

SmartOpt: An Industrial Strength Framework for Logic Synthesis

Stephen Jang, Dennis Wu, Mark Jarvin

Billy Chan, Kevin Chung

Xilinx Inc.

{sjang,wudenni,mjarvin,billy,kevinc}@xilinx.com

Alan Mishchenko Robert Brayton

University of California, Berkeley

{alanmi,brayton}@eecs.berkeley.edu

ABSTRACT

In recent years, the maximum logic capacity of each successive FPGA family has been increasing by more than 50%, which motivates scalable solutions. Meanwhile, academic research in logic synthesis has been fruitful, but these advances have been demonstrated on academic architectures and benchmark designs which are not representative of modern industrial FPGAs. This paper presents a framework (SmartOpt) for mapping complex FPGA architectures to a simple netlist model, which can be supported by academic tools. SmartOpt was applied to leverage the algorithms implemented in the ABC package and to study their relative contributions. This work is integrated into the Xilinx ISE 11.1 software flow for FPGAs and shows significant improvements in both the LUT count and performance of large industrial circuits described in HDL. Xilinx Synthesis Technology (XST) reference flow was compared experimentally against the same flow augmented by SmartOpt. When applied to a set of 20 large industrial Virtex-5 benchmarks ranging from 17K to 69K 6-LUTs, the augmented flow produced 8.3% fewer LUTs and led to 2.1% higher operating frequency while keeping runtimes reasonable. With dual-LUT-merging, the LUT count is reduced by 22.7%, while increasing the operating frequency only by 0.7%.

Categories and Subject Descriptors

B.6.3 [Logic Design]: Design Aids – Optimization;

B.7.1 [Integrated Circuits]: Types and Design Styles – Gate arrays

General Terms

Algorithms, Performance, Design, Experimentation, Modelling

Keywords

FPGA, Technology Mapping, Edge Flow, Interface, BLIF, ABC

1. INTRODUCTION

The work of [6] represents a prior attempt to harness recent academic synthesis tools in an industrial flow. The academic and industrial FPGA architectures were bridged by protecting certain aspects of the netlist through the addition of the ability to

instantiate opaque sub-modules (“black boxes”). This approach is limited because it replaces box inputs/outputs with circuit outputs/inputs. The lack of functional and timing information does not allow the synthesis tool to have an accurate view of the circuit structure, limiting its optimization opportunities and preventing it from synthesizing a netlist capable of meeting the user-imposed timing constraints.

This paper presents an improved approach to bridge academic and industrial FPGA architectures at the logical level. It allows cutting-edge academic synthesis tools to be applied in an industrial CAD flow targeting an advanced FPGA architecture.

2. BACKGROUND

2.1 Challenges for Industrial FPGAs

Industrial FPGAs [1][2] include complex blocks such as memories, multipliers, DSP logic, carry chains, wide-function multiplexers, multi-output LUTs, and so on. In addition, the circuit model in the industrial setting is more flexible: combinational cycles are possible; a node’s fanins may each have multiple drivers; flip-flops may include control sets, including synchronous or asynchronous resets and clock enables.

Furthermore, industrial synthesis requires support of various constraints to be enforced during synthesis, such as, nets that must be preserved and hierarchical boundaries that must not be altered [7]. A variety of delay requirements may be expressed, ranging from simple period constraints to complex path requirements.

The black-boxing strategy of [6], while sufficient for representing circuits for academic exploration, is inadequate for integrating academic tools into industrial flow for two reasons:

1. It provides no information on delay requirements to the synthesis tool, making it impossible for the tool to achieve performance objectives.
2. It provides no information about the contents of the boxes. This restricts optimization potential by limiting the propagation of timing information needed to evaluate delay tradeoffs, under-approximating area flow during mapping, not extracting don’t-cares using logic functions of the boxes, *etc.*

The result of this black-box integration strategy is that both the synthesis and mapping applications work on the netlist broken into a number of independent shallow logic cones, which severely limits optimization.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FPGA Conference '09, February 22-24, 2009.

