Technology Mapping for Heterogeneous FPGAs

Jianshe He and Jonathan Rose
Department of Electrical and Computer Engineering,
University of Toronto, Toronto, M5S 1A4, Canada

October 16, 1997

Abstract

Truly heterogenous FPGAs, those with two different kinds of logic block, don't exist in the commercial world in part because logic synthesis for them is difficult. The difficulty arises because FPGAs are pre-fabricated, and so the ratio of the number of each type of block is fixed, which requires a constraint on the mapping that is not present for either homogenous FPGAs or ASICs.

This paper presents a general approach for technology mapping into heterogenous lookup-table based FPGAs. It is applied both at the boolean network node level and at a more global level across a collection of mapped sub-circuits. This latter portion of the algorithm is optimal, and is applicable to any heterogenous FPGA, not simply those based on lookup tables.

In comparison to a reasonably obvious alternative approach (adjusting the output of an homogenous mapping algorithm) the new algorithm achieves, depending on the heterogenous architecture, an average improvement of up to 48% over a large set of benchmark circuits.

1 Introduction

As Field-Programmable Gate Arrays (FPGAs) become more accepted and integral to the digital design process, there will be a strong drive to produce faster and higher-density devices. One architectural dimension that needs to be explored for its speed and density benefits is that of *heterogenous* FPGAs which employ more than one basic kind of logic block. Several commercial FPGAs employ limited forms of heterogenous structures: the Actel Act-2 [Ahre90] has separate combinational and sequential logic blocks. The Xilinx 4000 logic block [Hsie90] consists of two 4-input lookup tables (4-LUTs) hard-wired to a single 3-LUT. While the latter device has two different sizes of LUT, it is not a purely heterogenous device because of the hard-wired connections between the blocks. As such,

synthesis algorithms for the Xilinx 4000 do not apply to the kind of architectures discussed in this paper.

This paper presents a technology mapping algorithm for heterogeneous FPGAs that contain two sizes of LUT which are completely independent in the routing structure. This tool is useful both for exploration of different variations of heterogenous architectures and in the design process for the use of such chips.

We are motivated to investigate heterogenous structures because a selection of logic blocks may permit more area-efficient implementations of two different kinds of logic block. For example, consider the boolean networks illustrated in Figure 1 and Figure 2 and their respective implementations using 4-LUTs and 3-LUTs. The two most important area measures of an FPGA architecture are the total number of pins on the logic blocks in the circuit, and the total number of lookup table bits in all of the logic blocks. The network in Figure 1 is more area-efficient when implemented using 4-LUTs, as it requires fewer bits and pins. The network in Figure 2, however, is more efficient using 3-LUTs in terms of the number of pins and bits required. We expect that most circuits contain mixtures of different types of networks such as those pictured in these examples, and so could benefit from an architecture that provides two types of logic blocks.

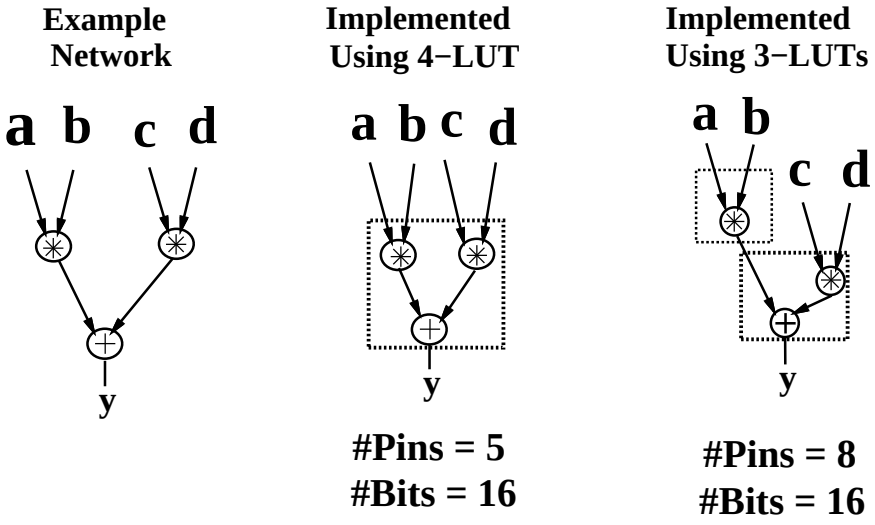

Figure 1: Example in which 4-LUT is Superior

A barrier to the use of heterogeneous FPGAs is the difficulty of synthesizing into such devices, as discussed in the next section. This paper presents a technology mapping algorithm for FPGAs which have a mixture of p-input LUTs and s-input LUTs in the ratio of r, where $s < p$, and r $= \frac{N_s}{N_p}$, where N_p and N_s are the number of p-LUTs and the number of s-LUTs respectively in the pre-fabricated heterogenous FPGA.

