Benchmarking Method and Designs Targeting Logic Synthesis for FPGAs

Joachim Pistorius, Mike Hutton
Altera Corp.
101 Innovation Drive
San Jose, CA 95134
{jpistori,mhutton}@altera.com

Alan Mishchenko, Robert Brayton
EECS Department
University of California
Berkeley, CA 94720
{alanmi,brayton}@eecs.berkeley.edu

Abstract

A new set of benchmark designs is presented together with a reference experiment flow based on state of the art industrial and academic tools. Interfacing the tools became possible by extending the academic design specification format (BLIF) with the capability to represent blocks with unknown logic specification. This extension is required for handling large HDL designs containing memories and black-boxes. All tools and designs are made publicly available in an attempt to standardize on a benchmarking method for future tool evaluations and extensions to placement, routing, and other CAD algorithms.

1. Introduction

Benchmark designs are the basis for the performance evaluation of today's EDA tools. The most commonly employed method to verify the competitiveness of a new tool consists of applying this tool to a set of benchmark designs in a given experimental setting. The experimental results are then compared to those obtained by applying a comparable state-of-the-art tool to the same set of designs. This method implies that the quality of a tool's evaluation is only as good as the designs that are used for benchmarking and only as meaningful as the reference results obtained by the tool it is compared with.

In recent years there have been several initiatives for assembling big and realistic designs to improve the quality of those experimental evaluations. Especially physical design tools such as partitioning, placement, and routing have benefited tremendously from those initiatives which include not only the generation of synthetic benchmark designs [15], [16], [17], [18], [19], [28] and the collection of real industrial designs in the different versions of the ISPD benchmark sets [1], [3], [31], but also the recent development and improvement of very competitive academic placement tools, such as APlace, Capo, Dragon2005, FastPlace, FengShui, Kraftwerk, mPL, mFAR, and NTUPlace. These tools all competed at the ISPD 2005 placement contest [20] which was repeated in 2006. A growing traction for the improvement of routing tools is expected to come from the global routing contest organized at the International Symposium on Physical Design (ISPD 2007) this year.

The logic synthesis domain has also received more attention in recent years with the introduction of a new benchmark design set at IWLS 2005 [2]. This benchmark set contains the MCNC designs [32], the ITC'99 designs [10], the OpenCores designs [27], and a few other designs. The ITC'99 benchmark set consists of small VHDL designs collected from the internet and from class work. Bigger designs were generated by stitching together smaller designs [12]. The OpenCores designs are a collection of IP cores released into the public domain.

The lack of large benchmark designs is often attributed to confidentiality concerns in industry. However, it is often more of a design flow restriction. Most designs with more than a couple thousand gates have multi-entity hierarchies and contain memories and other "hard blocks", or adder/multiplier functionality synthesized to macros in ASIC flows or mapped to dedicated circuitry in FPGAs. By supporting these elements in a benchmarking flow, numerous large FPGA-targeted designs can become available to the CAD tool designer as benchmarks.

The goal of this paper is to provide an improved benchmarking methodology that consists of a design set together with a reference compilation flow and experimental results for evaluating logic synthesis and technology mapping algorithms. Our initial design set consists of 8 large designs each containing at least 10,000 4-LUTs. The intention is to broaden the content of the design set by regularly adding new and diverse designs and extending the number of designs over the next few months.

Having large designs in the benchmark set allows us to measure runtime and memory usage in addition to the traditionally measured properties of new tools such as area, delay, and power. It also enables us to follow new trends in design style and characteristics. For instance we have found that a lot of modern designs are heavily pipelined which changes the ratio of combinational logic and registers. More recent designs also tend to be bigger. An increased design size and complexity favors the use and re-use of intellectual property (IP) cores which has gained momentum in recent years. The I/O bandwidth also increases with the design size leading to more serial interfaces which in turn require more on-chip memory.

Our reference flow supports any RTL design written in synthesizable VHDL, Verilog, or SystemVerilog amongst others. It allows for easy evaluation of new synthesis tools by adopting the widely used Berkeley Logic Interchange Format (BLIF) [8] as the basis language and by extending the concept of hierarchy in BLIF with the capability to represent unspecified modules. This extension is essential for supporting today's designs containing numerous macros that cannot be represented in the traditional BLIF format.

We show the typical scenario of using the extracted hierarchical BLIF specifications by running them through a logic optimization flow in the academic synthesis and verification tool ABC [7]. The flow includes fast logic synthesis based on AIG rewriting [23] and AIG resubstitution [22], FPGA mapping into K-LUTs [25][26] and combinational equivalence checking [24] based on structural analysis, simulation, and SAT.

Finally, all designs used in our experiments are available for download [33] in RTL format and in several types of hierarchical BLIF. Since the tools used in this paper, Quartus [5] and ABC [7], are also available online, an interested reader can easily reproduce our results.

