_2 \cdot LUTs \cdot Flops \qquad (1)$$

where *LUTs* is the number of estimated LUTs in the design and *Flops* is the number of estimated flip-flops. Whereas the relationship to *LUTs·Flops* has no physical correlation with the hardware, it allows the equation to correctly handle

Table 1. Factors used in FPGA slice equations

Factor	Virtex-4	Virtex-5	Spartan-3
a_0	4.497e-01	2.447e-01	4.893e-01
a_1	3.543e-01	1.813e-01	2.965e-01
a_2	2.067e-07	-5.415e-07	-8.126e-07

designs that may be relatively heavy in either LUTs or flops. Table 1 lists the factors in Eq. (1) found from curve-fitting the representative designs for each FPGA family. The average error across all families in using Eq. (1) to estimate slices using the actual LUT and flop counts was 7.85%.

After these three sets of optimizations are performed by the estimator tool, a *searchable database* of resource estimations is created. The designer can then find estimated hardware resources at any level in the design hierarchy as well as a novel view of the design categorized by resource functionality (see Section 3.3). These resource functional categories are also stored in the database at every level of the hierarchy, providing a detailed view of expected hardware resources for any requested portion of the design.

3. EXPERIMENTAL RESULTS

The resource estimation tool described in Section 2 has been run on a large suite of benchmark designs representing a diverse set of resource sizes, applications, and types of source code (i.e., VHDL, Verilog, as well as HDL generated from a few high-level synthesis tools). Three FPGA families were also targeted: Spartan^{TM}-3, Virtex-4, and Virtex-5. The two important criteria used to judge the quality of the results were estimation accuracy and run-time. A third, novel output of the estimator is the resources by functional categories, described in Section 3.3.

3.1. Estimation Accuracy

Table 2 lists the estimation accuracy for six different designs, representative of the benchmark suite and of three distinct applications: video processing, soft, instruction-set processors, and communications/networking. The estimated FPGA resources are compared against post-map results for all types of resources: slices, LUTs, flip-flops, block RAMs (BRAMs), and multipliers/DSP48s (Mult/DSP48). Across the entire benchmark suite of 90 designs, each targeting all three FPGA families, the average estimation errors were 21.9%, 26.3%, and 14.2% for slices, LUTs, and flip-flops, respectively.

As mentioned in Section 2.2, estimating the block RAM and multiplier/DSP48 counts can be complicated as these memory and arithmetic operations can be implemented in either the programmable fabric (i.e., LUTs, flip-flops) or the hard, on-chip elements included specifically for these functions. Previous work either used design-specific

Table 2. Estimated FPGA resources vs. post-map results for six designs

Design	FPGA Family	Post-Map Results					Estimation Results				
		Slice	LUT	Flop	BRAM	Mult/ DSP48	Slice	LUT	Flop	BRAM	Mult/ DSP48
Video Processing Design #1	Spartan-3	5203	9013	3331	32	42	5815	9356	4284	33	33
	Virtex-4	5036	8751	2719	32	32	4682	7988	3062	33	33
	Virtex-5	2057	5753	2733	34	31	1774	4970	3123	33	33
Video Processing Design #2	Spartan-3	3992	6403	3584	9	8	4019	5284	4906	9	8
	Virtex-4	4066	6609	3510	9	8	4144	5491	4714	9	8
	Virtex-5	2177	5657	3479	10	8	1806	4180	4375	9	8
Inst. Set Processor Design #1	Spartan-3	29669	53294	11720	36	5	24831	44409	11908	29	5
	Virtex-4	30145	53999	11759	36	5	24169	44251	11748	29	5
	Virtex-5	12053	34661	11654	27	5	9832	32231	11866	29	5
Inst. Set Processor Design #2	Spartan-3	8808	15184	3544	0	0	9361	17168	3400	0	0
	Virtex-4	8875	15266	3570	0	0	8950	17197	3400	0	0
	Virtex-5	3828	10559	3512	0	0	3847	13301	3400	0	0
Comm/ Networking Design #1	Spartan-3	8312	12618	5628	9	0	10317	16615	7728	10	0
	Virtex-4	9440	14255	7294	10	0	10206	16549	7728	10	0
	Virtex-5	4033	10389	7261	8	0	3685	9507	7711	10	0
Comm/ Networking Design #2	Spartan-3	4509	6776	4397	0	0	4143	5462	5036	0	0
	Virtex-4	4404	6575	4405	0	0	4246	5462	5036	0	0
	Virtex-5	2294	5397	4401	0	0	1896	4065	5036	0	0

