

# Fracturable FPGA Logic Elements

Mike Hutton, David Lewis, Bruce Pedersen, Jay Schleicher, Richard Yuan,  
Gregg Baeckler, Andy Lee, Rahul Saini and Henry Kim

**Abstract**— The longstanding conventional wisdom in FPGA architecture is that a 4-input lookup-table (LUT) provides the most efficient tradeoff between area and delay. This paper introduces a new adaptive logic module based on fracturable 6-LUTs with the goal of fundamentally altering this relationship. This logic module can achieve the performance benefits of larger LUT sizes (up to 7) while actually decreasing area cost through sharing of input muxing resources and additional features that allow multiple functions to share the same LUT-mask. Benchmarking results show a 15% performance improvement and 12% area decrease using a 6,2 ALM comparing to a standard 4-LUT based logic module on the same process technology and routing architecture.

**Index Terms**—Architecture, FPGA, logic element, LUT, programmable logic.

## I. INTRODUCTION

THE design of logic cells for programmable logic devices drives both efficiency and performance. One of the primary parameters in logic element design for SRAM-programmable FPGAs is the size of the lookup table or LUT.

Rose *et al.* [15] studied the general area-efficiency of logic elements based on  $k$ -input lookup tables, and found the most efficient LUT-size to be  $k=4$ . Subsequent studies by Singh [17] and Kouloheris [11] showed that  $k=4$  is also best for area-delay product. Since these studies, however, there has been a 10X increase FPGA density and also a change in the proportion of critical path delay spent in the routing vs. the logic portion of the FPGA.

Figure 1 illustrates the basic tradeoff between area and delay found in previous research. Area per logic element grows with increasing LUT size while unit delay decreases. The goal of this paper is to introduce a logic element with variable LUT-size, thus capturing the delay benefits without area penalty and shifting the area curve out to the dashed line.

Previous studies have tried to take advantage of fixed heterogeneity to change the operation of this curve. He and Rose [9] proposed logic elements with a combination of 7-input and 3-input functions and a dynamic-programming based technology mapper. Kaviani and Brown [10] proposed combining the PTERM and LUT-based logic elements in the

same architecture. Both approaches involve CAD tools mapping to a fixed distribution of the two logic elements.

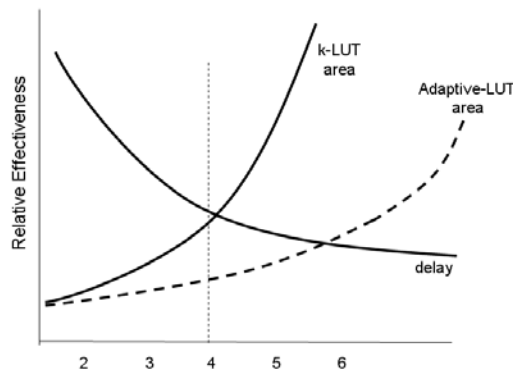


Figure 1. Delay-cost tradeoff with LUT granularity.

This paper proposes a new logic element structure which increases the size of the LUT, hence decreasing logic depth. However, to avoid the inherent area cost of increasing LUT size the logic element is adaptable or *fracturable* into smaller LUT sizes and able to pack smaller LUTs together efficiently with a combination of shared inputs and extra inputs for use in fracturing and other modes. This greatly reduces the input muxing cost to the logic module and improves utilization of the LUT mask, essentially shifting the cost curve out.

Section II discusses the area/delay tradeoff for pure 6-LUTs, then develops the new “adaptive logic module” or *ALM* and several interesting enhancements. Section III discusses the important issues of technology mapping necessary to achieve efficient packing. Experimental results are contained in Section IV, and conclusions in Section V.

## II. LOGIC ELEMENT ARCHITECTURE

Figure 2 shows a diagram of a generic FPGA logic element based on a 4-input LUT – the comparison LE for this study. This logic element contains a 4-LUT for general logic, arithmetic chains (not shown), and a DFF with selectable control signals. To the left is the connection block, in the terminology of Brown *et al.* [6], which is connected to a proportion  $F_c$  of the  $W$  global routing tracks adjacent to the logic element. In this diagram “globals” refers to clock and other secondary signals, and an ‘x’ represents a programmable connection in a logical routing multiplexor driving a LE input.

Henceforth this generic 4-LUT base logic module including its routing will be referred to as BLE4. Similarly a BLE5 and BLE6 will refer to the obvious extensions to 5 and 6 inputs

Manuscript submitted May 1, 2006.

All authors are with Altera Corp., San Jose, California except D. Lewis with Altera, Toronto, Ontario. Contact author mhutton@altera.com.

respectively.

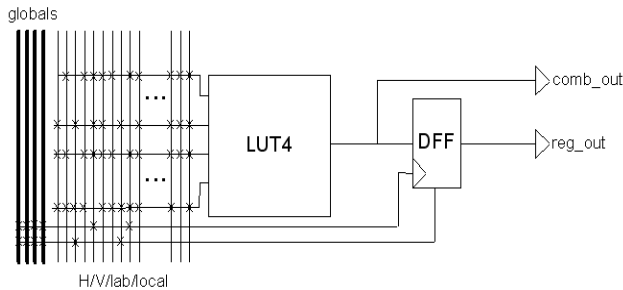


Figure 2. BLE4 – a 4-LUT based logic element.

Figure 3 shows the effect on a technology-mapped netlist of varying LUT-size  $k$ . Figure 3 (left) shows that the number of logic elements decreases, for example by about 22% when mapping to BLE6 vs. BLE4. Figure 3 (right) shows decrease in unit depth of the critical path – from BLE4 to BLE6 the critical path decreases by about 36% on average. This data is extracted using RASP [8] and FlowMap [7] on relatively small public benchmarks. The results are qualitatively identical to the larger circuits used later in the paper, but these illustrate less dramatic shifts on designs with carry-chains, more hard boundaries and greater pipelining. It might appear at the surface that both area and delay win, but when one takes into account the larger size of the LUT mask and input logic, as detailed in [15][16], the  $\#LUT * \text{sizeof}(LUT)$  metric for overall chip area loses beyond  $k=4$ .

