Exploring FPGA Technology Mapping for Fracturable LUT Minimization

David Dickin, Lesley Shannon

School of Engineering Science, Simon Fraser University
8888 University Drive Burnaby, B.C., Canada
drdickin@sfu.ca
lshannon@sfu.ca

***Abstract*—Modern commercial Field-Programmable Gate Array (FPGA) architectures contain look-up-tables (LUTs) that can be "fractured" into two smaller LUTs. The potential of packing two LUTs into a space that could accommodate only one in traditional architectures complicates technology mapping's LUT minimization objective. Previous works introduced edge recovery techniques and the concept of LUT balancing, both of which produce mappings that pack into fewer fracturable LUTs. We combine these two ideas and evaluate their effectiveness for one commercial and four academic FPGA architectures, all of which contain fracturable LUTs. When used in conjunction, edge-recovery and LUT balancing yield a 9.0% to 15.8% reduction in fracturable LUT use, depending upon architectural constraints.**

I. Introduction

Technology mapping (tech-mapping) transforms a technology-independent logic network into a functionally equivalent netlist of primitive elements available for implementation on a target device. For a Field-Programmable Gate Array (FPGA), the logic primitive is a *look-up table* (LUT). An FPGA is fabricated with a limited number of LUTs. When performing tech-mapping for FPGAs, one objective is to minimize the number of LUTs in the mapping. Each LUT is restricted to a maximum of K inputs (a K-LUT), where K is defined by the FPGA's architecture. The *size* of a LUT is the number of inputs actually used. Any arbitrary Boolean function with up to K inputs can be implemented on a FPGA using a K-LUT.

Modern FPGAs feature structures that can function as either a single K-LUT or two (K-1)-LUTs with input sharing constraints. This type of structure is called a *fracturable* LUT (FLUT), since the K-LUT of a traditional architecture can be "fractured" into two smaller LUTs. Currently, tech-mapping tools produce netlists of LUTs, not FLUTs, and later stages of the tool flow "pack" the LUTs into FLUTs, either individually or in pairs. Thus, the number of FLUTs utilized is always less than or equal to the number of LUTs in the mapping, and minimizing the number of LUTs in a mapping does not necessarily minimize FLUT use. Previous works have shown that tech-mapping with the edge-recovery techniques of WireMap [1] and the concept of LUT balancing [2] have a positive effect upon the "packability" of a mapping, i.e. minimizing FLUT usage. WireMap is a technology mapper that reduces the number of wires in a mapping [1]. LUT balancing involves modifying cost functions such that LUTs with K inputs are undesirable [2]. Both approaches produce mappings with fewer size-K LUTs, which occupy an entire FLUT, and more LUTs with sizes less than K. Two LUTs with sizes less than K can potentially be packed together into a FLUT, decreasing the number of FLUTs required to pack a mapping.

In this work, we investigate the relative improvements obtained from both tech-mapping techniques and demonstrate that they are complementary. In addition, we examine the effects of applying LUT balancing to LUTs with K-1 inputs and demonstrate that this further reduces FLUT usage on FLUT architectures with tight input sharing constraints. We evaluate the success of a mapping by the reduction in FLUTs used after packing, placing, and routing the design. The previous works presenting WireMap [1] and LUT balancing [2] each evaluated their tech-mapping results using a single FLUT enabled commercial FPGA architecture. We consider four different FLUT enabled academic FPGA architectures and perform packing, placement, and routing using a version of VPR that includes AAPack[3]. Two of these academic architectures emulate the FLUT configurations found in the commercial FPGA architectures of the previous works. When these mapping techniques are used together, we observe a 9.0% to 15.8% average percent reduction in FLUTs, depending on the target architecture, compared to tech-mapping without WireMap or LUT balancing. In addition to the academic FPGA architectures, we use QUIP [4] to investigate the effects of these tech-mapping techniques when targeting an Altera Statix II device [5] and obtain a 12.4% reduction in FLUTs.

The remainder of the paper is structured as follows. Section II summarizes previous work on FPGA tech-mapping. Section III describes the FPGA architectures used in our experiments. Section IV presents the experimental results. Section V concludes the paper and outlines future work.

II. FPGA Technology Mapping

Tech-mapping a circuit to a FPGA is the process of converting all logic in the initial network into a functionally equivalent netlist of LUTs [6][7][8][9][10][11]. Minimizing first the *depth* and then the *area* are typical optimization goals of FPGA tech-mapping. The *depth* of a mapping is the

number of LUTs on the longest combinatorial path. Depth-optimal mapping algorithms exist [6][10][8], where depth is used to estimate the delay of a circuit. *Area* is measured as the number of LUTs in the mapped circuit, but LUT minimization is a nondeterministic polynomial-time hard (NP-hard) problem [12][7].

An FPGA mapping can be found by first converting the input network into an *AND-Inverter Graph* (AIG), comprised of only 2-input AND gates and inverters. Next, *K-feasible cuts* are generated for all AIG nodes. A *cut* is a set of nodes associated with a *root* node. The nodes in the cut are called *leaves*. A cut *covers* the root and all nodes on the paths between the root and the leaves, but not the leaves themselves. A cut is valid if all paths from the network inputs to the root node pass through one or more leaves. A cut with *K* or less leaves is *K-feasible* and can be implemented using a K-LUT. Once the K-feasible cuts have been enumerated, a number of the cuts are selected such that all nodes in the AIG are *covered*. The selected cuts become LUTs in the mapping.