In the next section, we give an overview of BLIF and extend it with the notion of unspecified modules. Section 3 describes how hierarchical BLIF netlists can be generated from RTL design formats. Section 4 contains the description of the different benchmark designs followed by the reference experiment flow in section 6, and reference results in section 7. We conclude with an overview of future work.

2. Hierarchical BLIF Format

Berkeley Logic Interchange Format (BLIF) [8] is used to represent combinational and sequential logic networks used in academic logic synthesis and verification tools. BLIF became popular with SIS [29] and is currently supported in VIS, MVSIS, and most recently in ABC [7]. BLIF is also supported by the academic physical design tool VPR [9].

A complete description of the BLIF syntax and semantics can be found in the format manual [8]. Here we briefly review the hierarchy specification in BLIF. Each module instantiation is represented by a *.subckt* directive. The directive is followed by the name of the module that is instantiated and a space-separated list of pairs of formal and actual inputs. Each module instantiated at least once should be defined in the current file.

The hierarchical BLIF representation is illustrated in Figure 1 by a two-input adder generated using command "gen –a" in ABC. The main module "Adder02" has one logic node and two instances of the full adder "FA".

```
 .model Adder02
 .inputs a00 a01 b00 b01
 .outputs y00 y01 y02
 .subckt  FA  a=a00  b=b00  cin=c  s=y00
cout=00
 .subckt  FA  a=a01  b=b01  cin=00  s=y01
cout=y02
 .names c
  0
 .end

 .model FA
 .inputs a b cin
 .outputs s cout
 .names a b k
 10 1
 01 1
 .names k cin s
 10 1
 01 1
 .names a b cin cout
 11- 1
 1-1 1
 -11 1
 .end
```

Figure 1: Example of hierarchical BLIF

```
 .model Adder02
 .inputs a00 a01 b00 b01
 .outputs y00 y01 y02
 .subckt  FA  a=a00  b=b00  cin=c  s=y00
cout=00
 .subckt  FA  a=a01  b=b01  cin=00  s=y01
cout=y02
 .names c
  0
 .end

 .model FA
 .inputs a b cin
 .outputs s cout
 .blackbox
 .end
```

Figure 2: Hierarchical BLIF with a black-box

To enable representation of modules whose logic specification is not available, we introduce directive *.blackbox*, which is placed inside the description of a module. If the directive *.blackbox* is present, specification of logic functions cannot be used. This is illustrated in Figure 2 where the full-adder is represented as a module without logic representation. The *.blackbox* directive is essential for reading the synthesized BLIF netlist back into Quartus II and replacing un-synthesizable constructs by the original modules. This feature is planned for future releases.

3. Generating Hierarchical BLIF Netlists

The Quartus II software environment [5] and in particular the Quartus II Integrated Synthesis tool (QIS) is used as the front end of the reference flow. It parses, translates, and pre-processes behavioral logic written in VHDL, Verilog, or SystemVerilog[1] and generates hierarchical BLIF netlists. A commercial tool is essentially required at the front end to allow for arbitrary HDL designs to be parsed and translated. Note that this is a very challenging task, because the tool needs to handle all different kinds of design styles and ever changing rules and standards.

The Quartus II Integrated Synthesis tool processes the RTL description of a design by performing three distinguished compilation steps as outlined in Figure 3. A hierarchical BLIF output can occur after each of the three processing stages using specific control settings to the tool that are not part of the Quartus II release documentation and hence not available through the graphical user interface. Those control settings are typically introduced as assignments with the help of Tcl commands.

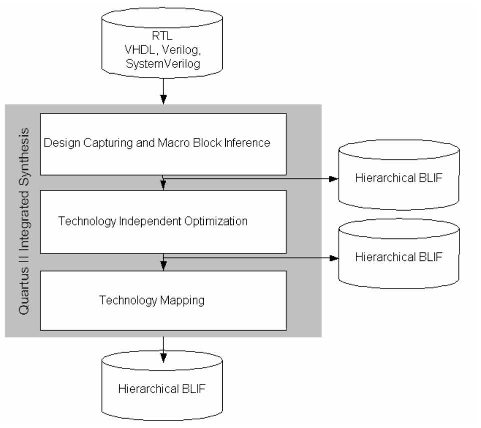

RTL
VHDL, Verilog, SystemVerilog
Quartus II Integrated Synthesis
Design Capturing and Macro Block Inference
Technology Independent Optimization
Technology Mapping
Hierarchical BLIF
Hierarchical BLIF
Hierarchical BLIF