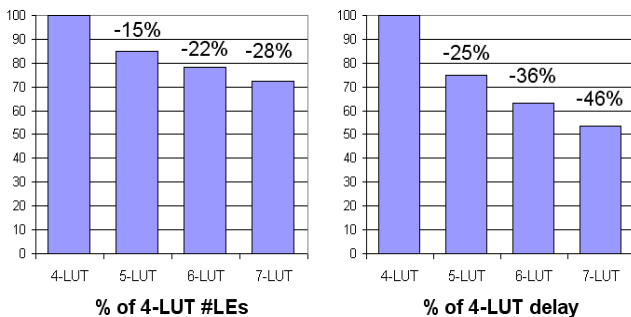


Figure 3. Decreasing #LUTs and depth with larger LUT size.

In addition to the longstanding argument that 6-LUTs would be better for depth, there are other compelling reasons why a 6-LUT would now be a better choice than it was in the past. As the density of programmable logic increases, two compounding factors arise. First, the proportion of overall area devoted to programmable routing is increasing – this is simply the super-linear increase in routing channel width to support larger FPGAs. Second, whereas the propagation delay through the LUT used to be very significant, routing delays increasingly dominate in deep-submicron process geometries because wires don't scale as well as transistors..

#### A. Area-growth with LUT-size

The area comprised by an FPGA architecture consists of the logic element itself plus the routing switches required to connect its inputs and outputs to the global routing network.

A 4-LUT requires 16 SRAM bits to hold the configuration memory  $LUT\text{-mask}$  plus a 16:1 mux (tree of 2:1 muxes). To increase from  $k=4$  to  $k=5$  basically doubles this area, since a 5-LUT is two 4-LUTs and a 2:1 mux. Because the adder, FF circuitry and output buffers is unchanged and comprises roughly half the area, the overall area of the logic-only portion of the LE footprint increases by roughly 50%.

The dominant routing cost is in the additional input mux shown to the left of Figure 2. For smaller-density architectures, the theory of [6] and associated papers would have the size of this mux to be more than half of  $W$ . Modern architectures, however, improve this efficiency with clustering. Clusters of logic blocks, beginning with the FLEX 8000 architecture [2], and studied by Betz and Rose [4] and more recently Ahmed and Rose [1] mitigate the growth in input mux size with hierarchy. This study shows that LUT sizes of 4-6 combined with cluster (LAB or CLB) sizes of 4-10 are best for area-delay product and also quantifies the loose relationship between unit delay and post-routing delay. Lemieux and Lewis [12] further show that using more sparse connections within clusters allowed for an area savings of 14% using clusters with 10 BLE6 logic elements, though no significant improvement in delay was found in these experiments.

Figure 2 does not show any switching between global routing wires, though these connections are assumed to be included in the overall area. They are not shown because for the comparisons done in this paper they can be considered a fixed cost independent of the change in the logic cell for reasons that should become apparent.

Figure 4 illustrates a logic array block or LAB cluster [19].

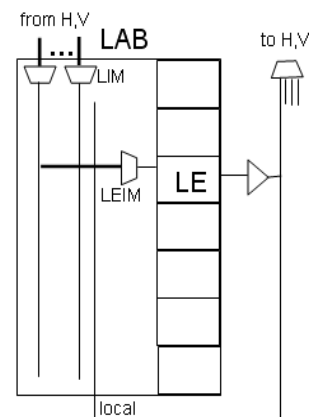


Figure 4. Cluster / LAB structure.

In a LAB the input mux is broken into two stages, the LIM or  $LAB\text{-input mux}$  which selects some number of signals from the global network and the LEIM or  $LE\text{ input mux}$  which selects from those. Each LIM selects from a subset of the  $W$  tracks passing over it, and each LEIM selects from a subset of the LIM lines and the local interconnect within the cluster. The effect of clustering is that the size of the LEIM can be reduced to about 20-30 even though  $W > 100$  so the introduction of a 5<sup>th</sup> input will cost roughly a 20 to 30:1 mux

and associated SRAM (at least 5 bits) to program it. In the less hierarchical Xilinx Virtex family [20] of architectures, the input muxing structure to a 4-LE CLB cluster is different, but the size of the LEIM mux is also approximately 30:1 [3].

In this paper, the metric of logic area is the amortized area per logic element, including the FF, input muxing and all other routing switches. This is computed as  $\#LAB \cdot \text{sizeof}(LAB)$  to additionally consider constraints on clustering that might arise from modifying the logic element.

Also interesting is the distribution of LUT sizes as the software technology maps to larger LUTs, shown in Figure 5. Observe that when tech-mapping for BLE4, the average distribution of LUT-sizes is 16% 2-LUTs, 27% 3-LUTs and 56% 4-LUTs – roughly half of all functions are not actually implementing a 4-LUT. For BLE6 mapping only 30% of all LEs are actually mapped as 6-LUTs, resulting in a “waste” of LUT-mask bits.

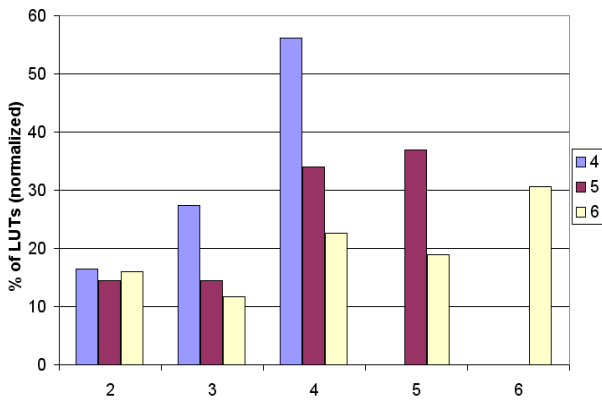


Figure 5. LUT-size distribution from tech-mapping.