A. Area Recovery Techniques

Performing a depth-optimal mapping leads to *logic replication*, where AIG nodes end up being covered by more than one LUT in the mapping. Once a depth-optimal mapping is found, cuts on non-critical paths can still be changed to minimize the number of LUTs in the mapping. Two cost functions for evaluating the area of a cut are *Area Flow* and *Exact Area* [8]. Both functions include a $Weight()$ function, which returns the base cost of a cut depending upon how many leaves the cut has (i.e. the size of LUT the cut would result in). $Weight()$ typically returns equal values for all cut sizes. The Exact Area of a cut, *c* is calculated by summing the $Weight()$ of all cuts added to the mapping as a result of including *c*. The Area Flow (AF) of *c* is given by:

$$AF = Weight(c) + \sum_{i} \frac{AF(BestCut(Leaf_i(c)))}{nEstFanouts(Leaf_i(c))} \quad (1)$$

where $BestCut(Leaf_i(c))$ is the best cut of the *i*-th leaf of *c* and $nEstFanouts(Leaf_i(c))$ is the estimated number of fanouts the *i*-th leaf of *c* will have in the mapping. If zero fanouts are estimated then one is used in Equation 1 to avoid dividing by zero.

B. WireMap and LUT Balancing

WireMap is a technology mapper that uses *edge recovery* techniques to reduce the number of wires in a mapped design [1]. Reducing the number of wires has a favourable effect upon routing complexity. In addition, WireMap produces mappings that pack into a smaller number of fracturable LUTs. This is due to the increased number of small-sized LUTs in the mapping, which are easier to pack. WireMap uses edge cost functions to decide between cuts with equal area costs.

LUT balancing modifies the cut selection cost functions (i.e. $Weight()$) in order to influence the LUT size distribution of a mapping [2]. Although implementation details are not given, it is reported that when LUT balancing is used to discourage the use of size-*K* cuts, mappings that pack into fewer FLUTs are produced [2].

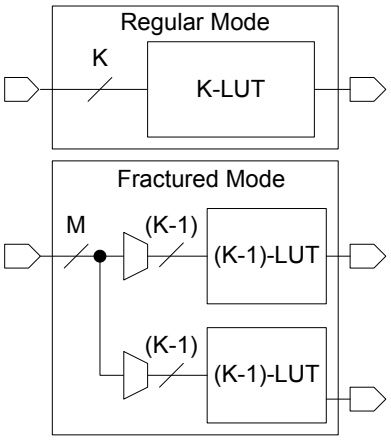


Fig. 1. The two modes of operation of a (K,M)-FLUT.

III. Fracturable LUT Architectures

A. Fracturable LUT Model

A FLUT has two modes of operation, *regular*, and *fractured*. While in regular mode, the FLUT acts as a K-LUT. In fractured mode the FLUT has *M* unique inputs, from which the inputs to the two (K-1) LUTs are selected. The two modes of operation of a generic FLUT are shown in Figure 1. The values of *K* and *M* define specific FLUT architectures. For the remainder of this work, we will refer to a particular FLUT architecture as a (K,M)-FLUT.

B. Academic Architectures

For our experiments, we created four FPGA architectures containing fracturable LUTs. The value of *M* is varied for each architecture, all other aspects of the architectures are the same. The four architectures have $K = 6$ ($K6$) and *M* values of 5, 6, 7, and 8 ($M5$, $M6$, $M7$, and $M8$). The $M5$ architecture is included to mimic the dual-output 6-LUT of a Xilinx Virtex 5 [13]. The $M8$ FLUT is similar to the Adaptive Logic Module (ALM) found in an Altera Stratix II [5]. However, our FLUT models are only meant to approximate, not replicate, these commercial structures.

A generic version of the *Logic Element* used in our architectures is shown in Figure 2. The LE contains one FLUT and two registers, it has eight inputs and four outputs. The number of FLUT inputs in Figure 2 is the maximum of *K* and *M* to ensure that both modes of the FLUT will always have a sufficient number of pins available. The mode of the FLUT determines how many of the inputs are usable. For example, with $M5$ and $K6$, there will be 6 inputs to each FLUT, but only 5 of them will be available in fracturable mode. The register inputs are given access to all of the LE inputs and the outputs of the FLUT. This is so that registers are trivial to pack into LEs with FLUTs.

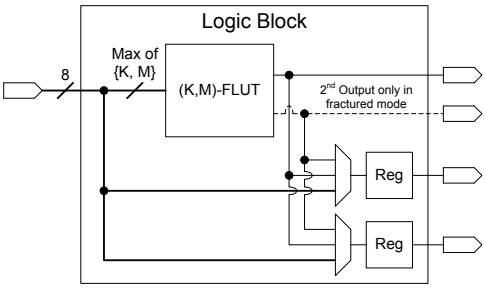


Fig. 2. Block diagram of the Logic Element used in the VPR architectures.