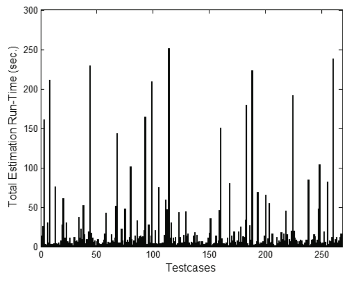
knowledge to estimate these counts [8], or simply ignored them altogether [3], [4], [5], [6], [7]. Identifying the characteristics and parameter thresholds for memories and arithmetic functions to target block RAMs and multipliers/DSP48s, respectively, is expressed in our estimator in the form of macro-level equations which are then verified in the second-tier optimizations. For our entire benchmark suite targeting all three families, the average errors for block RAMs and multipliers/DSP48s were 16.6% and 3.6%, respectively.

3.2. Run-Time

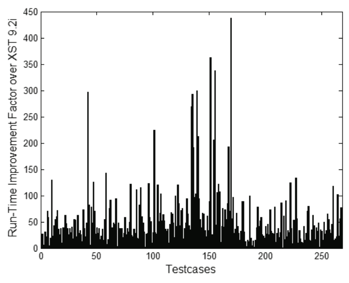
Fig. 2 shows the total run-times of the estimator tool (in seconds) as well as the speed-up improvement factor over XST, the Xilinx synthesis tool, for each test case. Across the entire benchmark suite of 90 designs, each targeting three families, the median and average run-times for the resource estimator were 7.1 seconds and 19.1 seconds, respectively. The longest run-time was 251.6 seconds while the shortest run-time was 1.28 seconds. This run-time includes both the Verific parser/elaborator as well the estimator tool. This run-time is on the average 60.8 times faster than XST [1] run-time.

3.3. Resources by Functionality

Fig. 3 shows the functional categories that can provide helpful insight into the contents of a design. This type of information is expected to be invaluable to the designer in many stages of the design process, pinpointing not just *where* but also *how* the resources are being consumed. These functional categories are provided at all levels of the design hierarchy, specifying not just slices as shown in Fig. 3, but all FPGA resources, including slices, LUTs, flip-flops, block RAMs, and multipliers/DSP48s.



(a)



(b)

Fig. 2. Resource estimation run-time: (a) in seconds; and (b) improvement factor over XST 9.2i

Armed with this information, a designer should be able to identify the most resource-expensive portions of the design as well as the causes and reasons for this size. For example, both of the soft, instruction-set processor designs are quite heavy in bitwise/binary logic. As the designer peruses the hierarchy of the design, it may become apparent that, say, an ALU is expressed in an inefficient manner. A