The goal of this paper is to find a way to build a 6-LUT for its speed benefits without the requisite area cost.

### B. Composable Logic

If LUT-mask were the only concern, one could compose larger LUTs from smaller LUTs by adding a small amount of additional stitching hardware. Figure 6 shows how adding 3 2:1 muxes with additional inputs can be used to combine 4 neighbouring BLE4s with some additional hardware into one block of logic that can implement one 6-LUT, two 5-LUTs or four 4-LUTs. Note that one has to add the second set of 3 muxes to merge the 5-LUT and 6-LUT outputs with associated 4-LUTs or pay a large area cost for the output drivers.

Composing LUTs in this manner, however, is not really a viable method to make 6-LUTs. In the example shown, a 6-LUT requires 19 LEIM muxes (vs. 6 for a native BLE6), and the routing network is stressed by the need to route 4 copies of each of the 4-LUT inputs (a,b,c,d) and 2 copies of e. A further issue is that the second level muxes required to control the number of functional outputs give a speed hit on all paths through the logic element.

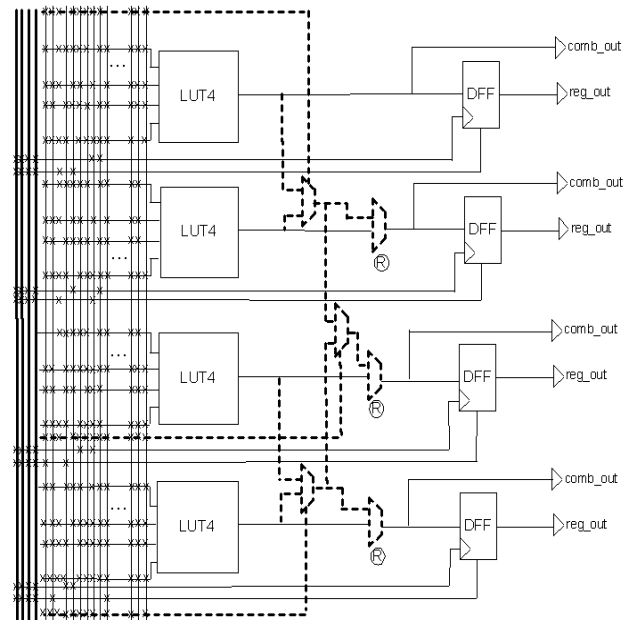


Figure 6. Composing 4-LUTs for larger functions.

The subtle assumption made by composability is that it is beneficial to provide all four BLE4 logic elements independently when not implementing a 6-LUT. However, this makes the 6-LUT cluster more than 4X the area of a native BLE4 (four FFs, four output muxes, 4X the number of LEIMs, etc). Applying the results of Figures 3 and 5, a netlist with 100 BLE4s will tech-map into 78 BLE6’s with a distribution of 23 LUT6, 32 LUT5, 17 LUT4, 9 LUT3 and 13 LUT2. Using 4 BLE4s for each LUT6 and 2 for each LUT5 implementing the netlist using composable BLE4 LEs will use 163 BLE4s vs. the original 100, an infeasible area bloat. Even assuming that an optimistic number of 6-LUT functions can be implemented in 32 bits of LUT-mask, the cost cannot get even close to parity. The reason is that every 6-LUT wastes 3 or more FFs, 3 output buffers and double route most LE input signals.

An improvement on this first attempt is to build a 6-LUT capable cluster based on two BLE5s as shown in Figure 7.

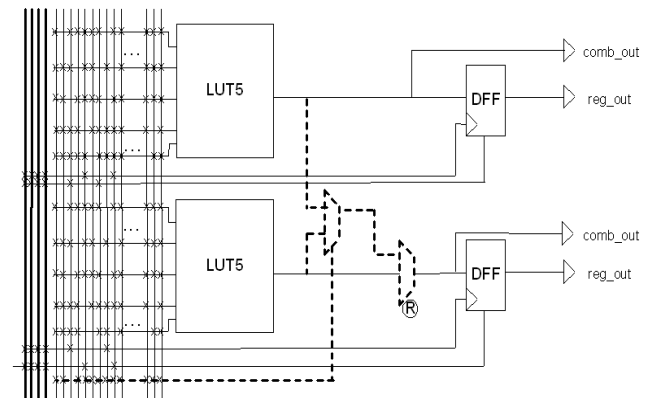


Figure 7. Composing 5-LUTs for larger functions.

It might appear initially that this illustration is just a re-packaging of Figure 6, but it is very important to note the

decreased cost from having only two FFs (vs. 4) and two output drivers (vs. 4). The input-a (and b,c,d) connections are now bridged internal to the 5-LUT functions, and do not need to be selected separately, saving about half the input muxes. This pair of BLE5s can implement either two 5-LUTs or one 6-LUT. To implement one 4-LUT an input is unused as in the base case for a 3-LUT. However, applying the distribution of Figure 5 results in 101 BLEs using two BLE5s per LUT6 and one for all other functions. A BLE5 is roughly 20% larger, accounting for the doubled LUT-mask size and extra LEIM, but the area devoted to LIM muxing, FFs, secondary signal generation and outputs is unchanged from the BLE4. The overall chip-cost ( $101 \cdot 1.2$ ) is still larger than the BLE4 architecture, albeit much better than BLE4 composition.