The architectures are made for use with a version of VPR [14] containing AAPack [3]. This version of VPR can target architectures containing fracturable LUTs and other complex structures. The routing architecture is specified as length-4, single driver, wire segments with $F_s = 3$, $F_c(in) = 0.15$ and $F_c(out) = 0.125$. The architectures consist of a square grid of LEs and routing resources, surrounded by I/O pins. The grid size and channel width of the routing are allowed to grow as needed to fit the design. The tool is currently area-driven, as timing-driven functionality has not yet been implemented; however, this is sufficient for our area-based investigation.

C. Minimizing the Number of Fracturable LUTs

The number of FLUTs in a design is given by

$$nFLUT = \left\lceil \frac{nLutTotal + nLutNoPair}{2} \right\rceil \qquad (2)$$

where $nLutTotal$ is the total number of LUTs in the mapping and $nLutNoPair$ is the number of LUTs that are packed into a FLUT operating in regular mode. Examples of LUTs in $nLutNoPair$ are size-*K* LUTs, which require the FLUT to operate in regular mode, and smaller-sized LUTs that cannot be packed into a fracturable mode FLUT due to input sharing restrictions. The value of *M* determines the input sharing requirements. To pack two size-5 LUTs together in the *M*5 architecture requires that both size-5 LUTs have all 5 inputs in common. Conversely, with *M*8, only 2 common inputs are required between the two size-5 LUTs. Note that it is possible to pack two LUTs with no common inputs into a FLUT, providing the size of both LUTs is sufficiently small.

IV. Experimental Results

In this section, we describe the methodology and experiments used to investigate the effects of using WireMap and LUT balancing in FPGA tech-mapping to minimize FLUT utilization.

A. Experimental Setup and Procedure

We used a benchmark suite consisting of the twenty largest MCNC circuits [15] and ten other benchmark circuits from sources such as the Opencores organization [16]. Each benchmark circuit is run through ABC [17] for synthesis and technology mapping. The circuits are first synthesized using the *resyn2* script, then tech-mapped for *K*6 using the priority cuts [11] mapper with structural choices enabled [18], and finally the mapping is checked for combinatorial equivalence with the initial circuit. The version of ABC we used includes the edge recovery techniques of WireMap [1].

We technology mapped both with and without edge recovery techniques enabled and refer to these two scenarios as the *WireMap* and *ClassicMap* mappers. We implemented LUT balancing by modifying the value returned by the *Weight()* function used in the Area Flow and Exact Area cost functions. The use of LUT balancing and/or edge-recovery are the primary tech-mapping variables we alter in our experiments. Thus, we perform tech-mapping of the benchmark suite circuits with the following settings:

- ClassicMap - no LUT balancing (i.e. the baseline)
- WireMap - no LUT balancing
- ClassicMap - with LUT balancing
- WireMap - with LUT balancing

For the two settings that have "with LUT balancing", we also sweep a variety of LUT size weights to identify the best LUT balancing parameters. So while the "no LUT balancing" options only require one mapping pass of the benchmark suite, the "with LUT balancing" options map the benchmark suite multiple times with different weights.

In our first set of experiments, the weight of a size-6 cut, *Weight(6)*, is varied from 1.0 to 2.5 in 0.1 increments. The weights of the smaller sized cuts are left at 1.0. When Weight(6) is greater than 1.0, the inclusion of size-6 cuts (and thus LUTs) is unfavourable for the area recovery cost functions. We ran a second set of experiments that modified both Weight(6) and Weight(5). In this second set of experiments, Weight(6) is always set to a larger value than Weight(5). The values of Weight(6) and Weight(5) chosen for the second set of experiments were selected after examining the results of our first set of experiments.

We define our *baseline* to be the mappings produced by the ClassicMap mapper without any LUT balancing. This means that the edge-recovery techniques of WireMap are disabled and the weight of all LUT sizes is 1.0. All FLUT percent reduction comparisons in this work are with respect to our baseline mapping results. Table I lists each benchmark's name, the number of Flip-Flops (FFs) it includes, and the depth and number of LUTs of the baseline mapping.

After mapping, circuits are packed, placed, and routed using AAPack [3] and VPR [14] for the four architectures described in Section III-B. Similarly, the mapped circuits are processed using Altera's Quartus II software tool flow, via QUIP [4], targeting the Stratix II [5] device EP2S60F1020C3. Quartus II will perform synthesis on the mappings it reads in. To preserve our tech-mapping solutions, the What You See Is What You Get (WYSIWYG) flag is set, which tells Quartus II to refrain from any significant synthesis operations (buffers and unused pins may still be removed). Other flags are set to tell Quartus II to pack for density, to perform a "Standard Fit", to pack registers for minimal area, and to turn off logic and register duplication during routing.

TABLE I
BENCHMARK SUITE CIRCUITS WITH BASELINE MAPPING STATISTICS.