Figure 3: Generating Hierarchical BLIF Netlists

In a first step, QIS parses the design source files and checks for syntax and semantic errors. It also identifies constructs in the source code that can be implemented using hard macro blocks available on the targeted device such as RAM or DSP blocks and makes the appropriate functional inference. All logic hierarchies inside the design besides from macro blocks are removed by default during this step as well if no settings have been made to preserve hierarchy. We also need to prevent the inference of adders with carry-chains since BLIF doesn't have the syntax constructs to support carry-chains. A first hierarchical BLIF netlist can be generated after this initial step by adding the following Tcl command to the Quartus Settings File (.qsf file) in the project directory.

```
 set_global_assignment -name INI_VARS
“no_add_ops=on; dump_blif_before_optimize=on"
```

During the second step the tool performs technology independent optimization. Another version of the hierarchical BLIF netlist can be generated after this step with the following commands in the .qsf file.

```
 set_global_assignment -name INI_VARS
“no_add_ops=on; dump_blif_after_optimize=on"
```

Finally during the last step QIS performs technology mapping to lookup tables (LUTs) for all combinational logic. The hierarchical netlist after this step is generated with the following Tcl command.

```
 set_global_assignment -name INI_VARS
“no_add_ops=on; dump_blif_after_lut_map=on"
```

Other flows to generate a hierarchical BLIF netlist are available [4] as well as alternative experimental flows. In particular, QIS can read a technology mapped design, decompose LUTs into combinational logic, and perform technology independent optimization followed by technology mapping. Another flow has QIS write a hierarchical BLIF netlist before technology independent optimization. This netlist is then processed by the AIG-based synthesis in ABC, and read back into QIS for technology mapping. The reference results for those flows can be found on the web page where the designs are available.

4. Handling Hierarchical BLIF in ABC

The BLIF parser in ABC was extended to support the *.blackbox* directive described in section 2. After the network is parsed, the hierarchy modules with given logic specification are flattened while the modules without such specification (black-boxes) are handled by adding new primary inputs and primary outputs with the same names as the actual names of the black-box instances.

As a result of this processing of the hierarchy, the logic network used for synthesis and verification in ABC contains only combinational logic nodes, and possibly sequential elements, but it does not contain black-boxes. The result of logic optimization and technology mapping produced by ABC can be written out in BLIF or Verilog in two ways: (a) as a hierarchical network with black-boxes, which have the same input/output names as in the initial network, or (b) as a logic network with black-boxes with additional primary inputs and outputs.

[1] Other languages historically supported by Quartus II can be used to generate hierarchical BLIF netlists as well.

Table 1: Characteristics of Benchmark Designs

Design Name	# 4-LUTs	# FF	# 9-bit MAC	# RAM bits	# I/O pins	Type	Origin
ucsb_152_tap_fir	10199	9449	0	3092	42	FIR Filter	[20]
umass_weather	11598	7189	8	2604404	655	Weather Radar	[11]
oc_des_des3perf	11742	5850	0	2744	298	Encryption Core	[4]
ava	12254	2593	0	0	85	AVA Decoder	[30]
sudoku_check	17188	5919	0	9280	54	Sudoku Solver	Altera
top_rs_decode	22796	5940	13	59792	755	Reed Solomon	[6]
uoft_raytracer	23673	13986	135	57254	788	Ray Tracing Engine	[13]
uoft_stereo_vision	57138	47341	290	202822	389	Stereo Vision	[12]

5. Benchmark Designs

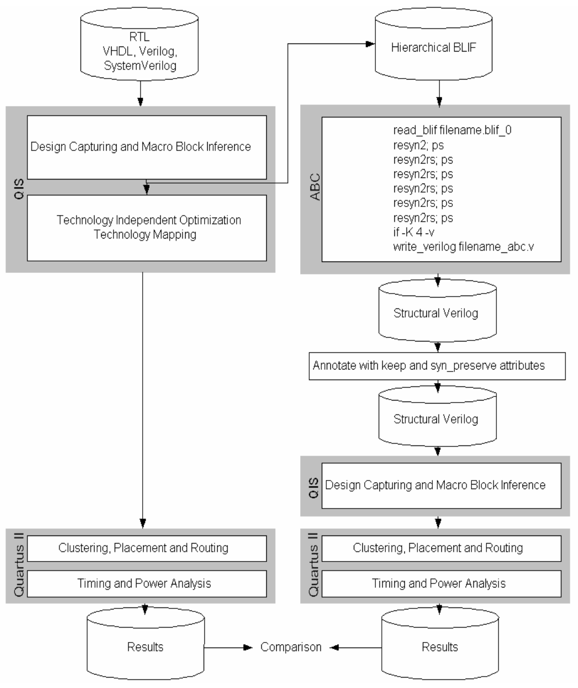
Our benchmark design set aims at providing realistic designs that challenge synthesis and technology mapping tools in the same way as real industrial designs would challenge a commercial synthesis tool. Therefore, we include only designs that contain at least 10K 4LUTs after a successful compilation with Quartus II Integrated Synthesis version 7.1 for implementation in Cyclone II devices. Because of this selective criterion we were able to use only the two biggest designs from the previous QUIP benchmark design set [4]. The other designs are from various different sources. The characteristics of the benchmark designs as well as their origins and the results of the Quartus II compilation are displayed in table 1.