### C. Fracturable Logic

Figure 8 shows the alternative approach taken in this paper. Rather than assuming a constant base, the LUT size is permitted to vary between being a 5-LUT and a 4-LUT or smaller based on how the total of 8 inputs are used. When two 4-LUTs are packed together they are independent, similarly when a 5-LUT and 3-LUT are packed together they are independent, but when two 5-LUTs are adjacent they must share two inputs. This makes the base logic module significantly cheaper as long as the restrictions imposed by sharing are achieved.

The characteristics of the adaptable logic module are:

- 8 total inputs (LEIM muxes)
- 4 functional outputs (2 comb, 2 reg)
- 64 total bits of LUT-mask (6-input capable)
- 2 FFs, bits of arithmetic and FF control logic.

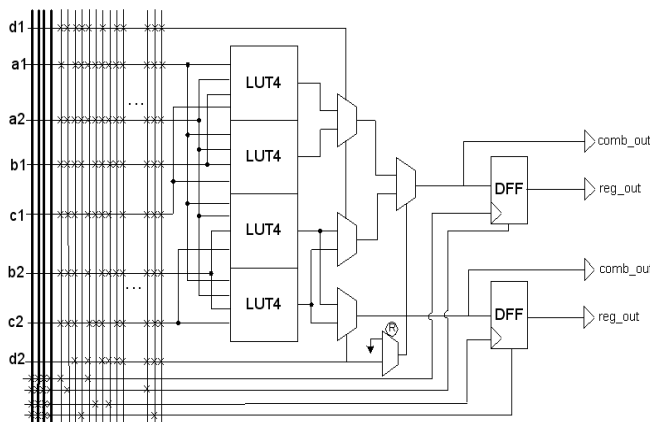


Figure 8. Fracturable 6,2 ALM with 8 inputs, 2 outputs

We denote this structure as a 6,2 ALM (adaptive logic module) because it is at its largest a 6-LUT, and has 2 spare inputs for use in fractured mode. The spare inputs are key to making the structure able to pack smaller functions efficiently without adding too much overall cost, and for the additional modes discussed later in the paper. It is also possible to build other  $k,m$  combinations such as a 4,2 fracturable ALM or a 6,1 fracturable ALM. Overall, the 6,2 was found to be the most effective.

The 6,2 fracturable ALM can implement

- one 6-LUT
- two 5-LUTs (with input sharing)
- two independent 4-LUTs.

among other combinations.

Some new terminology is now required: Each of the two functions potentially packable into the same ALM will be termed an *ALUT* or *adaptive LUT*. An ALUT is any  $k$ -input function,  $k \leq 6$ , that is output from technology mapping, and an ALM can support either one or two ALUTs, depending on their sizes and number of independent inputs. This term is necessary because for the BLE- $k$  case the number of ALUTs and LEs is identical, whereas the results for fracturable logic elements must distinguish between the number of ALUT functions implemented and the number of ALMs used after packing.

A key point to make here is that the logic module of Figure 8 is comparable in physical area with two BLE4 logic elements (half of Figure 4), but comparable in expressive power with two BLE5 logic elements (Figure 5).

### D. Speed and efficiency enhancements

There is one important problem that still exists with the ALM development thus far, and that is the number of outputs. In Figure 8 the problem is partially solved by pushing the cost of muxing the results of the larger LUT back in by putting a speed hit only on the 6<sup>th</sup> stage input d2. When in fractured mode setting the configuration SRAM-bit to 0 disconnects the upper ALM-half from the lower.

However, a better result is achievable with the following transformation to Figure 8. First the 2<sup>nd</sup> level muxes controlled by the 5<sup>th</sup> stage (d1 input) are duplicated, then new muxes to choose c1 or GND on the top ALUT and c2 or VCC on the bottom are added. The effect is to remove the additional output muxing from the critical path of the LE for all speed-paths, pushing it to the middle inputs only. Since the routing and FF/output interfaces are identical to Figure 8, they are not shown in Figure 9. A final modification is to introduce swap muxes controlled by SRAM configuration bits R and T, for reasons which will become clear shortly.

The operation of the logic module of Figure 9 is now a bit more complex. There are still two outputs and 8 inputs, and all previous properties. However there is an additional benefit from the latter transformation that makes it particularly powerful. Note that when a 6,2 ALM is used in 6-LUT mode, there are two outputs unused (wasted). With the additional circuitry, which denoted *shared LUT-mask* or SLM, the logic module can now be configured to implement two 6-input functions that share 4 inputs as long as they share the identical LUT-mask function. By setting  $R=1$  and  $T=1$ ,  $S=0$  and  $U=1$ , and reorganizing the LUT mask appropriately, SLM1 becomes a 6-LUT function of  $(a1,a2,b1,b2,c1,d1)$  and SLM2 becomes the same function, only of  $(a1,a2,b1,b2,c2,d2)$ .

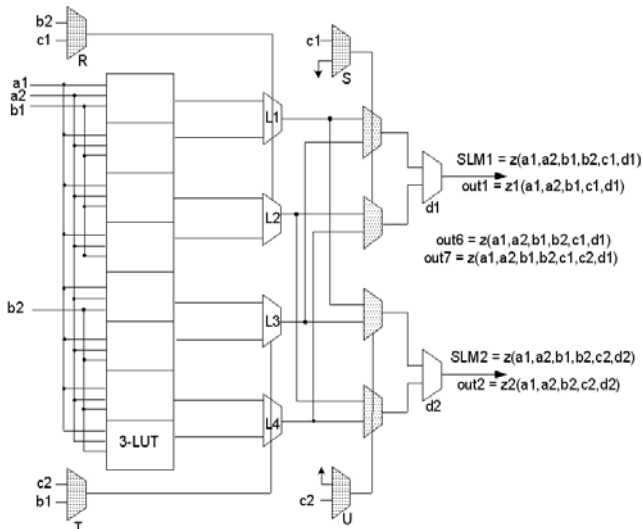


Figure 9. 6,2 ALM with 2 outputs and SLM

Though it might not seem like such functions would commonly occur, there are several particularly interesting situations where they do. One such instance is a 4:1 mux. Though a lone 4:1 mux can occur as a synthesized 6-input logic function, 4:1 muxes which occur in groups as part of a barrel shifter or crossbar almost always have 4 common data inputs, but different select lines. Thus, input designs which contain many crossbars or barrel shifters, and especially crossbars and barrel shifters operating on busses receive great advantage. For example, a benchmark SPI-4 (posphy level 4) core is able to implement about 12% of all ALMs as packed pairs of 6-LUTs implementing 4:1 muxes, meaning a 12% overall savings in ALM area (more in Section III.H).