Copyright 2008 ACM 1-58113-000-0/00/0004...\$5.00

3. NETLIST MODELING AND INTERFACE

This section presents the netlist modeling and interface techniques used to connect Xilinx’s ISE tool chain to an external synthesis package (in this case the ABC Logic Synthesis and Verification package [4]). We will describe how the ABC circuit model [3] is extended with boxes similar to those in [6] but augmented to include timing and logical information; how optimizations due to user or device constraints are prevented; and how complex FFs available in modern FPGAs are modeled. In this paper, all sequential components are assumed to be driven by the same clock. We conclude this section with a description of how this model is integrated into the Xilinx ISE tool chain

3.1 Modeling Architecture-Specific Delays

We use the term “**black box**” to refer to an opaque multi-input, multi-output block that represents a portion of the logic that can be neither modified nor modeled in the academic tool. The important distinction between these and the black boxes of [6] is that timing information can be included for any combinational or sequential paths passing through the black box.

In this work, similar to [6], the format traditionally used to describe logic networks in Berkeley tools (BLIF) was modified to represent timing and other information related to the boxes.

```
.model FA
.inputs a b cin
.outputs s cout
.attrib black box comb
.delay a s 0.01
.delay b s 0.01
.delay cin s 0.01
.delay a cout 0.02
.delay b cout 0.02
.delay cin cout 0.02
.end
```

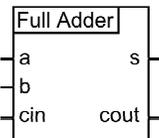


Figure 3.1: Combinational black box model in the extended BLIF, with timing information of a full adder.

A black box can be combinational or sequential. Combinational black boxes, marked with the *comb* attribute in the extended BLIF, allow us to link these segments together using propagation delays of the box paths. The *delay* attribute (shown in Figure 3.1) specifies the propagation delay between each input-output pair of the box

```
.model FDRSE
.attrib black box seq
.inputs D R S CE
.outputs Q
.input_required D 0.0
.input_required R 0.01
.input_required S 0.01
.output_arrival Q 0.5
.end
```

Figure 3.2: Sequential black box model with timing for a flip-flop with clock enable and synchronous set/reset.

Black boxes containing state elements, such as flip flops, registered DSPs, or BRAMs, are called sequential black boxes, and are specified using the *seq* attribute in the extended BLIF (see Figure 3.2). Sequential boxes are annotated with *output_arrival* and *input_required* times in place of the propagation delays of combinational boxes. *input_required* represents the setup time of a box input whereas *output_arrival* times represent the clock-to-output time of a box output. This additional timing information is used to make better optimization decisions – for instance,

unbalancing paths to accommodate differences between BRAM and FF clock-to-output delays.

3.2 Modeling Persistent Logic

The simple gates present in the slice are instantiated either by users or during macro inference and cannot be modified with the same degree of freedom as the general logic of the circuit. However, these simple gates provide valuable information about the functionality of the circuit. We model them with transparent multi-input, multi-output blocks, which allow representation of the logical function but prevent modification of their contents. We refer to these blocks without analogue in [6], as “**white boxes**”.

White boxes, like black boxes, may be either sequential or combinational and may have timing information associated with them. Despite not being able to modify the implementation of white boxes, the synthesis tool is able to make inferences that depend on the functionality implemented by the box and to propagate functional information across the box, for example, during simulation of the circuit. This improves the synthesis tool’s ability to find equivalent nodes in the netlist and to extract and exploit don’t-cares, further augmenting the benefit of having accurate timing information for the box.

An example of a white box representing a full adder is shown in Figure 3.3. The difference between the white box representation and the black box model in Figure 3.1 is the addition of functional information for the full adder.

```
.model FA
.inputs a b cin
.outputs s cout
.attrib white box comb
.delay a s 0.01
.delay b s 0.01
.delay cin s 0.01
.delay a cout 0.02
.delay b cout 0.02
.delay cin cout 0.02
.names a b cin s
100 1
010 1
001 1
111 1
.names a b cin cout
-11 1
1-1 1
11- 1
.end
```

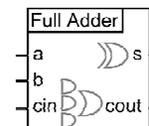


Figure 3.3: Combinational white box model in the extended BLIF with timing and functional information of a full adder.