While there has been a great deal of effort applied to the homogeneous FPGA technology mapping problem for LUTs [Murg90] [Murg91a] [Fran90] [Fran91a] [Abou90] [Filo91] [Karp91] [Woo91] [Chen92] [Cong92] [Cong93] [Sawk92], to our knowledge there is no prior research on the heterogeneous problem. As mentioned above, the Xilinx 4000 logic block is not a purely heterogeneous structure.

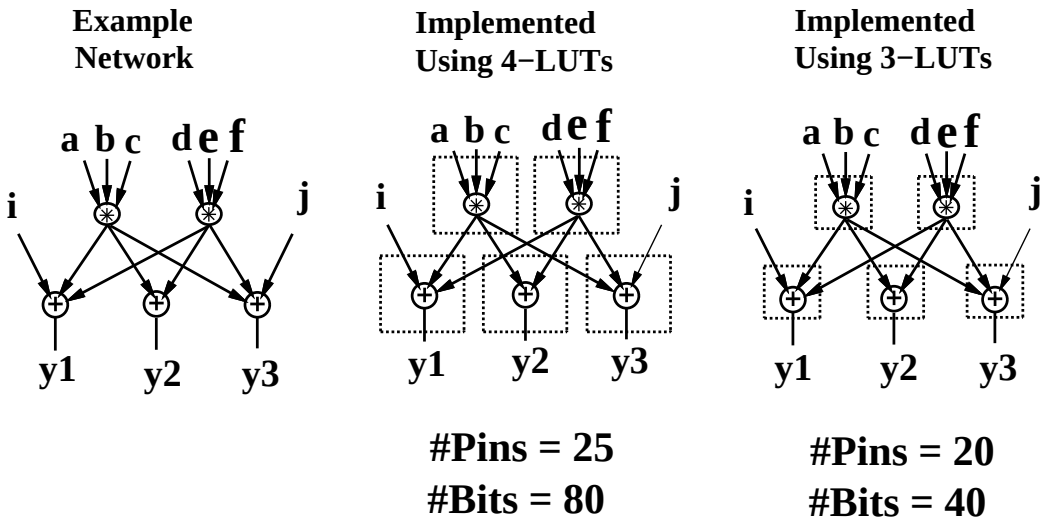

Figure 2: Example in which 3-LUT is Superior

This paper is organized as follows: the next section describes the basic notation and defines the technology mapping problem. Section 3 describes the heterogeneous technology mapping algorithm, while Section 4 provides experimental comparisons.

2 Notation and Problem Definition

We will consider heterogeneous FPGAs with just two sizes of lookup table. The larger lookup table will be referred to as the $p-$LUT, and the smaller as an s-LUT ($p > s$). An important architectural parameter of a heterogenous FPGA is the ratio of the number of the two types of block that are present in the FPGA, $r = \frac{N_s}{N_p}$. Note that this ratio is fixed for a given FPGA, because FPGAs are pre-fabricated. We will assume that r is either an integer, when $r \geq 1$, or the reciprocal of an integer, when $r < 1$. Thus if $r \geq 1$ then there are r s-LUTs for each p-LUT, and if $r < 1$ there are $\frac{1}{r}$ p-LUTs for every s-LUT.

The heterogenous technology mapping problem can be stated as follows: given a boolean network [Bray90], G, produce a mapped circuit M, which is a network of p-LUTs and s-LUTs of equivalent functionality to G. We are concerned with minimizing the size of the FPGA needed to implement the boolean network. Since r is fixed in a family of heterogenous FPGAs, the basic unit of *size* of a heterogenous FPGA is $\mathbf{r}$ s-LUTs and one $p-$LUT for $r \geq 1$ (we will usually assume $r \geq 1$ for the sake of brevity, but similar definitions apply when $r < 1$). Hence we wish to minimize the number of these units, which we will call a *supertile*.