When technology mapping to 6-LUTs, up to a third of all 6-LUTs generated can be packed in this way, dependent on technology mapper settings (area vs. speed mapping) and the particular design.

#### E. Logic element propagation delay

In the BLE4 logic element of Figure 3 the LE delay is skewed, meaning that the propagation delay for an input routed on the D input is significantly smaller than an input routed on the A or B inputs. Though it varies with different FPGA architectures, this can be as much as a 3X difference. The skewing is more than simply the later use of D relative to C, it is heavily influenced by the detailed circuit design.

In the CAD flow, synthesis generates a preferred implementation of the LUT function based on timing estimation, but the router is free to rotate LUT inputs (and modify the LUT-mask) to put the latest arriving signal on faster inputs. In general this can significantly reduce delay on the critical path with a relatively minor operation.

The ALM maintains this property. Even though the worst-case propagation delay through the LE is increased by growing to a 6-LUT, the fastest four delays are relatively unchanged (ignoring some parasitic effects). When implementing two independent 4 or 5-input functions from the ALM, the router is still able to rotate the most critical inputs

of the top function to c1 and d1 and of the bottom function to c2 and d2.

#### F. Measuring ALM Area

The concept of the ALM is significantly more powerful than simply building a 6-LUT. No matter how a 6-LUT is constructed it is going to look like 64 bits of LUT-mask with a tree of 2:1 muxes. The benefits of the proposed approach come from the minimal number of input muxes, further bridging of inputs required to achieve these various combinations and efficient use of LUT-mask via fracturing and with the additional 2 inputs. To implement the two 5-LUT functions  $z1$  and  $z2$  the common inputs  $a1$  and  $a2$  are already bridged and do not need be selected twice. The signals  $b1/b2$  and  $c1/c2$  are routed independently, and the upper function uses  $d1$  and the lower  $d2$ . When implementing a 6-LUT function, it is necessary to route  $b1,b2$  and  $c1,c2$  as the same signal.

The cost of this modified logic element is 8 total LEIMs (the same as two base BLE4), two FFs, two bits of arithmetic, and two sets of secondary signal generators.

In a naïve implementation, the final physical area is a little larger than two base BLE4s, roughly 15%, and this is essentially offset by the greater expressive power of the 5-LUTs (see Figure 3). The area cost arises from the additional LUT-mask SRAM bits and extra 2:1 muxes and configuration. Thus, the physical implementation of Figure 9 is roughly area-neutral with the BLE4 architecture yet achieves the 36% decrease in logic depth.

The following sections identify some additional area savings, both of which require CAD tools to leverage them. Then in Section III technology mapping improvements are described which can take the initially area-neutral approach and provide an area benefit by restructuring the LUT-size distribution of Figure 3.

#### G. Partial 7-input functions

An interesting side-effect of fracturability is that the ALM of Figure 9 can also implement a restricted set of 7-input functions, i.e. and incomplete 7-LUT.

Setting  $R=0$ , the upper two 4-LUTs are arbitrary functions of  $(a1,a2,b1,b2)$ . Setting  $T=1$  the bottom 4-LUTs are arbitrary functions of  $(a1,a2,b2,c2)$ . Setting  $S=1$  makes the upper gray muxes controlled by  $c1$  and the results of these controlled by  $d1$ . When  $c1=0$   $out7$  is driven by the L1 and L3 outputs chosen by  $d1$ . When  $c1=1$   $out7$  is driven by the L2 and L4 outputs chosen by  $d1$ . The result is that the ALM can compute a class of 7-input functions using all the inputs except for  $d2$ .

Specifically, the ALM can implement any 7-input function that can be expressed as

$$F1 = \text{fn}(a1, a2, b1, b2, d1)$$

$$F2 = \text{fn}(a1, a2, b2, c2, d1)$$

$$\text{Out} = \text{mux}(F1, F2; c1)$$

The specific template of incomplete 7-LUT functions is a  $c1$ -controlled mux of two 5-input functions which share 4 of

their inputs, differing only in b1 and c2. The reason that the output of the functional template is controlled by c1 rather than d1 as shown in the physical diagram comes from the LUT-mask changes performed by synthesis to rotate the c1 and d1 effects.

#### H. Muxes, Barrel Shifters and Crossbars

A natural property of the 6-LUT is that it can implement a 4:1 mux in one LUT. Figure 10 shows how to build an 8:1 mux in 2 ALMs (4 ALUTs). In one ALM sub-functions  $y_0$  and  $y_1$  are computed and packed as two 5-LUTs with shared inputs. In the second ALM the output of the 8:1 mux can be computed using 7-LUT mode by  $F1=fn(s_0,s_1,d_3,y_0,y_1)$  and  $F2=fn(s_0,s_1,d_7,y_0,y_1)$  and the 2:1 mux controlled by  $s_2$ . An 8:1 mux implemented with simple BLE4 requires 5 BLE4 logic elements vs. 2 ALMs, which saves roughly the area of one BLE4.

It is worth noting that though the composable BLE4 structure of Figure 4 is not efficient for making 6-LUTs or 5-LUTs, it is quite useful for building muxes – it can build an 8:1 mux in 4 composable BLE4s, which is comparable in area to the 2 ALM solution.