6. Reference Experiment Flow

In this section we present a reference experiment flow that leads to meaningful comparisons when used together with the benchmark designs from the previous section. ABC is used here as an example for a synthesis tool under evaluation. This flow is also used to generate the reference results presented in the next section.

The RTL version of the benchmark designs are read using QIS. The targeted device family is Cyclone II. Cyclone II has been chosen, because it is a fairly simple architecture compared to Stratix III or even Stratix II in the sense that it doesn't contain as many and as complex hard macro blocks. This implies that QIS will limit the inference of macro blocks to the available hard blocks and their features in this particular device family. For instance, Cyclone II MAC blocks don't have support for rounding and saturation. Any multiplication that uses this feature will be implemented using combinational logic. Note that this is a desirable property since our goal is to test logic synthesis and technology mapping tools and not the quality of the macro block inference tools. Even though Cyclone II has a less complex architecture than the Stratix families, it is still a state-of-the-art FPGA architecture that allows for useful comparisons in an industrial setting.

QIS flattens all hierarchies inside the design, identifies parts of the RTL code that can be implemented in hard macro blocks and executes the inference of those blocks. In the reference experiment flow, it generates a hierarchical BLIF netlist before technology independent optimization as described in section 3.


RTL
VHDL, Verilog, SystemVerilog
Hierarchical BLIF
QIS
Design Capturing and Macro Block Inference
Technology Independent Optimization
Technology Mapping
ABC
Structural Verilog
Annotate with keep and syn_preserve attributes
Structural Verilog
QIS
Design Capturing and Macro Block Inference
Quartus II
Clustering, Placement and Routing
Timing and Power Analysis
Quartus II
Clustering, Placement and Routing
Timing and Power Analysis
Results
Comparison
Results

```
read_blif filename.blif_0
resyn2; ps
resyn2rs; ps
resyn2rs; ps
resyn2rs; ps
resyn2rs; ps
resyn2rs; ps
if -K 4 -v
write_verilog filename_abc.v
```

Figure 4: Experiment Flow

After finishing logic synthesis and technology mapping, using ABC, it is possible to read the resulting netlists back into Quartus II for placement and routing onto the target device as shown in Figure 4. This step allows for analyzing the impact the logic synthesis and technology mapping had on fitting and routing congestion. It also enables the use of a power analysis tool to explore dynamic power consumption as well as the use of a timing analysis tool to explore the effect those algorithms had on post placement and routing delays as opposed to unit delays that are usually observed. Since Quartus II requires synthesis to be run

Table 2: Results of Technology-Independent Synthesis

Design	Original		resyn2		resyn2rs		Runtime, sec				
	Lev	Node	Lev	Node	Lev	Node	read	res2	res2rs	map	cec
ucsb_152_tap_fir – 0	86	68489	79	42036	79	40069	0.87	16.05	22.54	2.63	6.41
ucsb_152_tap_fir – 1	57	50784	57	41861	57	37401	0.87	6.52	11.34	2.54	6.91
ucsb_152_tap_fir – 2	108	74453	56	43103	56	39073	0.92	9.02	13.75	2.62	7.32
umass_weather – 0	169	65091	168	42138	168	41442	0.83	12.02	21.65	2.53	8.40
umass_weather – 1	113	55387	113	44491	113	43164	0.84	7.12	13.98	2.45	7.87
umass_weather – 2	151	86225	115	45674	115	44191	0.86	10.00	17.47	3.08	10.71
oc_des_des3perf – 0	22	100363	16	70014	16	65808	1.01	23.88	56.47	4.60	8.61
oc_des_des3perf – 1	22	100363	16	69923	16	65808	1.03	24.54	57.76	4.78	8.89
oc_des_des3perf – 2	24	116203	19	71653	19	67838	1.03	22.80	52.27	4.92	10.15
ava – 0	254	49629	243	29211	217	28126	0.42	5.70	11.03	1.68	3.42
ava – 1	75	47523	73	28895	73	27962	0.41	6.09	8.37	1.65	3.34
ava – 2	113	68154	70	24282	70	22775	0.37	5.19	6.58	1.62	5.87
sudoku_check – 0	729	81313	32	50960	31	48237	0.82	14.45	27.71	3.11	14.59
sudoku_check – 1	45	53267	34	41692	33	40849	0.75	11.91	26.19	2.74	13.22
sudoku_check – 2	51	80027	37	42738	35	41648	0.74	14.02	26.78	2.89	21.40
top_rs_decode – 0	217	170565	154	125726	154	123938	1.74	52.12	87.18	10.01	27.76
top_rs_decode – 1	99	140023	98	123596	98	121859	1.75	49.72	82.30	9.67	27.19
top_rs_decode – 2	195	199469	141	125102	141	123779	1.79	55.56	85.92	9.67	42.12
uoft_raytracer – 0	267	148290	231	103323	231	100087	2.05	39.03	65.09	7.69	101.12
uoft_raytracer – 1	184	120258	169	102960	167	97244	2.02	23.20	39.63	6.92	136.02
uoft_raytracer – 2	297	178146	217	102985	217	99822	2.05	37.42	58.29	8.37	110.30
uoft_stereo_vision – 0	159	370587	149	223575	149	213375	2.84	88.69	116.79	17.76	47.60
uoft_stereo_vision – 1	108	283927	108	221004	108	205201	2.78	59.09	81.52	11.86	35.45
uoft_stereo_vision – 2	200	369177	150	218825	150	208532	2.79	78.76	107.91	12.51	47.38
Ratio	1.00	1.00	0.80	0.66	0.79	0.63	1.00	18.91	32.63	4.13	18.78