Figure 3(a) gives an example of supertile with $p = 3, s = 2$, and $r = 2$. Figure 3(b) illustrates an array of such supertiles. If we define the number of $p-$LUTs in the mapped network M to be N_p, and the number of s-LUTs to be N_s, then the number of supertiles, N_{sup}, is given by:

$$N_{sup} = \mathbf{max}(N_p, \lceil \tfrac{N_s}{r} \rceil) \qquad (r \geq 1)$$

The maximum function makes this a non-linear cost function and hence is difficult to mini-

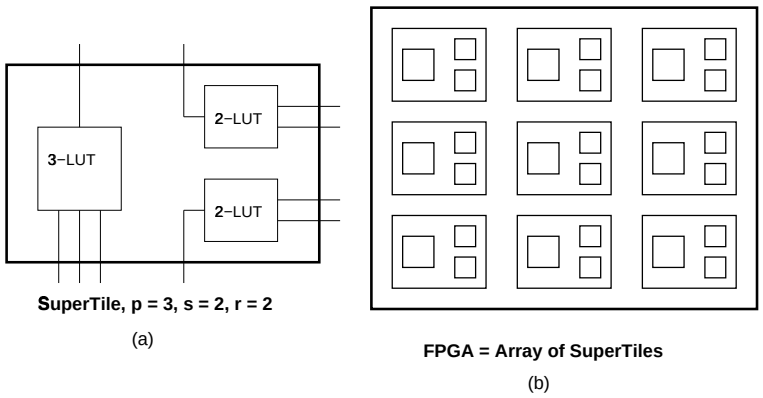

Figure 3: Example Supertile and Heterogenous FPGA

mize. For example, if $r = 1$, then for *every* p-LUT that is used in M, an s-LUT *must* be used by the mapper, or else it is wasted. This is different from standard technology mapping into an ASIC library, in which a mapper is free to choose *any* number of each kind of library element.

It is important to note that Figure 3 displays only the *abstraction* of a supertile, and is not meant to speak to the actual positioning or interconnection of the lookup tables. While this is an important issue, our purpose in this work is to explore the benefits of heterogenous architectures at the logic level. Should it prove successful, this will motivate subsequent work on the actual physical design of such an FPGA.

3 A Mapping Algorithm for Heterogeneous FPGAs

To solve the non-linear optimization problem, our general approach is to break it up into a set of linear optimization problems, each of which is more tractable. The essence of the approach is that the network is mapped several times, with different constraints each time: In the first mapping the number of p-LUTs (N_p) in the circuit is constrained to be zero and the number of s-LUTs (N_s) is minimized. In the second mapping, N_p is constrained to be exactly one, and N_s is again minimized. This process continues with the fixed value of N_p increasing by one until the value of N_s achieved reaches zero. This results in several mappings. Given the value of r, one of these mappings will result in the minimum number of supertiles, as defined above, and can be easily determined by calculating N_{sup} for each mapping.

The overall flow of the algorithm is as follows: As in [Keut87], it begins by breaking the boolean network into a forest of fanout-free trees, and each tree is mapped separately. Each tree is mapped several times as described above, resulting in multiple implementations for each tree. This is followed by a multi-tree optimization step which selects the set of mappings, one for each tree, that minimizes the number of supertiles in the entire circuit. The latter algorithm is optimal.

3.1 Mapping a Single Tree

The principal tree mapping technique used in the algorithm is a generalized version of dynamic programming [Corn87]. As in dynamic programming for technology mapping [Keut87], the combinational network is traversed from the inputs of the tree and proceeds to the root.

At each node, a *list* of best circuits is constructed, each of which has a different number of p-LUTs, in the same way as described above for the entire tree. That is, each node is implemented several times, for $N_p = 0$, 1, 2 and so on, while the number of s-LUTs, N_s, is minimized. The circuit list terminates when $N_s = 0$. Each circuit implements the cone extending from the node to the inputs of the tree.

If the node is a leaf, an s-LUT is used since it is smaller than a p-LUT (by definition) and can always be changed into a p-LUT later if that is beneficial. For a non-leaf node, a set of best circuits are constructed from the list of circuits that have already been constructed on its fanin edges. Figure 4 gives the pseudo-code to illustrate the mapping of a single tree.

```
MapTree(tree,p,s)
 { Traverse tree from leaves to root, at each node:
  { if node is a leaf
       BestList[node]⟵ s-LUT
    else BestList[node]⟵ MapNode(node)
     /* BestList will contain one circuit for each N_p & Min(N_s) */
  }
   return(BestList[root])
 }
```

Figure 4: Pseudo-code of Generalized Dynamic Programming for Mapping a Tree

Figure 5 gives pseudo-code for the mapping of a single non-leaf node. The input is a list of best circuits (one circuit for each value of N_p) for each fanin edge. The output is a similar list describing the best circuits (with the fewest s-LUTs) for each value of N_p.