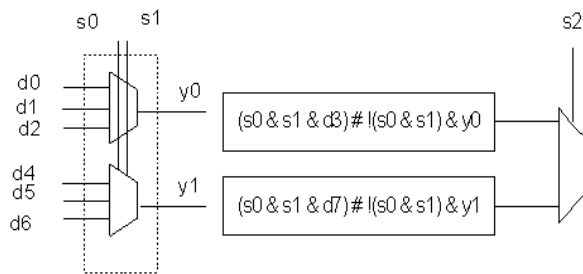


Figure 10. 7-input mode for implementing 8:1 mux.

Much more interesting than the simple mux are systems built of multiplexors, specifically barrel shifters and crossbars.

A simple 16-bit barrel shifter defined by

$$\text{out}[15:0] = \text{in}[15:0] \lll k[3:0]$$

results in 16 16:1 muxes or  $16 \times 5$  4:1 muxes =  $16 \times 5 \times 2$  LUTs = 160 LUTs synthesized in the naïve way or  $16 \times 4$  2:1 muxes = 64 LUTs synthesized properly into a  $n \cdot \log(n)$  shifter network of 2:1 muxes. However, SLM-mode (two 6-LUTs with the same LUT-mask and 4 common data) of the ALM allows this to be greatly compressed. In the ALM this can be implemented in 32 4:1 muxes (ALUTs) in a 4:1 shifter network, which together pack into 23 ALMs a nearly 2X area savings over BLE4.

Similar results can be had for crossbars, because the same property holds – systems of 4:1 muxes with common data and different select lines. A  $16 \times 16$  crossbar requires 160 BLE4s, which can be reduced to 128 BLE4+ logic cells with the added hardware of Figure 6, then further reduced to 88 ALUTs by the SLM property of the ALM.

#### I. 6-LUTs and DES

As a final comment on the choice of 6-LUTs as the basis for the ALM note that in addition to 4:1 muxes, 6-input

functions are natural implementations for many other logical functions. One such class of functions is DES encryption.

The core operation of DES is an array of 8 sboxes or substitution tables. Each sbox has 6 inputs and produces 4 outputs. In a parallel implementation when targeting speed, it is usual to produce 128 SBOXes, each of which needs 6 BLE4 for each of 4 outputs (3072 LEs).

The sbox has a natural implementation in 4 6-LUTs. Due to the complex nature of the function each of the 6-LUTs would otherwise require the worst-case 6 4-LUTs shown in Figure 11. This behaviour is typical of other encryption functions such as Rijndael and is a further justification that 6-LUTs are a “natural” building block for combinational functions.

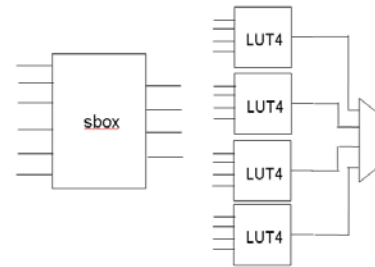


Figure 11. DES sbox with BLE4.

For an area-optimized DES core, the ALM described in this paper uses 239 ALMs vs. 736 in BLE4. For the speed-optimized version the result is 1465 ALMs vs. 5352 BLE4 logic elements. This represents a roughly 35% and 45% overall area improvement, respectively.

### III. BALANCED TECHNOLOGY MAPPING

Since the final implementation area depends on the total number of ALMs used in the device, not on the number of ALUTs generated, it is very important to generate a distribution of LUTs that is amenable to efficient packing.

Figure 12 summarizes the configurations that can be supported by the ALM of Figure 9, as outlined in the previous sections. Any two arbitrary 4-input ALUTs generated by tech-mapping can be packed into an ALM. This means that an ALM is backwards compatible with a netlist generated from a BLE4 technology mapping (though less efficient than re-mapping). Similarly a 5-input and 3-input ALUT can always be packed because they require no shared inputs. In the case of two 5-input ALUTs, or a 5-input and a 4-input, the pairing depends on the ability to share inputs, and in the case of more than one 6-input ALUT it depends both on the ability to share inputs according to a common input rotation that generates an identical LUT-mask.

To motivate the need for balanced technology mapping, consider Figure 13. The network on the left can be covered by three ALUTs – two 6-LUTs and one 4-LUT (upper solution), and this is the solution that technology mapping will generate. After packing into ALMs, the solution has depth 2 and  $2\frac{1}{2}$  ALMs. The solution to the bottom, though it generates 4 ALUTs (all 5-LUTs), can be efficiently packed

into just 2 ALMs while maintaining the depth-2 solution.

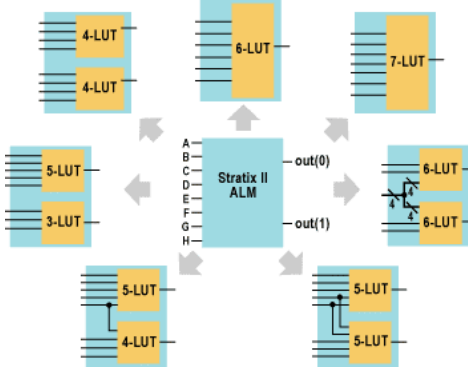


Figure 12. ALM packing configurations.

The synthesis flow is the following: a network of synthesis gates is decomposed into 2-input gates then covered by k-input functions. The resulting k-input functions are clustered into LABs so that delay of the critical path (now using placement timing estimates) is minimized, all input and secondary signals to a LAB meet hard hardware constraints and attempt to meet soft LAB input constraints. This latter problem is called LAB-clustering.