to be run before placement and routing, we need to annotate the Verilog netlist generated by ABC such that QIS doesn't process the Verilog input again. Any logic synthesis tool can be evaluated in this flow by substituting ABC as long as this tool can read hierarchical BLIF and write structural Verilog.

Note that the current Quartus II version doesn't allow for reading back the black box content. This is planned for a future release.

Besides the reference flow that is described above there are a whole lot of other flows users can explore:

- QIS performs technology independent optimization followed by technology mapping with ABC or any other synthesis tool.
- QIS performs technology mapping followed by resynthesis with ABC or any other synthesis tool under evaluation.
- Other device families such as Stratix II can be targeted allowing for 6 LUT mapping and more complex hard macro blocks.
- Optimization can target speed or area depending on the synthesis settings.
- ABC can use all kinds of command combinations in different scripts.
- Any combination of the points above is a possible compilation flow.

7. Reference Results

In this section we apply ABC to the hierarchical designs produced by Quartus II. Table 2 shows the results of technology-independent synthesis and the runtimes. Table 3 shows the results of technology mapping while Table 4 shows the outcome of applying the experiment flow in Figure 4 to design *ava*.

In Table 2, column "Name" lists the benchmark name. Columns "Lev" and "Node" show the number of logic levels and nodes in the AIG. The first section of Table 2 characterizes the original network: The next two sections show the results of AIG rewriting using script "resyn2" [23] followed by script "resyn2rs" [22]. In the last section of Table 2, column "read" is runtime of reading the hierarchical BLIF and converting it into an AIG by structural hashing. Columns "res2" and "res2rs" are the runtimes of scripts "resyn2" and resyn2rs". Column "map" is the runtime of technology mapping with 6-LUTs and 8 priority cuts [26]. Finally, column "cec" is the runtime of combinational equivalence checking after optimization and mapping [24].

In Table 3, columns "Depth", "Area", and "Nets" show the depth (longest combinational path measured in terms of LUTs), the number of LUTs, and the

Table 3: Results of Technology Mapping into 4-LUTs, 5-LUTs, and 6-LUTs

Design	4-LUT mapping			5-LUT mapping			6-LUT mapping		
	Depth	Area	Nets	Depth	Area	Nets	Depth	Area	Nets
ucsb_152_tap_fir – 0	25	15673	39967	14	13355	41101	13	12976	41475
ucsb_152_tap_fir – 1	19	15627	40759	14	12918	39056	11	13268	43192
ucsb_152_tap_fir – 2	19	17152	45976	14	13661	42225	11	13694	45814
ucsb_152_tap_fir – 3	26	14831	37019	14	13158	39428	13	12653	39853
umass_weather – 0	55	13187	41720	29	12287	44383	28	10349	42831
umass_weather – 1	38	15471	49066	29	12272	43878	23	11833	48210
umass_weather – 2	38	15556	48873	29	13531	48187	24	11884	49433
umass_weather – 3	38	16543	50048	29	14507	52848	24	12435	50548
oc_des_des3perf – 0	5	24897	79644	5	14949	52720	3	9746	26133
oc_des_des3perf – 1	5	24904	79784	5	14952	52757	3	9746	26133
oc_des_des3perf – 2	6	20352	65254	4	12874	42994	3	10638	31884
oc_des_des3perf – 3	5	16038	50324	4	12197	38655	3	10946	36794
ava – 0	73	9334	33727	55	7043	31682	44	6234	31353
ava – 1	25	9321	33673	20	7077	31818	16	6209	31415
ava – 2	24	7704	27153	19	5657	24745	15	4984	24375
ava – 3	30	14348	52542	23	11333	48819	19	9434	47357
sudoku_check – 0	10	24194	83315	9	20452	81944	7	16989	75802
sudoku_check – 1	11	20442	69150	9	17435	67674	7	14133	62905
sudoku_check – 2	11	20574	69324	9	17272	66893	7	14121	62891
sudoku_check – 3	11	19683	66092	10	17791	65210	8	13719	58325
top_rs_decode – 0	52	31403	111364	39	25103	108973	32	21262	106533
top_rs_decode – 1	33	30512	107963	26	25422	109383	21	21066	106214
top_rs_decode – 2	43	31007	111259	28	25164	108194	24	21363	107235
top_rs_decode – 3	48	31001	108092	32	27386	110623	26	22978	115575
uoft_raytracer – 0	69	33641	106172	42	28142	107038	36	26042	107702
uoft_raytracer – 1	54	34687	108742	41	27402	103787	33	26736	110665
uoft_raytracer – 2	69	33423	104755	41	28251	106056	35	26011	105054
uoft_raytracer – 3	72	32321	98450	44	28530	104480	39	25612	101653
uoft_stereo_vision – 0	50	77864	214151	27	70453	221515	26	65164	226201
uoft_stereo_vision – 1	36	77155	211501	26	68938	221170	22	66013	229362
uoft_stereo_vision – 2	50	77018	211902	28	69614	220926	26	64772	225270
uoft_stereo_vision – 3	47	76855	212021	27	68999	218183	25	64852	227167
Ratio	1.00	1.00	1.00	0.73	0.82	0.95	0.59	0.72	0.93