Notice that many different combinations of the fanin circuits will lead to a node circuit that has a fixed value of N_p. For example, suppose there are two fanin edges, a and b, to a node and each of the edges has a list of two circuits, $\{{C_0}^a, {C_1}^a\}$ and $\{{C_0}^b, {C_1}^b\}$, where subscript in each circuit represents its number of p-LUTs. There are three combinations of these fanin circuits that may lead to a node circuit with $N_p = 1$: ${C_1}^a$ & ${C_0}^b$, or ${C_0}^a$ & ${C_1}^b$, or ${C_0}^a$ & ${C_0}^b$. In the last case the p-LUT would be created in the mapping of the node itself; in the two former cases the p-LUTs are inherited from the fanins. Note that once a p-LUT is created it cannot turn back into an s-LUT, because $p > s$. However, an s-LUT can later become a p-LUT.

Since it is not known which of the fanin circuits will result in the very best value of N_s, every possible combination of the input circuit lists is evaluated, and the best is selected.

While this could result in a large number of combinations, in our experience on a range of benchmark circuits, the number is tractable. For the worst case the total number of combinations did not exceed 414,724 and only 13 cases fall in the range $10^4 - 10^6$ for 40

MCNC benchmark circuits. Over 98.7% of the nodes required fewer than 50 combinations of input fanin circuits. This happens most likely because we operate on fanout-free trees, which are typically small.

The outer loop in procedure MapNode is to enumerate each such combination. In each inner loop iteration in procedure MapNode, the desired number of p-LUTs is fixed. The lower limit on this value is the sum of the number of p-LUTs in the immediate fanin circuits. The loop runs until the number of s-LUTs is reduced to zero.

Inside the inner loop, the following problem is solved: given a fixed number of p-LUTs to create, N_p_create, and a fixed set of mapped fanin circuits, map the current node using exactly N_p_create p-LUTs and the minimum number of s-LUTs.

At this point the algorithm uses a bin-packing strategy similar to [Fran91]. The problem to be solved is more difficult, however, because there are two kinds of LUTs to pack the logic into. The following sections describe the *bin-packing* of the fanin circuits into two different sizes of bins, and the final construction of the tree at the current node.

```
MapNode(node)
 { BestList[node] ← empty
   For each combination of fanin circuits to node {
     N_p_in ← total # p-LUTs of immediate fanins
     N_p_create = N_p_in
     While (N_s ≠ 0) {
        /* pack into N_p_create p-LUTs and minimum # s-LUTs */
        Packing ← BinPack(node, N_p_create)
        /* make packed LUTs into a tree */
        Tree ← TreeForm(Packing)
        if Tree best so far with value of N_p, record it in BestList[node]
        N_p_create = N_p_create + 1
      }
     }
 }
```

Figure 5: Pseudo-code for Mapping a Node

3.1.1 Packing Fanin Lists into Heterogeneous LUTs

Francis' Chortle algorithm [Fran91] makes use of a bin-packing algorithm to pack the root LUTs of the fanin circuits and the current node into an optimal tree circuit with the best possible decomposition of the current node. This is based on the observation that only the number of used inputs in the LUTs is important in determining if logic will fit into a LUT. In [Fran91] the fanin root LUTs correspond to "boxes" to be packed, and the results LUTs are the receiving "bins", of size K. We apply the same approach, except that the problem is more difficult because there are now two sizes of bin, p and s.

An illustration of heterogeneous bin packing is given in Figure 7(a) and 7(b), for $N_p\_create = 2$, where $p = 5$ and $s = 4$.

The heterogeneous bin packing problem can be stated as follows: Given a number of p-LUTs to create (N_p_create), and the fanin circuits' root LUTs, pack the fanin root LUTs into exactly N_p_create p-LUTs and a minimum number of additional s-LUTs.

We apply a variation of the first-fit decreasing algorithm: first create the number of p-LUTs that already exist in the fanin root LUTs. Then sort the remaining fanin root LUTs ("boxes") into decreasing order and put it into the first p-LUTs in which it fits. If a new "bin" is needed, create a p-LUT if N_p_create p-LUTs have not yet been created, and otherwise create an s-LUT. Figure 6 gives the pseudo-code outline of this packing algorithm. Although not shown in Figure 6, we also apply the re-convergent fanout optimization described in [Fran91a].

```
BinPacking(node, N_p_create)

{ BoxList⟵ fanin root LUTs sorted by decreasing size
  N_p_in⟵ number of p-LUTs in BoxList

  /* pack p-LUTs into bins of size p because
     they will not fit into s-LUTs, by definition */
  BinList ← N_p_in p-LUTs in BoxList
  N_p_created⟵ N_p_in ( N_p_created is the number of p-LUTS created)

  while (BoxList not empty )
    { box⟵ largest LUT in BoxList

      find first bin in BinList such that
       size(bin) + size(box) ≤ capacity(bin)

      if such a bin doesn't exist create a new bin:
       { if ( N_p_created < N_p_create )
         { create a bin of size p
           N_p_created++
         }
         else create a bin of size s
       }
      pack box into bin
    }
  return (BinList)
}
```