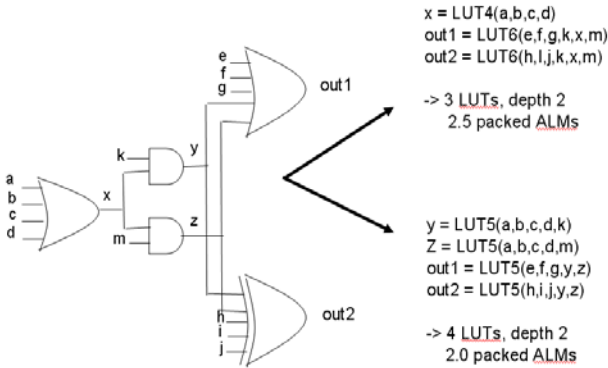


Figure 13. Better packing with balanced mapper.

A balanced technology mapping algorithm is aware of both the combinational delay of the network and also the LUT distribution, and is able to generate a solution that is more packable – in other words choosing the second solution in the example of Figure 11, because even though it has an additional ALUT it generates one fewer ALUT or “½ ALM” after packing while still maintaining minimal delay.

The easiest way to look at balanced technology mapping is simply that the tool should not generate a given number of 6-LUTs when it can instead generate proportion less than 2X that many smaller ALUTs and maintain optimal depth – i.e. a 6-LUT should be charged as 2 ALUTs (except in SLM mode) in the mapper.

Figure 14 shows the effectiveness of the pre-existing k-LUT tech-mapper, tech-mapping weighted to avoid 6-LUTs, and tech-mapping weighted to aggressively avoid 6-LUTs. The data is constructed using 80 industry designs and commercial tools, with a prototype technology mapper.

It is necessary to point out that much of the benefits of

packing are only seen because of the flexibility of the 2 extra inputs in the 6,2 ALM. A 6,0 ALM would still get the speed benefit for 6-LUTs, but the poor packing results for LUTs with 5 and fewer inputs would make for a significant area cost. A 5,2 ALM would have some of the benefits of the 6,2 but without the 4:1 mux and other benefits of a 6-LUT.

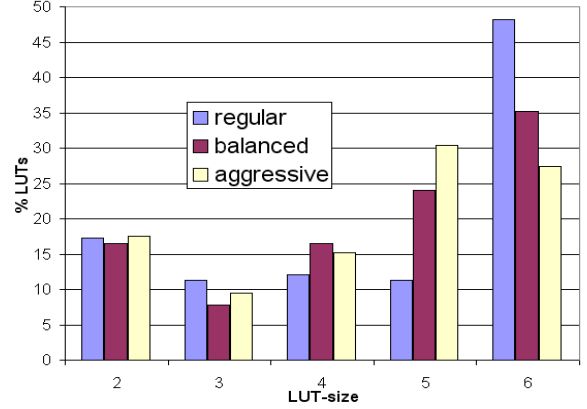


Figure 14. LUT-size distribution with different balancing.

#### A. 7-LUT mapping

As mentioned earlier, the ALM shown in Figure 9 is capable of implementing a subset of 7-input functions. Thus it might be curious that there is no 7-LUT bar in Figure 15. Empirical trials showed that the best solution is not to map to 7-input functions, but rather to find these functions in two ways. First, to specifically recognize 8:1 muxes and package them as per Figure 11 during RTL-level synthesis. Secondly, to post-process the technology mapped netlist to look for 7-cuts (in the FlowMap sense [7]) in that netlist, and attempt to remap these into 7-LUT functions admissible in the ALM.

Empirically the designs in the industrial benchmark suite have a 7% incidence of 7-LUT functions in an ALM.

## IV. EXPERIMENTAL RESULTS

This section evaluates the efficiency and performance of the 6,2 ALM and software flow to technology map to it.

#### A. Experimental Setup

The experiment design set consists of 80 large industrial designs in VHDL or Verilog, all over 20,000 BLE4 in size. Each design was synthesized mapped, clustered into a netlist of LABs, memories and DSP blocks and placed and routed to a proxy BLE4 architecture based on the Stratix FPGA [2]. Place and route uses the Altera PLD Modeling Toolkit (PMT) which is based on VPR place&route [5].

For evaluating the ALM, the routing architecture of the LIM and global network outside the LAB is held fixed. However optimization sweeps for connectivity of the LEIM is performed for both the BLE4 and ALM 6,2 cases, so that each receives its optimal layout and connectivity while maintaining routability. Timing delays for the different delays through the BLE4 and ALM are obtained by Spice simulation based on a

preliminary layout on a common 130nm process. Routing delays are also obtained by Spice, and are common between the two architectures. Area models are computed using preliminary layout estimation, also using common 130nm process design rules. Layout optimizations and rough transistor sizing is done to optimize the BLE4 and ALM independently.

Comparison data points for both delay and area will be presented as a ratio of new to old in a gain chart – one bar for each design – to illustrate the per-design changes. Geometric mean is used for descriptive statistics, because this is statistically more valid than arithmetic mean and always reports a conservative benefit.

### B. Performance

The introductory discussion showed that mapping to 6-LUTs will generate an abstract netlist in which depth is reduced on average by 36%. For benchmarking delay results actual delay after place&route is used. The comparison statistic is the performance of the slowest clock, which is roughly the inverse of the critical path delay after accounting for any clock skew.

Figure 15 shows the bottom-line performance achieved by the 6,2 ALM, shown as a gain chart. Note again that this chart isolates the effects due only to the change of the BLE4 to ALM 6,2 on the same process and with no other architectural improvements. Overall, the ALM is able to capture a 15% mean improvement in circuit performance.

### C. Area

To account for all software and architectural issues, particularly the requirement for clustering and packing of ALMs, the metric used for area is #LABs\* sizeof(LAB) for each of the ALM-based and BLE4-based architecture. Each design is placed into the smallest possible floorplan.