In general, we find white boxes to be ideal for representing the simple fixed-function blocks abundant in modern FPGAs—MUXCYs, XORCYs, and MUXFXs—as well as user-instantiated LUTs and other small logic that must be preserved during optimization. Black boxes are best-suited for representing complex hard blocks with atypical timing characteristics—such as DSPs, BRAMs, and hard processors, which would be too costly to model—and other primitives that cannot be modeled using simple synchronous circuit elements, such as scan-chains and latches.

3.3 Sequential Representation

Accurate modeling of complex sequential elements is crucial to exploiting the full potential of innovative approaches to sequential synthesis.

White boxes enable representation of complex sequential elements while preserving their functionality through network optimization.

As an example, Figure 3.4 shows a white box model capable of representing the functional behaviour of a complex flip-flop with a synchronous set, a synchronous reset, and a clock enable. If the control logic and basic flop are not white boxed together, then, in general, the control logic will be merged and optimized with the general LUT fabric. By white boxing the control logic together with the basic flop, the complex flop is guaranteed to be preserved during optimization and can be mapped back to dedicated hardware. The complete timing of the FF including control set pins is used during mapping. All FF pins are treated as combinational primary inputs/outputs.

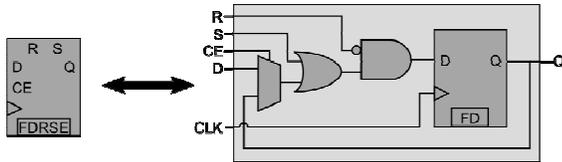


Figure 3.4: Representing a flip-flop with clock enable, synchronous set, and synchronous reset under the academic model.

3.4 Integration

The Xilinx implementation tool flow is illustrated in Figure 3.5. The user’s design can be presented to the tool chain using HDL source or an EDIF file produced by a third party synthesis tool. For HDL designs, Xilinx Synthesis Technology (XST) is used to perform RTL parsing and elaboration, register level optimization, and logic level optimization. The output of XST (or equivalently the EDIF file from a third party synthesis tool) is fed to the MAP application, which performs packing and placement on the synthesized netlist. The output of MAP is then fed to the PAR application to perform routing.

In XST, technology-independent logic synthesis is performed after high-level (advanced) synthesis, and before low-level (architecture-specific) synthesis. Alternatively, the MAP application can perform the same logic synthesis operations prior to technology mapping, allowing re-synthesis of designs that are created by a third-party synthesis tool, such as Synplicity.

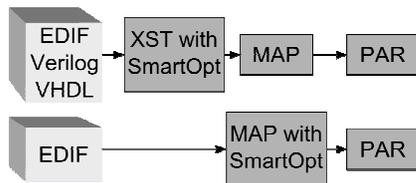


Figure 3.5: Supported flows for integrated academic-industrial synthesis.

As illustrated in Figure 3.5, we integrate the ABC synthesis engine into Xilinx’s XST and MAP applications using the box model described above. This configuration is referred to as XST/MAP with SmartOpt.

4. EXPERIMENTAL RESULTS

Experiments were conducted to demonstrate the power of SmartOpt enhanced with ABC algorithms and address the following points:

1. The overall effect of the ABC algorithms in the new framework.
2. The individual contributions due to white boxing with timing information.
3. The effect of the WireMap heuristics on LUT merging for Virtex-5 designs.

These experiments were performed on 20 large designs from an internal benchmark suite targeting Virtex-5 devices. All of the ISE tools are using the default settings which obtains the best performance. The reported maximum frequency is obtained using the TRCE static timing analysis tool. Flip-flop and LUT counts for the placed netlist are obtained from the MAP (technology mapping) report. All experimental results are in Table 4.1 at the end of this paper.