Name	FFs	LUTs	Depth
s298	14	24	2
glue2	40	316	12
elliptic	194	318	6
ex5p	0	369	4
misex3	0	425	5
alu4	0	519	5
diffeq	305	560	7
apex4	0	571	5
bigkey	224	579	3
tseng	385	640	7
pajf	512	650	3
seq	0	657	5
ex1010	0	660	5
apex2	0	662	6
des	0	812	5
desa	64	865	6
iir1	204	870	18
dsip	224	873	3
rsd1	506	1102	10
pdc	0	1379	7
spla	0	1469	6
frisc	886	1745	13
s38584.1	1260	2387	6
s38417	1462	2499	6
rsd2	609	2531	15
oc54	386	2537	38
clma	33	2988	9
cfc18	2052	3410	8
cfc	2052	3411	8
cft8	2685	7081	10

B. Tech-Mapping Results

Figure 3(a) shows the distributions of LUT sizes produced by the ClassicMap mapper with varying Weight(6) values, other weights are held at 1.0. Each bar represents the total number of x-sized LUTs in the benchmark suite mappings, normalized against the baseline mapping (ClassicMap mapper, all LUT size weights = 1.0). Examining the chart shows that once Weight(6) is greater than 1.0, the number of size-6 LUTs drops off dramatically, validating our LUT balancing implementation. The frequency of smaller LUTs, in particular size-5 LUTs, increases to compensate for the missing size-6 LUTs. Also, the total number of LUTs shows a gradual increase with higher Weight(6) values.

Figure 3(b) shows the LUT size distributions when WireMap is used instead of the ClassicMap mapper. As before, the full range of Weight(6) values are used while other weights are held at 1.0, and each bar in the graph is normalized to the baseline mapping. As with ClassicMap, we observe that when Weight(6) is greater than 1.0: the number of size-6 LUTs is reduced, there are more small sized LUTs to compensate, and the total number of LUTs increases. The most notable difference is that the WireMap distribution favours the smallest sized LUTs, whereas ClassicMap compensated for the lack of size-6 LUTs primarily with more size-5 LUTs. In addition, in the absence of any LUT balancing (i.e. Weight(6) is 1.0) the WireMap mappings show the same trends as when LUT balancing is applied, indicating that WireMap and LUT balancing manipulate the LUT distribution with similar aims.

In our second set of experiments, we varied both Weight(6) and Weight(5). The LUT size distributions from this second set of experiments are shown in Figure 3(c) and Figure 3(d) for the ClassicMap and WireMap mappers respectively. As expected, the number of size-5 LUTs decreased with rising Weight(5) values for both tech-mappers.

During earlier runs of our experiments in our study, we noticed that for some benchmarks, LUT balancing induced a trade off between logic depth and FLUT minimization. For example, the depth of the benchmark *elliptic* increased from 6 to 7 for Weight(6) values of 1.7 and greater (for both WireMap and ClassicMap). This is undesirable as an increase in depth is equivalent to an increase in the longest combinatorial path of the circuit. We determined that this increase was due to the interaction of our LUT balancing scheme with the priority cuts mapper, which does not enumerate all possible cuts and thus cannot guarantee an optimal depth mapping [11]. We remedied this issue by disabling our LUT balancing scheme during the depth discovery portion of the mapping process.

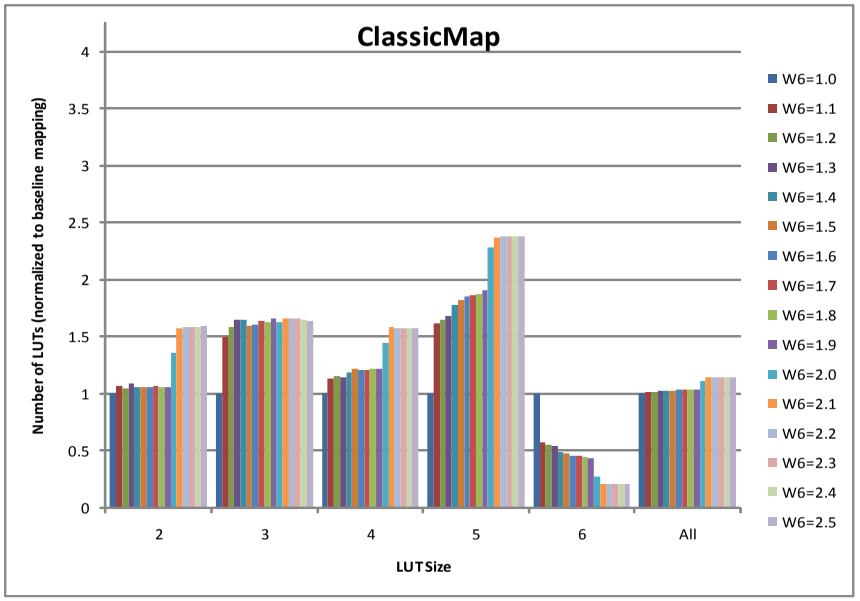
C. Packing Results

The mappings are run through VPR for the four academic FPGA architectures ($M5$, $M6$, $M7$, and $M8$). Figure 4(a) and Figure 4(b) graph the number of FLUTs utilized to pack the mappings for the first and second set of experiments. In each figure there are eight lines on the graph; each represents a mapper-architecture combination. The y-axis is the *number of FLUTs*; note that it starts at 500 to provide better resolution. The x-axis of Figure 4(a) corresponds to different Weight(6) values (other weights are held at 1.0), while the x-axis of Figure 4(b) has varying Weight(6) and Weight(5) values (again other weights are held at 1.0). Each data point on the trend lines is the geometric mean of the benchmark suite's FLUT count.

