# TEMPT: Technology Mapping for the Exploration of FPGA Architectures with Hard-Wired Connections

Kevin Chung & Jonathan Rose
Department of Electrical Engineering
University of Toronto
CANADA  M5S 1A4

## Abstract

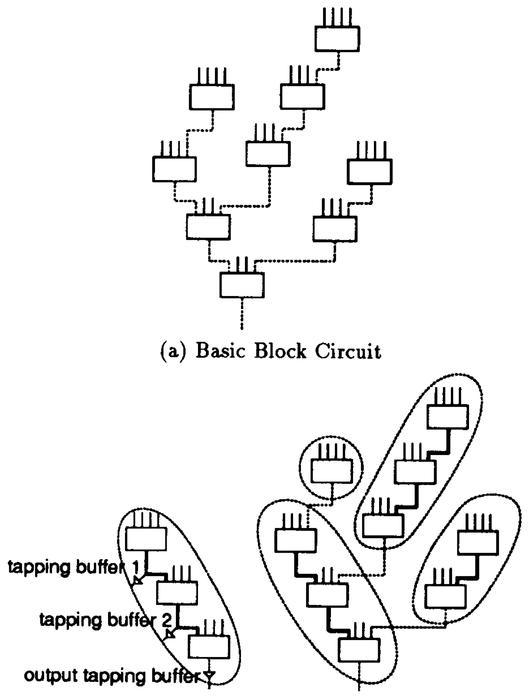
*This paper describes TEMPT, a technology mapping algorithm aimed at exploring FPGA architectures with hard-wired connections. Such FPGA architectures may be important because hard-wired connections are much faster and smaller than the programmable connections between basic logic blocks that they replace.*

*TEMPT maps a network of basic blocks to a netlist of hard-wired logic blocks (HLBs), in which each HLB consists of several basic blocks hard-wire connected in an arbitrary tree topology, and optimizes either speed or area. TEMPT is as effective as the Xilinx 4000 CLB mapper, PPR, when minimizing CLBs to implement a set of MCNC benchmarks. Using TEMPT we demonstrate empirically how many HLBs are significantly faster than FPGAs without hard-wired links. Also, we demonstrate several HLBs that exhibit superior logic density to the Xilinx 4000 CLB*[1].

## 1 INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are an increasingly popular means of implementing Application Specific Integrated Circuits (ASICs) because designers can manufacture their ASIC cheaply and in minutes. The user configures the FPGA logic and routing connections by a simple programming technology (such as downloading bits into static RAM cells [Hsie90] or blowing anti-fuses [ElGa89]). This user-programmability of routing connections, however, makes FPGAs slower and less dense than Mask-Programmable Gate Arrays because of the significant capacitance, resistance and size of the programmable switches. This routing delay and area can be significantly reduced by replacing some of the programmable connections with hard-wired connections, which are metal wires that have almost-zero delay and consume little area [Chun91] [Sing91]. For example, Figure 1(a) illustrates a network of basic logic blocks with five slow programmable connections in the routing along the critical path[2]. Suppose three basic blocks are hard-wired together, as in Figure 1(b), to make a hard-wired logic block (HLB) with a fast path through 3 basic blocks. Using this HLB to implement the circuit in Figure 1(a) results in the faster circuit shown in Figure 1(c)[3]. This faster circuit has only two

[1]This work was supported by an ITRC Fellowship, NSERC Operating Grant #URF0043298 & a Xilinx Inc. research grant.



(a) Basic Block Circuit

(b) An HLB (c) Faster HLB Circuit

Figure 1.

[2]A dotted line represents a programmable connection and only output connections along the path are counted.

[3]The hard-wired links are shown in thick lines and the hard-wire connected sets of basic blocks are circled by dashed ovals.

0738-100X/92 $3.00 © 1992 IEEE

programmable links instead of five along the critical path, giving a significant reduction in routing delay. However, the inflexibility of hard-wired links in the circuit can lead to unusable basic blocks, resulting in lower logic density.