4.1 Mapping without White Boxes or Timing

The first experiment studies the effect of using ABC algorithms without timing annotation and without white box support. This is the circuit model supported in the original BLIF specification. Combinational hard elements, such as MUXCYs and XORCYs, are converted to PIs with an *input_arrival* time of 0 and POs with an *output_required* time of 0. This has the effect of fragmenting combinational paths and prevents ABC from performing timing analysis over persistent logic or macro logic. Sequential elements are replaced in ABC with PIs with an *input_arrival* time equal to the clock-to-output time of the output, and POs with an *output_required* time equal to the setup time of the input. Real PIs and POs are converted into PIs with *input_arrival* times of $-\infty$ and POs with *output_required* times of $-\infty$.

Compared with the reference flow, this “SmartOptNwbNt” flow uses 2.5% less LUTs, the same number of flip-flops, and is 0.7% faster. ABC is able to achieve both LUT count reduction and modest speed improvement without complete timing and functional information. There are no flip-flop optimizations since they are treated as primary inputs/outputs. The delay improvement is less than that of the final SmartOpt result because without timing information, ABC will mistakenly optimize some critical paths for area at the expense of delay.

4.2 Mapping Without White Boxes and With Timing

In this experiment, full timing annotation is enabled. Persistent combinational elements, such as MUXCYs and XORCYs, are represented as boxes with combinational delays specified for each input and output pair. However, the logical content of the box is not provided, which prevents logical redundancies from being propagated through the box. In particular, the absence of set/reset control white box logic prevents sequential analysis because the input to the flip-flop is not longer related to the rest of the circuit.

We observed that this “SmartOptNwb” flow uses 3.1% less LUTs, the same number of flip-flops as the reference, and is 1.7% faster. As expected, the main improvement of adding timing support is that the resulting netlists are faster than SmartOptNwbNt by 1.1%.

4.3 SmartOpt: Mapping with White Boxes and Timing

In this experiment, both white boxes and timing are enabled in SmartOpt. The result of applying ABC optimizations with both timing annotation and white box support leads to 8.3% fewer LUTs, 7.8% fewer flops and 2.1% shorter delay on average

compared to the XST reference flow without ABC. The runtime of ABC on large industrial designs is affordable and speaks for the scalability of the current implementation. White boxes provide 5% additional LUT count improvement and 7.6% additional flip-flop count improvement.

4.4 Effect of WireMap on LUT Merging

In this experiment, we show the effect of the ABC implementation of WireMap[5] on merging single-output LUTs into the dual-output 6-LUTs in Virtex-5. This LUT merging is invoked using a new MAP switch called “-lc area”. The reference flow is run with LUT merging activated, and compared with two ABC flows, one without WireMap and one with WireMap.

Adding “-lc area” to the reference flow reduces LUT count by 12.3%. For SmartOpt with WireMap, LUT count and flip-flop count is reduced even further to 22.8% and 7.6%, respectively. When running SmartOpt with WireMap disabled, LUT count reduction is degraded by 5.0%. This is due to WireMap creating a LUT distribution with a higher frequency of mergeable LUTs.

For SmartOpt without WireMap, 44% of all LUTs are 6-LUTs, while for SmartOpt with WireMap, the number of 6-LUTs is reduced by more than 12%. There is no change in 5-LUTs. In contrast, the ratios of 2-, 3-, and 4-LUTs are increased by 4%, 5%, and 3%. A higher ratio of smaller LUTs increases opportunities to reduce area utilization via merging.

5. CONCLUSIONS AND FUTURE WORK

This paper describes the implementation of a framework that allows industrial tools to leverage the scalable synthesis algorithms recently produced by academia. A key contribution is the creation of a netlist model that allows for passing device-specific timing information, as well as logic functions and user constraints to external synthesis tools. This model was applied to

the ABC logic synthesis and verification system from UC Berkeley. The results show that, with the new netlist model, excellent results obtained on academic circuits can also be obtained on large industrial designs.

In particular, we observed significant improvements in both design area and design performance with reasonable runtimes.

6. ACKNOWLEDGEMENTS

The authors wish to thank David Nguyen Van Mau, and Yassine Rjmati of Tiempo for their feedback and suggestions in conceiving and implementing this work.