number of nets (the sum of the fanins of all LUTs) after technology mapping with 4-LUTs, 5-LUTs, and 6-LUTs [25][26] for the different versions of the BLIF output. Versions "-0", "-1", and "-2" of the designs are written by Quartus before technology-independent synthesis, before technology mapping and after technology mapping into 4-LUTs, respectively. All these versions were optimized in ABC by applying script "resyn2" followed by "resyn2rs" as shown in Table 2. Finally, version "-3" is the same as version "-2" without optimization in ABC. This version was mapped along with the optimized versions to compare mapping in Quartus II with mapping in ABC. Table 4 shows the effect that different synthesis tools have on the placed and routed design by using the experiment flow illustrated in Figure 4. The chosen target device was Cyclone II. QIS aims at maximizing speed during the technology-independent optimization and technology mapping which resulted in an increased number of logic elements, fewer logic levels on the critical path and hence a better critical path delay, but worse dynamic power consumption. In contrast ABC targeted area optimization which resulted in fewer logic elements and lower dynamic power, but more logic levels on the critical path and hence a longer worst case delay.

Table 4: Results for Quartus II and ABC on Design ava

Synthesis Tool	Logic Elements	Logic Levels	Delay	Dynamic Power
QIS	12528	29	13.8ns	120mW
ABC	9343	47	21.2ns	77mW
Ratio	1.34	1.62	1.54	0.64

Table 4 shows the effect that different synthesis tools have on the placed and routed design by using the experiment flow illustrated in Figure 4. The chosen target device was Cyclone II. QIS aims at maximizing speed during the technology-independent optimization and technology mapping which resulted in an increased number of logic elements, fewer logic levels on the critical path and hence a better critical path delay, but worse dynamic power consumption. In contrast ABC targeted area optimization which resulted in fewer logic elements and lower dynamic power, but more logic levels on the critical path and hence a longer worst case delay.

Further analysis using the Quartus II TimeQuest Timing Analyzer tool [5] reveals that the slack (difference between required delay and actual delay) for the netlist generated by QIS has a sharp decline on the extreme left towards a positive slack of 6.2ns as shown in Figure 5 while the slack for the netlist generated by ABC has a long "tail" that starts at a slack of 10ns and reaches all the way to -2.154ns with 131 paths having a negative slack and hence violating timing constraints as shown in Figure 6. The clock constraint was arbitrarily set to 20ns for both netlists.