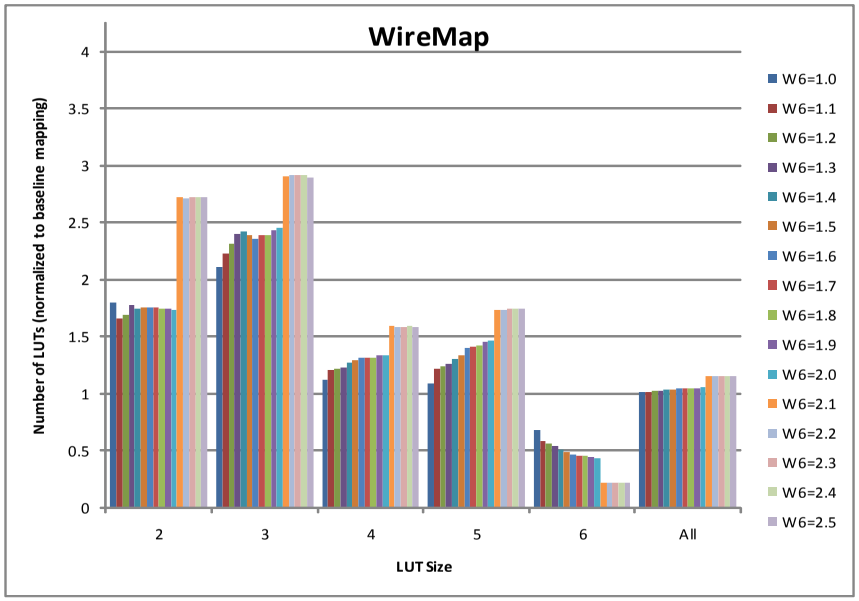
When LUT balancing is not used (i.e. all weights are 1.0), then WireMap outperforms ClassicMap (our baseline since no LUT balancing is used) for all architectures. However, when LUT balancing is enabled, this is no longer true. WireMap with LUT balancing mappings produced a greater reduction of FLUTs than the equivalent ClassicMap with LUT balancing mappings for the $M5$ and $M6$ architectures. This FLUT reduction is greater in our first set of experiments, when we are only varying Weight(6). In our second set of experiments, when we vary both Weight(6) and Weight(5), WireMap still outperforms ClassicMap, but the difference in FLUTs is less.

For the $M7$ architecture, WireMap and ClassicMap, both with LUT balancing enabled, produce similar results with WireMap having a slight edge for all but two weight settings. For the $M8$ architecture, ClassicMap with LUT balancing actually outperforms WireMap with LUT balancing by a small margin for all but one weight setting (Weight(6) = 2.4, Weight(5) = 2.0).

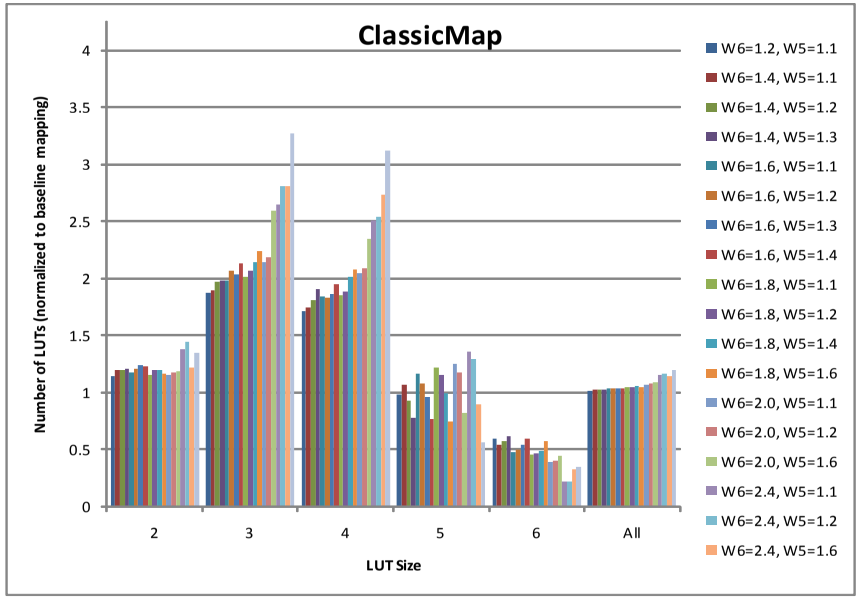
When M (i.e. the number of unique inputs a FLUT has in fractured mode) is small, WireMap does well. When M is



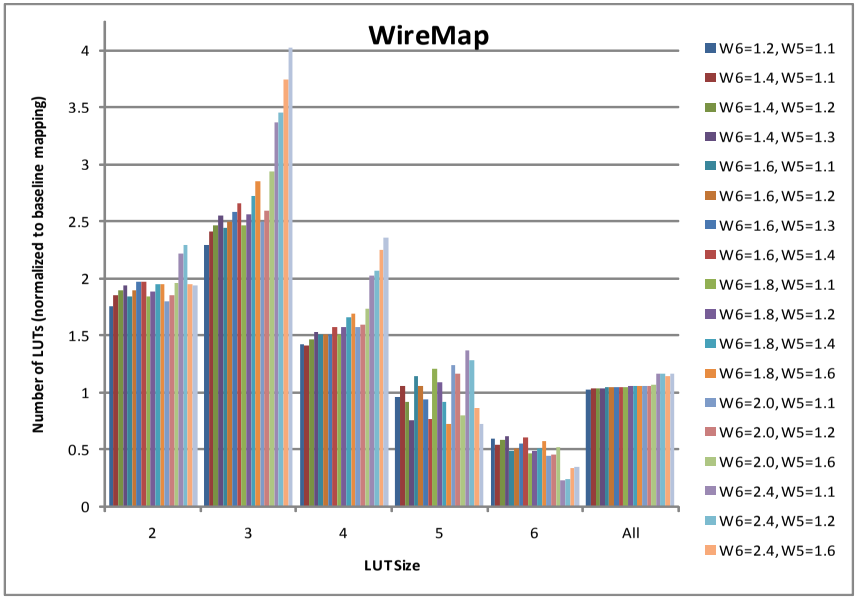
(a) ClassicMap LUT size distributions for varying Weight(6).