Figure 6: Pseudo-code for BinPacking

3.1.2 Forming a Tree

After the packing of the fanin root LUTs is completed, these packed LUTs are connected to form a tree to realize the current node and its fanins. The LUTs are sorted by decreasing order of number of used inputs and the output of the largest is connected to any unused inputs in the subsequent bins. The purpose of this procedure is to make the root node have as many unused inputs as possible. This is beneficial because the unused inputs can be utilized by subsequent nodes, as described in [Fran91a].

j	0	1	2	3	4	5	6	7
S_j^i	11	9	7	6	4	2	1	0

Table 1: Example Mapping Counts for One Tree

If there are insufficient inputs to connect all the LUTs together, then new s-LUTs are created. Figure 7(c) illustrates the tree forming procedure for the circuit of Figure 7(b).

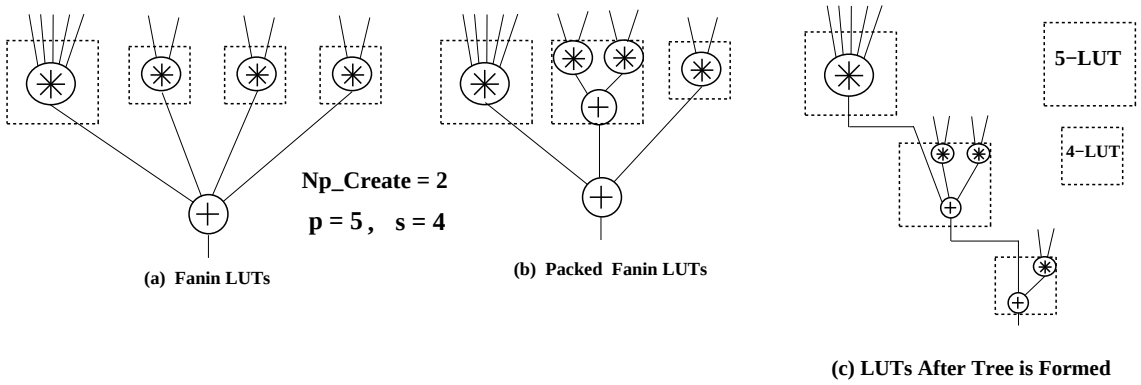

Figure 7: Illustration of Bin Packing and Tree Forming

3.2 Combination of Individual Tree Solutions

After each tree T_i has been mapped, the algorithm has produced a set of circuits $\{ C_j{}^i \}$ where i is the tree number, and j is the number of p-LUTs in the mapped solution for that tree. For each circuit, $C_j{}^i$, let S_j^i be the number of s-LUTs in tree i with j p-LUTs. Table 1 gives an example of several typical values of S_j^i for $p = 5$ and $s = 4$.

Recall that the optimization goal is to find the minimum number of supertiles given by: $N_{sup} = \mathbf{max}(N_p, \lceil \frac{N_s}{r} \rceil)$, where for Table 1, $N_p = j$ and $N_s = S_j^i$.

For each tree N_{sup} can be easily calculated simply from N_p and N_s in each mapped solution, and the best selected. For example, from Table 1, if $r = 1$, then the solution with $N_p = 4, N_s = 4$ (4 supertiles) table entry is minimal and the corresponding circuit should be selected. If $r = 2$, then $N_p = 3, N_s = 6$ (3 supertiles) is minimal.

While this is simple for a single tree, the problem becomes more difficult when optimizing the number of supertiles across a number of trees. It may be that one tree is efficient using mostly s-LUTs and a second tree is better with mostly p-LUTs. If the number of supertiles in the trees were optimized individually, as above, this advantage may never be realized, since the above procedure seeks to balance the s-LUTs and $p-$LUTs according to ratio r on an individual tree basis.