7. REFERENCES

- [1] Altera. *Stratix III Device Handbook*, http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf
- [2] Xilinx *Virtex-5 Product Table*. http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/v5product_table.pdf
- [3] *Berkeley Logic Interchange Format (BLIF)*, <http://vlsi.colorado.edu/~vis/blif.ps>
- [4] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*, Release 61225. <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [5] S. Jang, B. Chan, K. Chung, and A. Mishchenko, “WireMap: FPGA technology mapping for improved routability”, *Proc. FPGA’08*.
- [6] J. Pistorius, M. Hutton, A. Mishchenko, and R. Brayton, “Benchmarking method and designs targeting logic synthesis for FPGAs,” *Proc. IWLS ’07*, pp. 230-237.
- [7] *Xilinx Constraints Guide*. <http://toolbox.xilinx.com/docsan/xilinx10/books/docs/cgd/cgd.pdf>
- [8] Xilinx. “Achieving higher system performance with the Virtex-5 family of FPGAs” (white paper). <http://direct.xilinx.com/bvdocs/whitepapers/wp245.pdf>

Table 4.1: SmartOpt results

Circuit	Reference			SmartOptNwbNt			SmartOptNwb			SmartOpt		
	LUT	FF	PRD	LUT	FF	PRD	LUT	FF	PRD	LUT	FF	PRD
Ex1	32000	14450	16.29	31193	14434	17.18	31134	14434	17.43	31209	14402	17.67
Ex2	17551	10733	9.21	17274	10733	9.88	16989	10733	9.86	16895	10677	11.17
Ex3	43609	35649	8.58	43632	35647	8.44	43179	35647	8.29	33928	19055	8.98
Ex4	30987	13235	15.78	30772	13204	13.52	30683	13203	15.27	21581	9011	11.96
Ex5	44644	14446	23.35	45159	14445	24.29	44059	14445	23.96	44030	14447	22.51
Ex6	29381	26626	9.49	28993	26630	10.33	29028	26630	10.27	27174	24197	9.62
Ex7	66839	31389	22.44	65879	31388	19.41	65312	31388	19.27	65233	31239	18.10
Ex8	19472	21326	6.97	19068	21300	7.74	19083	21300	6.89	18408	20931	7.50
Ex9	34330	17663	12.23	32713	17662	12.75	32484	17662	12.05	32581	17578	11.82
Ex10	18412	9364	9.16	18314	9348	9.88	18307	9348	10.06	18165	9276	9.25
Ex11	50670	24415	11.37	48292	24374	12.16	47733	24375	12.09	45300	23132	12.30
Ex12	25699	23436	5.55	25071	22691	5.42	25084	22672	5.70	23814	21557	5.87
Ex13	44991	19230	11.97	43731	19224	11.33	43467	19226	11.56	43283	19178	10.88
Ex14	69223	36597	12.89	60201	36377	12.45	60033	36395	11.83	59339	36326	10.12
Ex15	48655	35478	15.99	45565	35502	17.49	45681	35502	15.43	45604	35406	16.27
Ex16	47356	22774	15.38	44200	22694	13.08	43913	22709	12.14	41009	21166	11.91
Ex17	17036	12374	10.90	16692	12373	11.18	16711	12373	10.99	16741	12371	11.93
Ex18	26275	30720	9.69	26275	30703	10.33	26200	30703	10.82	24823	29382	10.59
Ex19	22889	23378	10.12	23115	23369	9.67	23079	23369	9.72	22283	21905	9.92
Ex20	26586	27514	10.00	26816	27504	8.10	26548	27504	8.08	23592	23783	10.02
Geomean	32806	20879	11.62	31970	20826	11.54	31798	20826	11.420	30096	19244	11.38
Ratio1	1	1	1	0.975	0.997	0.993	0.969	0.997	0.983	0.917	0.922	0.979
Ratio2				1	1	1	0.995	1.000	0.989	0.941	0.924	0.985
Ratio3							1	1	1	0.946	0.924	0.996