(b) WireMap LUT size distributions for varying Weight(6).

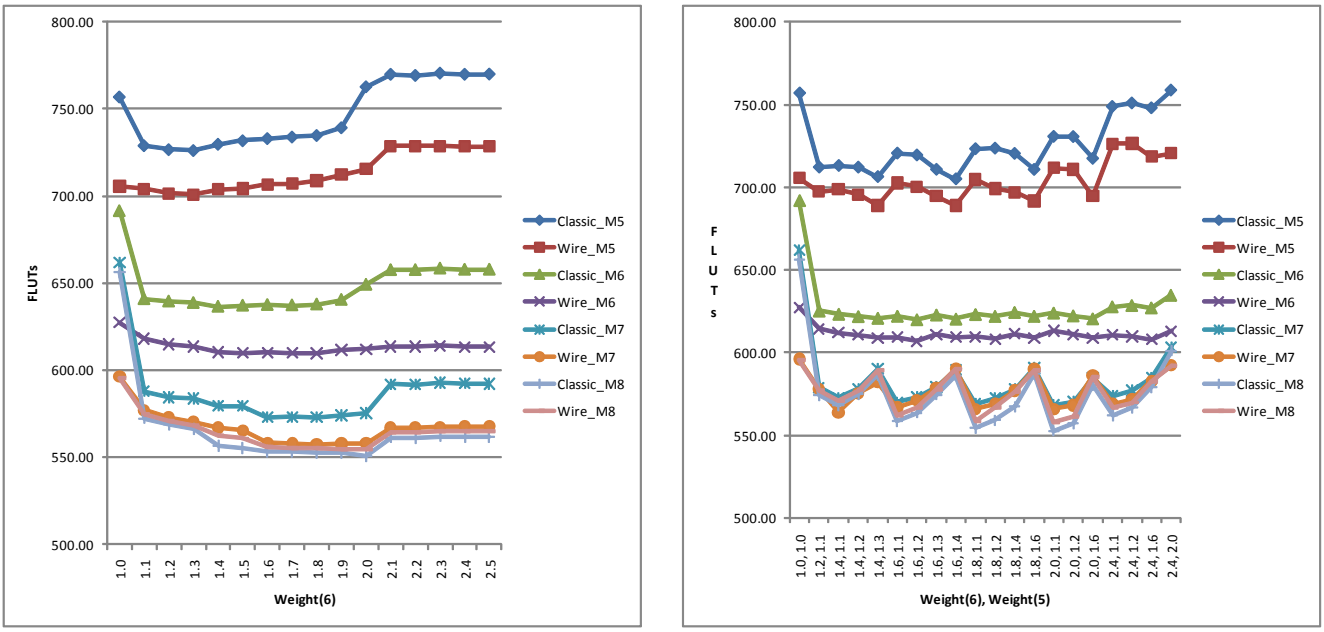


(c) ClassicMap LUT size distributions for varying Weight(6) and Weight(5).



(d) WireMap LUT size distributions for varying Weight(6) and Weight(5).

Fig. 3. LUT size distributions for different tech-mappers and LUT balancing parameters.



(a) Only Weight(6) is varied for LUT balancing.

(b) Both Weight(6) and Weight(5) are varied for LUT balancing.

Fig. 4. Graphs plotting the geometric mean of the number of FLUTs VPR packs the benchmark suite mappings into for various architectures and mapping techniques.

large then there is little difference between WireMap and ClassicMap. This can be explained by observing that WireMap's LUT size distribution tends to have more size-2 and size-3 LUTs than ClassicMap. The FLUT parameter M determines the input sharing constraints when two LUTs are being packed into a fractured mode FLUT. LUTs with fewer inputs are going to be easier to pack together into fractured mode FLUTs with tight input sharing constraints. For larger values of M, it seems that the input sharing constraints are sufficiently relaxed such that it is no longer difficult to fit the larger size-4 and size-5 LUTs into a fractured mode FLUT.