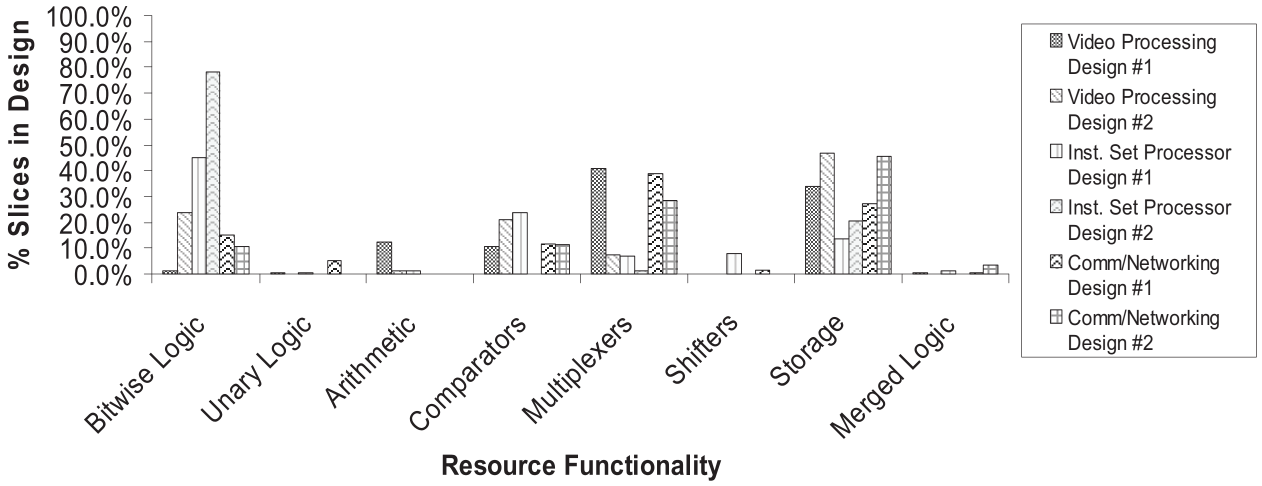


Fig. 3. Functional categories of estimated slice resources for six designs

change could then be made to the design to take advantage of the ALU functionality in the DSP48 element. Other pertinent information such as bit widths could also be used to help pinpoint the exact locations for targeted design improvements. This resource categorization feature is expected to be used extensively in many stages of the design process, from early on while the design is still being defined to much later in the process when optimizations are being performed in the refinement stage.

4. CONCLUSION

A novel RTL-based, FPGA resource estimator has been developed. The run-times are on average over 60 times faster than synthesis run-times, providing immediate feedback to the designer. The tool provides estimates for all available types of FPGA resources and has been shown to be accurate across a diverse set of designs and applications. A number of hardware-independent and hardware-specific optimizations are mentioned, enabling this accuracy in the fast modeling of synthesis. A fascinating view of the contents of a design is also compiled and provided in functional categories. This provides valuable insight into the composition of the design and enables the designer to pinpoint potential targets for improvements without the need to run synthesis.

5. REFERENCES

[1] http://www.xilinx.com

[2] Shi, C., Hwang, J., McMillan, S., Root, A., and Singh, V., "A System Level Resource Estimation Tool for FPGAs", *International Conference on Field Programmable Logic and Applications (FPL)*, 2004.

[3] Brandolese, C., Fornaciari, W., and Salice, F., "An Area Estimation Methodology for FPGA Based Designs at SystemC-Level", *Design Automation Conference (DAC)*, June 2004.

[4] Bilavarn, S., Gogniat, G., Philippe, J., and Bossuet, L., "Fast Prototyping of Reconfigurable Architectures From a C Program", *International Symposium on Circuits and Systems (ISCAS)*, 2003.

[5] Kulkarni, D., Najjar, W., Rinker, R., and Kurdahi, F., "Compile-Time Area Estimation for LUT-Based FPGAs", *ACM Transactions on Design Automation of Electronic Systems*, January 2006.

[6] Nayak, A., Haldar, M., Choudhary, A., and Banerjee, P., "Accurate Area and Delay Estimators for FPGAs", *Design, Automation, & Test in Europe (DATE)*, 2002.

[7] Bjureus, P., Millberg, M., and Jantsch, A., "FPGA Resource and Timing Estimation from Matlab Execution Traces", *CODES*, May 2002.

[8] Milder, P., Ahmed, M., Hoe, J., and Puschel, M., "Fast and Accurate Resource Estimation of Automatically Generated Custom DFT IP Cores", *FPGA Conference*, February 2006.

[9] Xu, M. and Kurdahi, F., "ChipEst-FPGA: A Tool for Chip Level Area and Timing Estimation of Lookup Table Based FPGAs for High Level Applications", *ASP-DAC*, 1997.

[10] http://www.verific.com