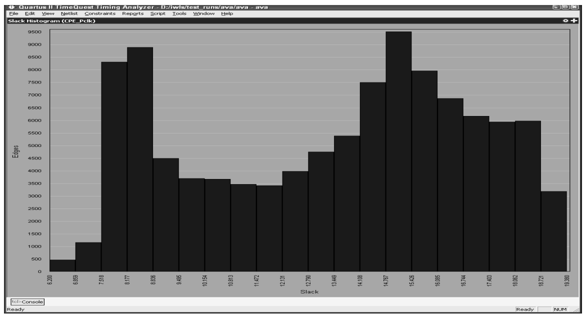


Figure 5: Slack Histogram for QIS

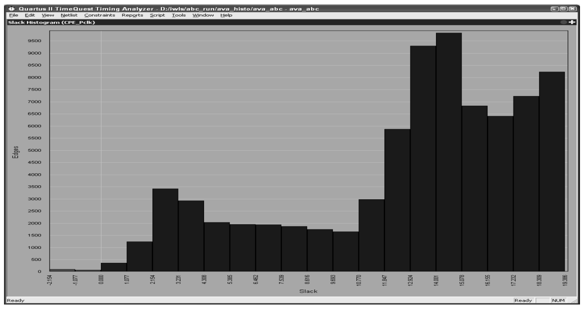


Figure 6: Slack Histogram for ABC

This can be explained with the higher number of worst case logic levels in the ABC generated netlist in comparison to the QIS generated netlist as shown in Table 4. In fact, the ABC generated netlist has 136 paths with 47 logic levels while the QIS generated netlist has only 72 paths with 29 logic levels. One could also draw the conclusion that the placement tool is able to handle a short "tail" of paths with higher logic levels well by placing them all uniformly as illustrated in Figure 5 by the steep decline towards the left. However, this particular placement tool couldn't handle the longer "tail" with the 131 outlier paths.

The goal of this experiment was not only to examine the effects synthesis had on placement and routing, but also to illustrate the opportunities this benchmarking flow provides for evaluating and improving logic synthesis tools by analyzing causes for differences from different angles and ultimately learn about specific weaknesses of the tool under evaluation. Here, it implied to investigate why ABC created those long paths. The reason is most likely that ABC doesn't have any restrictions on consuming slack along any path during area recovery.

With regard to benchmarking the performance of a synthesis tool, this experiment illustrates that it is important to consider the effects the tool has on the feasibility of placement, routing congestion, timing, and power as well as the overall runtime in addition to the traditionally observed number of LUTs and unit delay. This particular experiment also shows one of the pitfalls in benchmarking. The comparison seemed to be straightforward, but it was not fair because ABC targeted area improvement and QIS targeted speed optimization. The experiment also illustrates that there isn't a single result better than others, but a wide range of good results where area, speed, and power can be traded-off against each other leaving it to the user to choose the best solution to a given problem.

8. Conclusions and Future Work

In this paper we have presented an enhanced version of hierarchical BLIF which supports modules without logic specification (black-boxes). The black-boxes enable the handling of state-of-the-art designs that often contain a few macro blocks that cannot be described by the traditional hierarchical BLIF.

The capability to represent black-boxes allowed us to collect 8 large designs each containing more than 10K 4-LUTs after mapping to the Cyclone II architecture using QIS. We also introduced support for hierarchical BLIF into QIS and ABC and developed a reference experiment flow for benchmarking logic synthesis and technology mapping. This experiment flow was used to generate reference results that researchers can use to compare their logic synthesis or technology mapping tool against.

Our goal is to increase the number of public benchmark designs over time and to keep the designs in that set up to date as technology progresses to smaller technology nodes and larger design sizes.

In the future, we plan to increase the capabilities of hierarchical BLIF by introducing the concept of adders with carry-chains. We also plan to create a reference flow that targets academic VPR [9]. This will be done either by adding hierarchical BLIF support to VPR or by improving the current Verilog netlist generated by

ABC by stitching black-boxes back into the post synthesis netlist.

To simplify the benchmarking methodology and to obtain the best results, we envision creating a tool that automatically generates different tool and flow combinations similar to Altera's DSE tool.

Finally, all designs presented in this paper are available for download in RTL format as well as in the 3 flavors of hierarchical BLIF together with the tools used in the reference flow on the web page [33].

9. Acknowledgements

The authors would like to thank Russ Tessier, Sriram Swaminathan, Lilian Atieno, Jonathan Rose, Josh Fender, Ahmad Darabiha, Ryan Kastner, Shahnam Mirzaei, Anup Hosangadi, Ulrich Brüning, David Slogsnat, Alexander Giese, Sven Kapferer, Ian Milton, and Meghal Varia for donating their designs to this benchmark set.

The first author would also like to thank Swati Pathak for implementing the original version of the BLIF writer in Quartus II.

References

[1] S. Adya, I. Markov, "ISPD'02 IBM Mixed-Size Placement Benchmarks", http://vlsicad.eecs.umich.edu/BK/ISPD02bench/.

[2] C. Albrecht, "IWLS 2005 Benchmarks", http://iwls.org/iwls2005/benchmarks.html.

[3] C. Alpert, "The ISPD'98 Circuit Benchmark Suite", http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html.

[4] Altera Corp., "Quartus II University Interface Program", www.altera.com/education/univ/research/unv-quip.html .

[5] Altera Corp., "Quartus II Software", www.altera.com/products/software/products/quartus2/qts-index.html .

[6] L. Atieno, J. Allen, D. Goeckel, R. Tessier, "An Adaptive Reed-Solomon Errors-and-Erasures Decoder", In *Proc. FPGA'06,* pp.150-158.