A naive algorithm, however, that evaluates all possible combinations of table entries across all trees has enormous complexity. If the maximum number of table entries per tree is m, and there are n trees, then the number of evaluations is m^n which is intractable. In

	S_0	S_1	S_2	S_3	S_4	S_5
T_1	5 ($C^1{}_0$)	3 ($C^1{}_1$)	2 ($C^1{}_2$)	0 ($C^1{}_3$)		
T_2	4 ($C^2{}_0$)	2 ($C^2{}_1$)	0 ($C^2{}_2$)			
T_1 combined with T_2	5 ($C^1{}_0$) 4 ($C^2{}_0$)	3 ($C^1{}_1$) 4 ($C^2{}_0$)	2 ($C^1{}_2$) 4 ($C^2{}_0$)	0 ($C^1{}_3$) 4 ($C^2{}_0$)		
		5 ($C^1{}_0$) 2 ($C^2{}_1$)	3 ($C^1{}_1$) 2 ($C^2{}_1$)	2 ($C^1{}_2$) 2 ($C^2{}_1$)	0 ($C^1{}_3$) 2 ($C^2{}_1$)	
			5 ($C^1{}_0$) 0 ($C^2{}_2$)	3 ($C^1{}_1$) 0 ($C^2{}_2$)	2 ($C^1{}_2$) 0 ($C^2{}_2$)	0 ($C^1{}_3$) 0 ($C^2{}_2$)
$MS^{1\cup 2}$ ($T_1 \cup T_2 \mid_{MS}$)	9 ($C^1{}_0C^2{}_0$)	7 ($C^1{}_1C^2{}_0$)	5 ($C^1{}_1C^2{}_1$)	3 ($C^1{}_1C^2{}_2$)	2 ($C^1{}_3C^2{}_1$)	0 ($C^1{}_3C^2{}_2$)

Table 2: Example of Tree Combination

the next section we present an algorithm to solve this problem optimally with complexity $O((m \times n)^2)$.

3.2.1 Algorithm for Selection of Tree Mappings

We will first illustrate the basic algorithm on the combination of two trees, which have the family of solutions $\{C_j{}^1\}$ and $\{C_j{}^2\}$. Let the number of s-LUTs be $S_0{}^1$, $S_1{}^1$, $S_2{}^1$, ... $S_{k_1}{}^1$ for the first tree, and $S_0{}^2$, $S_1{}^2$, $S_2{}^2$, ... $S_{k_2}{}^2$ for the second tree, where k_1 and k_2 are integers. When the two trees are taken together, we need to determine $MS_k{}^{1\cup 2}$, the smallest number of s-LUTs for a fixed number of $p-$LUTs, k, where $k = 0, 1, 2, ...k_1 + k_2$ and $MS_k{}^{1\cup 2} = Min(S_k{}^{1\cup 2})$. $MS_0{}^{1\cup 2}$ can be found by summing $S_0{}^1$ and $S_0{}^2$. The value of $MS_1{}^{1\cup 2}$ is given by **Min**($S_0{}^1 + S_1{}^2$, $S_1{}^1 + S_0{}^2$). Similarly, for higher values of k, $MS_k{}^{1\cup 2}$ can be determined by finding the minimum sum of all possible pairs of $S_x{}^1$ and $S_y{}^2$, for which $k = x + y$.

Table 2 gives an example of this calculation for two small trees. The rows labelled T_1 and T_2 provide the values of $\{S_j{}^1\}$ and $\{S_j{}^2\}$. The next three rows show the possible combinations that provide the corresponding entries of $S_j{}^{1\cup 2}$. The final row gives the best of these combinations (that with the smallest number of s-LUTs). Note that if there is a tie, the circuit first encountered is chosen.

This algorithm produces the optimal combination of two trees. Proof of the optimality is omitted due to space limitations. It can extended to multiple trees by combining subsequent trees, forming, in turn, $MS^{1\cup 2\cup 3}$, $MS^{1\cup 2\cup 3\cup 4}$, ... , and $MS^{1\cup 2\cup ...\cup n}$. Using this final table, the optimal number of supertiles can be determined for a given ratio, r, by applying the above equation for N_{sup}. The pairwise comparison of each pair of tables will pessimistically require no more than every pair being compared, and hence has complexity $O((m \times n)^2)$, where m is the total number of possible circuits in all of the trees.

Note that the application of this part of the algorithm is not limited to either lookup tables or trees, but can be applied to family of solutions for sets of sub-circuits of any type.

4 Experimental Results

In this section we compare the quality of the heterogenous mapping algorithm with the best alternative approach we could find: a post-processing of the output from a homogenous LUT mapper.