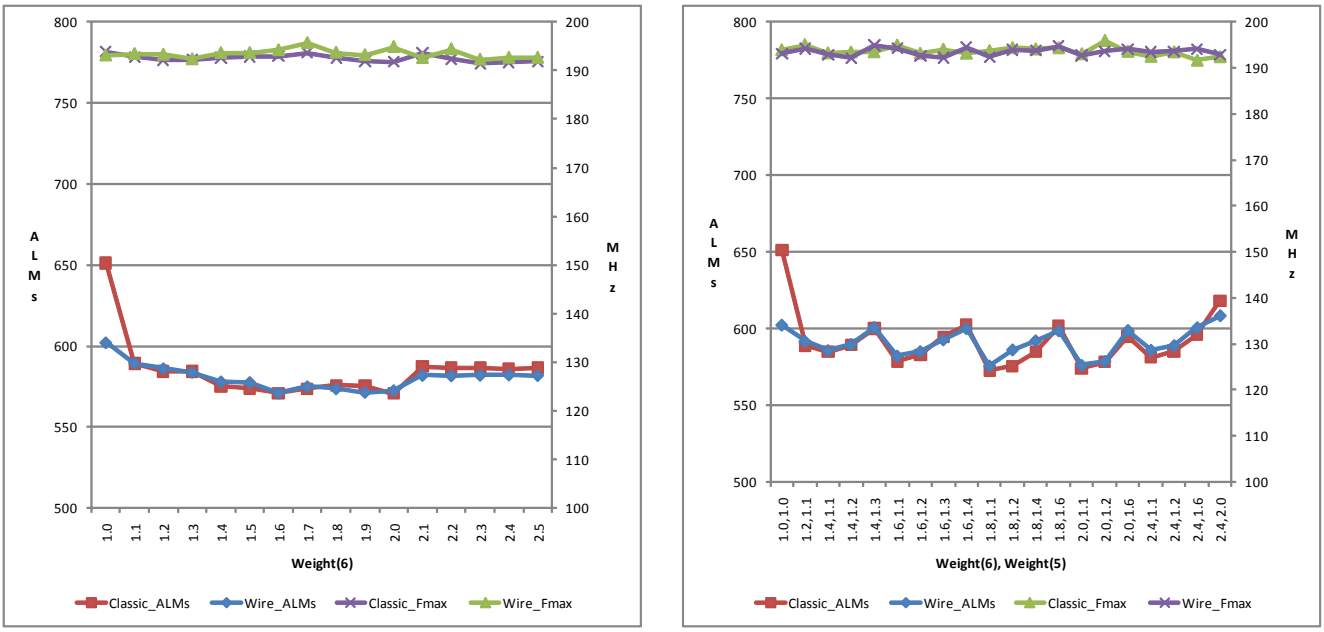
The Quartus II results are shown in Figure 5(a) and Figure 5(b), where the geometric means of the ALM count and maximum operating frequency (Fmax) of the benchmark suite circuits are graphed. An ALM can be viewed as roughly equivalent to the FLUT of the $M8$ VPR architecture. However, there are other significant differences between the two architectures. Thus, we do not make any direct comparisons between the FLUT counts for the $M8$ architecture and the ALM counts of the Stratix II. No Fmax is reported by Quartus II for designs that do not contain FFs (recall Table I). Therefore, the Fmax data points are the geometric mean of only those benchmarks that contain FFs.

Similar observations on FLUT reduction to those observed for the VPR $M8$ architecture can be made for the Quartus II data. WireMap and ClassicMap produce nearly identical results whenever LUT balancing is applied. Interestingly, the Fmax appears to remain essentially the same (all data points within 5 MHz of each other) irregardless of the packing density. We expect this would change if the Quartus II tool settings were changed to perform a more balanced flow instead of optimizing for area.

A summary of our best FLUT/ALM minimization results for each of the VPR and Stratix II architectures is given in Table II. Four entries are provided for each of the FPGA architectures. These entries correspond to the average benchmark suite FLUT usage when packing the mappings produced by both ClassicMap and WireMap with and without LUT balancing. The Weight(6) and Weight(5) values used during the mapping are provided for each entry in our table. These weights will be 1.0 if the "no LUT balancing" option was used. If the "with LUT balancing" option is used, then the weights listed are the best weights for FLUT minimization from all the LUT balancing runs we performed for the given architecture and tech-mapper. The "Percent Reduction" column gives the percent reduction in FLUTs versus the baseline of the architecture.

Examining the results of Table II yields the following observations.

- In the absence of LUT balancing, WireMap outperforms ClassicMap for all FPGA architectures.
- ClassicMap with LUT balancing outperforms WireMap without LUT balancing for all FPGA architectures.
- Adding LUT balancing decreases FLUTs usage for both ClassicMap and WireMap on all FPGA architectures.
- WireMap with LUT balancing outperforms ClassicMap with LUT balancing for the *M5*, and *M6* architectures.
- ClassicMap with LUT balancing and WireMap with LUT balancing produce similar results for the *M7*, *M8*, and Stratix II architectures.
- The "best" LUT balancing parameters for FLUT minimization vary with the FPGA architecture.



(a) Only Weight(6) is varied for LUT balancing.

(b) Both Weight(6) and Weight(5) are varied for LUT balancing.

Fig. 5. Quartus II ALM and Fmax results. Left y-axis is the geometric mean number of ALMs the benchmark suite packed into. Right y-axis is the geometric mean of the Fmax reported by those circuits in the benchmark suite that had flip-flops.

TABLE II
MAXIMUM PERCENT REDUCTIONS IN FLUT USAGE WITH RESPECT TO THE BASELINE FOR ALL ARCHITECTURES AND TECH-MAPPING TECHNIQUES.