[7] Berkeley Logic Synthesis and Verification Group, University of California, Berkeley, "ABC: A System for Sequential Synthesis and Verification", www.eecs.berkeley.edu/~alanmi/abc.

[8] Berkeley Logic Synthesis and Verification Group, University of California, Berkeley, "Berkeley Logic Interchange Format (BLIF)", http://vlsi.colorado.edu/~vis/blif.ps .

[9] V. Betz, et al., "VPR: Versatile Packing, Placement, and Routing for FPGAs", Univ. of Toronto, 1997, www.eecg.toronto.edu/~vaughn/vpr/vpr.html.

[10] CAD Group, Politecnico de Torino, "ITC'99 Benchmarks" http://www.cad.polito.it/tools/itc99.html.

[11] Embedded Signal Processing Group, University of Massachusetts, Amherst, http://www.ecs.umass.edu/ece/tessier/rcg/ERC_web/index.html

[12] F. Corno, M. Sonza Reorda, G. Squillero: "RT-Level ITC 99 Benchmarks and First ATPG Results", In *IEEE Design & Test of Computers*, pp. 44-53, Jul.,Aug. 2000.

[13] A. Darabiha, J. Rose, W.J. MacLean, "Video-Rate Stereo Depth Measurement on Programmable Hardware", In *Proc. Int. Conf. on Computer Vision and Pattern Recognition*, 2003.

[14] J. Fender, J. Rose, "A High-Speed Ray Tracing Engine Built on a Field Programmable Engin", In *Proc. FPT*, pp. 188-195, 2003.

[15] D. Ghosh, N. Kapur, J. Harlow, F. Brglez "Synthesis of Wiring Signature-Invariant Equivalence Class Circuit Mutants and Applications to Benchmarking", In *Proc. DATE*'98., pp. 656-663.

[16] D. Grant, S. Chin, G. Lemieux, "Semi-Synthetic Circuit Generation Using Graph Monomorphism for Testing Incremental Placement and Incremental Routing Tools", In *Proc. FPL,* pp. 725-728, 2006.

[17] D. Grant, G. Lemieux, "Perturber: Semi-Synthetic Circuit Generation Using Ancestor Control for Testing Incremental Place and Route", In *Proc.FPT*, pp. 189-195, 2006.

[18] M. Hutton, J.P. Grossman, J. Rose, D. Corneil, "Characterization and Parameterized Random Generation of Digital Circuits", In *Proc. DAC'96*, pp. 94-99, 1996.

[19] M. Hutton, J. Rose, D. Corneil, "Automatic Generation of Synthetic Sequential Benchmark Circuits", In *IEEE Trans. On Computer-Aided Design, vol. 21, no.8*, pp. 928-940, 2002.

[20] "ISPD 2005 Placement Contest", http://www.sigda.org/ispd2005/contest.htm.

[21] S. Mirzaei, A. Hosangadi, R. Kastner "FPGA Implementation of High Speed FIR Filters Using Add and Shift Method", In *Proc. ICCD,* 2006.

[22] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure", In *Proc. IWLS '06*, pp. 15-22. http://www.eecs.berkeley.edu/~alanmi/publications/2006/iwls06_sls.pdf

[23] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", In *Proc. DAC '06*, pp. 532-536. http://www.eecs.berkeley.edu/~alanmi/publications/2006/dac06_rwr.pdf

[24] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to combinational equivalence checking", In *Proc. ICCAD '06*, pp. 836-843. http://www.eecs.berkeley.edu/~alanmi/publications/2006/iccad06_cec.pdf

[25] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs". In *IEEE Trans. On Computer-Aided Design, vol. 26, no.2*, pp. 240-253, 2007. http://www.eecs.berkeley.edu/~alanmi/publications/2006/tcad06_map.pdf

[26] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Mapping with priority cuts", Submitted to IWLS '07.

[27] "OpenCores benchmarks", www.opencores.org.

[28] J. Pistorius, E. Legai, M. Minoux, "PartGen : A Generator of Very Large Benchmarks for the Partitioning of FPGAs", In *IEEE Trans. On Computer-Aided Design, vol. 19, no. 11*, pp. 1314-1321, 2000.

[29] E. Sentovich, et al., "SIS: A System for Sequential Circuit Synthesis", Technical Report No. UCB/ERL M92/41. University of California, Berkeley, 1992.

[30] R. Tessier, et al., "A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder", In *IEEE Trans. On VLSI Systems, vol. 13, no. 4,* pp. 484-488, 2005.

[31] N. Viswanathan, C. Chu, "ISPD'04 IBM Standard-Cell Benchmarks with Pads", www.public.iastate.edu/~nataraj/ISPD04_Bench.html.

[32] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0", Technical Report MCNC P.O. Box 12889, Research Triangle Park, NC 27709, 1991.

[33] http://www.eecs.berkeley.edu/~alanmi/benchmarks/altera