The latter method works as follows: to map an architecture with a given p, s, and r, the homogeneous mapper is executed to map into LUTs of size p. For each LUT produced, let the number of inputs that are actually used be U (note that not all inputs of every LUT will be completely used). The LUTs for which $U = p$ are mapped directly into $p-$LUTs, resulting in N_p p-LUTs. The LUTs for which $U \leq s$, are mapped directly into s-LUTs, resulting in N_s s-LUTs. In order to achieve the ratio r between N_s and N_p the LUTs for which $s < U < p$ are mapped either as s-LUTs or $p-$LUTs depending on the number that is needed to achieve the balance. Following this, if more $p-$LUTs are needed ($N_p < \lceil \frac{N_s}{r} \rceil$) then to achieve ratio r, the requisite number is simply converted into $p-$LUTs because a p-LUT can implement the function of any s-LUT. If more s-LUTs are needed to achieve the balance, then existing $p-$LUTs are replaced by a tree of s-LUTs. We refer to this algorithm as post-process-homo (PPH). Note that this post-process step only improves the quality of the heterogenous mapping, never making it worse.

We used the public domain version of Chortle [Fran91] as the basis for comparison since both the homogeneous mapper and the heterogeneous one use a similar mapping process. Both the new heterogenous mapper and the PPH mapper were run on 40 MCNC logic synthesis benchmarks circuits, and for many different values of s and p. The total number of supertiles obtained for each case was calculated. Note that Chortle was set to exclude its fanout optimization, as the algorithm presented here does not employ such an optimization, and we sought to make a fair comparison of the general approach.

Table 3 gives sample results for the heterogenous architecture $p = 5, s = 2$, and $r = 4$. The first column of this table gives the circuit name, and the second column gives the supertile count for the PPH algorithm. The third column gives the number of supertiles using the algorithm described in this paper and the fourth column gives the percentage difference between the two. For this example, the new algorithm achieves on average 31% fewer supertiles than the PPH algorithm for this architecture. The running times are usually a few seconds and not more than a minutes minutes on a SUN Sparcstation 2.

Figure 8 shows how the improvement of the new algorithm over PPH varies as a function of the ratio, r. This figure plots the percentage improvement, over all 40 circuits, of the number of supertiles versus ratio r for the 5-LUT/2-LUT combination. Observe that there is increasing advantage as the number of smaller LUTs in the supertile increases. This makes intuitive sense: at the extreme when the number s-LUTs is zero, then the problem is equivalent to the homogenous case. As the number of s-LUTs increases, the mapping problem becomes more heterogenous, and the algorithm that is designed directly for that case becomes significantly superior. For the ratio of 10 for this example architecture, the average improvement is 48%.

Circuit Name	#Supertiles (PPH)	#Supertiles (New Algorithm)	Percentage Difference
C1355	33	33	0.0
C432	42	32	-24
C880	48	37	-23
alu2	64	44	-31
alu4	105	74	-30
apex6	124	79	-36
apex7	37	24	-35
b9	16	13	-19
c8	17	13	-24
cht	29	17	-41
cm150a	8	5	-38
cm151a	4	3	-25
cm85a	6	4	-33
cmb	8	5	-38
count	15	13	-13
example2	43	32	-26
frg1	21	13	-38
frg2	131	86	-34
i1	7	5	-29
i6	61	43	-30
i7	88	57	-35
i8	175	112	-36
i9	111	66	-41
k2	138	99	-28
my-adder	22	16	-27
parity	3	3	0
pcler	10	8	-20
pm1	7	5	-29
rot	90	68	-24
sct	10	8	-20
t481	3	3	0
term1	21	14	-33
ttt2	23	16	-30
unreg	19	11	-42
vda	76	56	-26
x1	50	34	-32
x2	7	6	-14
x3	133	83	-38
x4	52	35	-33
z4ml	4	3	-25
total	1861	1278	-31

Table 3: Comparison of Heterogeneous Mapper and PPHo for $p = 5, s = 2$, and $r = 4$.