Architecture	Tech-Mapper	Weight(6)	Weight(5)	FLUTs (ALMs)	Percent Reduction
M5	ClassicMap - no LUT balancing (baseline)	1.0	1.0	756.9	N/A
	ClassicMap - with LUT balancing	1.6	1.4	705.0	6.9%
	WireMap - no LUT balancing	1.0	1.0	705.7	6.8%
	WireMap - with LUT balancing	1.4	1.3	689.1	9.0%
M6	ClassicMap - no LUT balancing (baseline)	1.0	1.0	691.7	N/A
	ClassicMap - with LUT balancing	1.6	1.2	620.1	10.3%
	WireMap - no LUT balancing	1.0	1.0	627.4	9.3%
	WireMap - with LUT balancing	1.6	1.2	606.8	12.3%
M7	ClassicMap - no LUT balancing (baseline)	1.0	1.0	662.0	N/A
	ClassicMap - with LUT balancing	2.0	1.1	568.4	14.1%
	WireMap - no LUT balancing	1.0	1.0	596.3	9.9%
	WireMap - with LUT balancing	1.8	1.0	557.4	15.8%
M8	ClassicMap - no LUT balancing (baseline)	1.0	1.0	656.2	N/A
	ClassicMap - with LUT balancing	2.0	1.0	550.4	16.1%
	WireMap - no LUT balancing	1.0	1.0	595.3	9.3%
	WireMap - with LUT balancing	2.0	1.0	554.7	15.5%
Stratix II	ClassicMap - no LUT balancing (baseline)	1.0	1.0	651.1	N/A
	ClassicMap - with LUT balancing	2.0	1.0	570.6	12.4%
	WireMap - no LUT balancing	1.0	1.0	602.3	7.5%
	WireMap - with LUT balancing	1.6	1.0	571.0	12.3%

Based on our observations, we recommend using LUT balancing with appropriate values of *Weight()* for all technology mapping runs targeting FPGA architectures with FLUTs when FLUT minimization under depth constraints is desired. The downside of this approach is that finding good LUT balancing parameters (i.e. weights) requires some trial and error. Whether or not the use of WireMap, as opposed to ClassicMap, is appropriate depends on the architecture. FPGA architectures that have FLUTs with smaller values of M appear to benefit from WireMap's edge-recovery heuristics. However, once M is sufficiently large, we observed only small differences between WireMap and ClassicMap in terms of FLUT minimization.

V. CONCLUSION

In this paper, we combine the edge-recovery techniques of WireMap with our implementation of LUT balancing to perform technology mapping with the objective of minimizing the number of fracturable LUTs a mapping will utilize after packing. Packing is performed for four different academic FPGA architectures with FLUTs and for the Stratix II architecture. When packing the mappings into FLUTs with smaller M parameters, the combination of WireMap and LUT balancing produces mappings that pack into fewer FLUTs than if either

technique is used alone. For larger values of M, LUT balancing provides good FLUT usage results irregardless of whether or not edge-recovery techniques are used.

For future work, we plan to experiment with academic architectures that are more complex and include multipliers, memory elements, and clusters of Logic Elements. Adding these elements to our FPGA architectures will allow us to use a benchmark suite with larger, more complex circuits that are more representative of modern designs. We also look forward to AAPack becoming timing driven so that we can obtain more concrete data on the relationship between a reduced FLUT count and the critical path of a circuit. We are also investigating the creation of a model that can be used to predict which LUT balancing parameters are appropriate for a given FLUT architecture.

ACKNOWLEDGMENT

REFERENCES

[1] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "Wiremap: FPGA technology mapping for improved routability and enhanced LUT merging," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 2, pp. 1–24, 2009.

[2] M. Hutton, J. Schleicher, D. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, M. Bourgeault *et al.*, "Improving FPGA performance and area using an adaptive logic module," *Field Programmable Logic and Application*, pp. 135–144, 2004.

[3] J. Luu, J. Anderson, and J. Rose, "Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 227–236.

[4] S. Malhotra, T. Borer, D. Singh, and S. Brown, "The quartus university interface program: enabling advanced fpga research," in *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*. IEEE, 2004, pp. 225–230.

[5] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee *et al.*, "The Stratix II logic and routing architecture," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. ACM, 2005, pp. 14–20.

[6] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 1–12, 2002.

[7] V. Manohararajah, S. Brown, and Z. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 11, pp. 2331–2340, 2006.

[8] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 240–253, 2007.

[9] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: enabling a general and efficient FPGA mapping solution," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*. ACM, 1999, p. 35.

[10] D. Chen and J. Cong, "DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*. IEEE Computer Society, 2004, pp. 752–759.

[11] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*. IEEE, 2007, pp. 354–361.

[12] A. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for FPGA technology mapping," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, no. 11, pp. 1319–1332, 2002.

[13] T. Ahmed, P. Kundarewich, J. Anderson, B. Taylor, and R. Aggarwal, "Architecture-specific packing for virtex-5 FPGAs," in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. ACM, 2008, pp. 5–13.

[14] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. Fang, and J. Rose, "VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, pp. 133–142.

[15] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991.

[16] (2011) OpenCores website. [Online]. Available: http://opencores.org/

[17] (2011) ABC: A System for Sequential Synthesis and Verification website. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[18] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2894–2903, 2006.