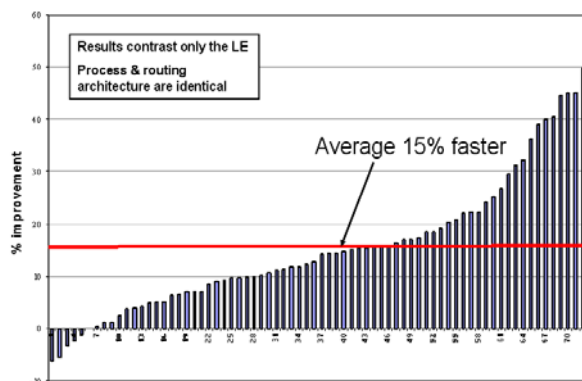


Figure 15. ALM performance vs. BLE4.

Figure 16 shows the results for area. On average, the results show 12% better area for the architecture based on 6,2 ALMs over the base architecture based on BLE4. Both architectures successfully route all designs in the design set.

## V. CONCLUSIONS

This paper presents a new and novel adaptive logic module

structure for FPGAs based on k,m fracturability. The goal of the ALM is to allow technology mapping to larger functions in order to capture the significant performance benefits of wider functions without the unacceptable cost that would be incurred with a native BLE6 logic element. Further enhancements to this structure showed shared-LUT-mask operation and incomplete-LUT modes allowing further area benefits.

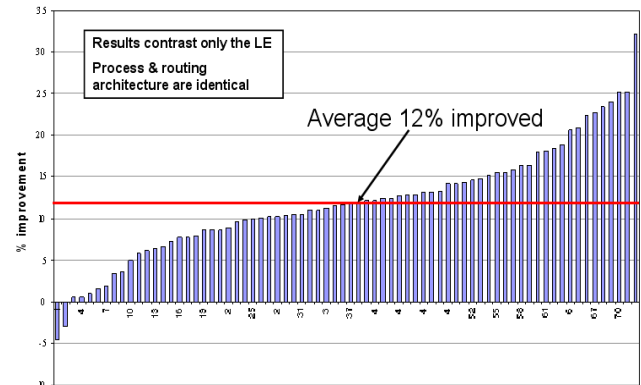


Figure 16. ALM area vs. BLE4.

For a specific 6,2 ALM based on a 6-LUT that is fracturable into 5 LUTs and has support for shared LUT-mask we achieve a 15% performance improvement in combination with a 12% area improvement over a traditional BLE4-based FPGA architecture.

## REFERENCES

- [1] E. Ahmed and J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," in Proc. ACM Symp. FPGAs, pp. 3-12, 2000.
- [2] Altera Corp. <http://www.altera.com>.
- [3] J. Anderson, F. Najm and T. Tuan, "Active Leakage Power Optimization for FPGAs", in Proc. ACM Symp. FPGAs, pp. 33-41, 2004.
- [4] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size", in Proc. Custom Integrated Circuits Conference 1997, pp. 551-554.
- [5] V. Betz, J. Rose and A. Marquardt. "Architecture and CAD for Deep-Submicron FPGAs", Kluwer, 1999.
- [6] S. Brown, R. Francis, J. Rose and Z. Vranesic, "Field-Programmable Gate Arrays", Kluwer, 1992.
- [7] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", IEEE Trans. CAD Vol 13 No 1, pp. 1-12, 1994.
- [8] J. Cong, J. Peck and Y. Ding, "RASP: A General Logic Synthesis System for SRAM-based FPGAs", in Proc. ACM Symp. FPGAs, pp. 137-143, 1996.
- [9] J. He and J. Rose, "Advantages of Heterogeneous Logic Block Architecture for FPGAs", in Proc. IEEE Custom Integrated Circuits Conf. (CICC), pp. 7.4.1-7.4.5, 1993.
- [10] A. Kaviani and S. Brown, "Hybrid FPGA Architecture", in Proc. ACM Symp. FPGAs, pp. 3-9, 1996.
- [11] J. Kouloheris and A.El Gamal, "FPGA Performance vs. Cell Granularity", Proc. of Custom Integrated Circuits Conference, May 1991, pp. 6.2.1 - 6.2.4.
- [12] G. Lemieux and D. Lewis, "Using Sparse Crossbars Within LUT Clusters", in Proc. ACM Symp. FPGAs, pp. 59-68 2001.
- [13] D. Lewis, et al., "The Stratix Logic and Routing Architecture", in Proc. ACM Symp. FPGAs, pp. 12-20, 2002.

- [14] D. Lewis, *et. al.*, "The Stratix II Logic and Routing Architecture", in Proc. ACM Synp. FPGAs, pp. 14-20, 2005.
- [15] J. Rose, R.J. Francis, D. Lewis and P. Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Functionality on Area Efficiency", IEEE Journal of Solid-State Circuits, pp. 1217-1225, 1990.
- [16] J. Rose, R.J. Francis, P. Chow and D. Lewis, "The Effect of Logic Block Complexity on Area of Programmable Gate Arrays", Proc. IEEE Custom Integrated Circuits Conference (CICC), pp. 5.3.1-5.3.5 1989.
- [17] S. Singh, J. Rose, P. Chow and D. Lewis, "The Effect of Logic Block Architecture on FPGA Performance", IEEE Journal of Solid-State Circuits, Vol 27 No. 3, pp. 282-287, 1992.
- [18] S. Trimberger, K.Duong, and B. Conn, "Architecture Issues and Solutions for a High-Capacity FPGA", Proc. ACM Synp. FPGAs, pp. 3-9, 1997.
- [19] K. Veenstra, B. Pedersen, J. Schleicher and C. Sung, "Optimizations for a Highly Cost-Efficient Programmable Logic Architecture", in Proc. ACM Synp. FPGAs, pp. 20-24, 1998.
- [20] Xilinx Corp. <http://www.xilinx.com>.



101 Innovation Drive  
San Jose, CA 95134  
(408) 544-7000  
<http://www.altera.com>

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.