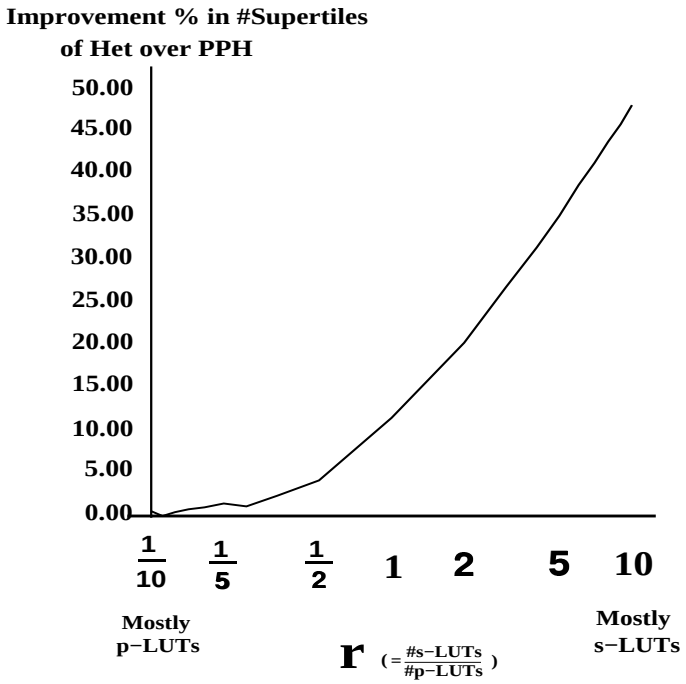

Figure 8: Improvement of New Algorithm over PPH as a function of r

5 Conclusions and Future Work

This paper has motivated the problem of technology mapping for heterogenous logic blocks in FPGAs, and presented an algorithm for its solution. The main features of this algorithm are its ability to handle heterogenous logic blocks directly through a generally applicable paradigm, and the optimal solution to the sub-problem of combining solutions from a group of sub-circuits. The new algorithm achieves average savings up to 48% in area over a post-process of a homogenous mapper, depending on the particular heterogenous architecture.

This mapping tool has already been used in a study to determine the area-efficiency advantage of heterogenous FPGAs over homogenous FPGAs. The final version of this paper will include a reference to a published paper on this subject. (It is excluded now only for reasons of blind review).

In the future, we will look at mapping heterogenous FPGAs for delay, and use this to investigate advantages in the tradeoff between speed and area for heterogenous FPGAs. Other future work includes the investigation of algorithms for FPGAs that use three or more different types of block, and algorithms for the automatic placement of heterogenous FPGAs.

References

[Abou90] P. Abouzeid, *et al*, " Lexicographical Expression of Boolean Function for Multilevel Synthesis of High Speed Circuits," *Proc. SASHIMI 90*, Oct. 1990, pp.31-39.

[Ahre90] M. Ahren, *et al*, "An FPGA Family Optimized for High Densities and Reduced Routing Delay," *Proc. 1990 CICC* , May 1990, pp.31.5.1 - 31.5.4

[Bray90] R. Brayton, *et al*, "Multilevel Logic Synthesis," *Proc. IEEE*, Vol.78, Feb. 1990, pp.264-300.

[Chen92] K. Chen, "Logic Minimization of Lookup-Table Based FPGAs," FPGA'92, 1992, pp.71-76.

[Cong92] J. Cong, *et al*, "Graph Based FPGA Technology Mapping for Delay Optimization," Proc. FPGA'92, 1992, pp. 77-81.

[Cong93] J. Cong, Y. Ding, "On Area/Depth Trade-off in LUT-Base FPGA Technology Mapping," Proc. DAC'93, 1993, pp. 213-218.

[Corn87] D. Corneil, *et al*, "A Dynamic Programming Approach to the Dominating Set Problem on k-Trees", *SIAM J. ALG. Disc Meth.*, Vol.8, No.4, Oct.1987, pp. 535-543.

[Filo91] D. Filo, *et al*, "Technology Mapping for a Two-Output RAM-based Field Programmable Gate Array",*Proc. EDAC91,* Feb. 1991,pp. 534-538.

[Fran90] R. Francis, *et al*, "Chortle: A Technology Mapping Program for Lookup Table-Based Field-Programmable Gate Arrays," *Proc. DAC90,* June 1990, pp. 613-619.

[Fran91a] R. Francis, *et al*, "Chortle_crf:Fast Technology Mapping for Lookup Table_Based FPGAs", *Proc. DAC91*, June, 1991, pp. 227 - 233.

[Hsie90] H. Hsieh, *et al*, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," *Proc. 1990 CICC*, May 1990, pp. 31.2.1-31.2.7

[Karp91] K. Karplus, "Xmap:A Technology Mapper for Table-Lookup Field- Programmable Gate Arrays",*Proc. DAC91,*June 1991,pp. 240-243.

[Keut87] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching",*Proc. DAC87*, June 1987, pp. 341-347.

[Murg90] R. Murgai, *et al*, "Logic Synthesis for Programmable Gate Arrays," *Proc. DAC90,* June 1990, pp. 620-625.

[Murg91a] R. Murgai, *et al*, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD,* 1991.

[Sawk92] P. Sawkar and D. Thomas, "Technology Mapping for Table-Look-Up based Field Programmable Gate Arrays," Proc. FPGA'92, 1992, pp. 83-88.

[Woo91] N. Woo, "A Heuristic Method for FPGA Technology Mapping Based on Edge Visibility," *Proc. DAC91,*June, 1991,