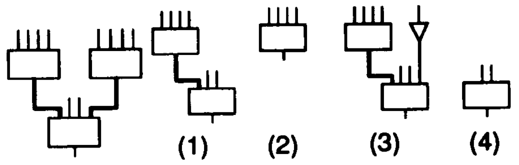
The presence of hard-wired connections provides several reasons for the introduction of a new synthesis tool. First, there already exists a commercial architecture [Hsie90] with hard-wired connected basic logic blocks. Second, hard-wired connected basic blocks present a new synthesis problem because they present a strong link between logic synthesis and routing. The third and most important motivation is to aid in the exploration of FPGA architectures with hard-wire connected basic blocks. In [Chun91] [Sing91] we investigated the speed/area tradeoffs of an FPGA architecture composed of an array of Hard-wired Logic Blocks (HLBs), where an HLB is several basic logic blocks (such as lookup tables or NAND gates) connected by metal wires in a tree topology. This study showed that certain HLB topologies result in a 44% decrease in the average number of programmable connections along the critical path compared to an FPGA without hard-wired connections, yet only reduce the logic density (measured by the total number of basic blocks) by 6% [Chun91]. The results in [Chun91] [Sing91] were obtained empirically using logic synthesis tools to implement benchmark circuits. This paper presents one of the tools, TEMPT, a technology mapper for tree-connected hard-wired logic blocks.

This paper is organized as follows. Section 2 presents some assumptions and terminology. Section 3 describes the TEMPT technology mapping algorithm for both delay and area optimization. Section 4 shows the results of using TEMPT to map to several different HLB architectures.

## 2 ASSUMPTIONS & DEFINITIONS

An important architectural assumption is that each basic block output in an HLB has a *tapping buffer* that makes the output accessible to the routing. Figure 1(b) shows an HLB with three 4-input basic blocks hard-wired together in a cascade. Tapping buffers can lead to faster HLB circuits since the output of one basic block can be accessed directly instead of propagating it through another basic block. Tapping buffers can also improve logic density because unrelated pieces of logic can be packed together in the same HLB.

The inputs to TEMPT consist of a description of the HLB topology and a network of basic blocks. In the remainder of this paper, we consider only lookup table (LUT) basic blocks since they exhibit reasonable area and delay in FPGAs without hard-wired connections [Rose90] [Sing91]. Also, there is an FPGA [Hsie90] with hard-wire connected LUTs and an associated mapper for testing and comparison. The LUT basic block network is generated



(a) L2-3 HLB (b) L2-3 HLB Fragments

Figure 2: The L2-3 HLB and its Fragments

using the following two steps: (1) logic optimization of the MCNC benchmark circuit using mis2.2 [Bray86], and (2) mapping to LUTs using Chortle [Fran91a] [Fran91b].

Synthesizing HLB circuits is a new problem because HLBs are a combination of logic and routing resources. Thus, the mapping to HLBs can occur during either logic *or* layout synthesis. In TEMPT, we chose to perform the assignment of hard-wired links in the HLBs during logic synthesis because the goal of our previous work [Chun91] [Sing91] was an exploration of FPGA logic block architectures with no assumption of specific routing architectures. The use of a placement and routing algorithm would require more detailed specification of the routing architecture, whereas mapping to HLBs during technology mapping abstracts out physical layout details. This abstraction for technology mapping also allows a tighter focus on optimizing the use of the hard-wired connections for each HLB and may lead to better optimization for HLBs.

### 2.1 HLB Fragment Patterns

An HLB fragment is a connected subset of the basic blocks in an HLB [4]. Since we assume tapping buffers make every basic block output in an HLB accessible, any HLB fragment that matches a portion of the basic block network can be used to implement that portion. An HLB fragment that matches part of the basic block network rooted at a specific node is said to be *feasible* at that node. Since HLB fragment DAGs are trees, the feasible HLB fragments can be quickly found using tree pattern matching [Keut87].

Given an HLB topology described by a tree, the HLB fragment patterns are generated by one of the following operations on the tree. The first operation deletes a hard-wired link edge between two basic blocks (this assumes that a hard-wired input of the downstream block can be ignored) and the second operation converts a basic block into a buffer. Note that LUTs allow both of these operations, but a basic block such as a 4-input NAND gate does not. Figure 2 shows the L2-3 HLB[5] in Figure 2(a) and its

[4]Note that the complete HLB is also a fragment

[5]The HLB name is the height of the HLB followed by a listing of the sizes of the subtrees from a pre-order traversal. Subtrees of size 0 or 1 are not listed.

generated fragments in Figure 2(b). HLB fragment number 1 is produced by deleting an input edge of the root basic block of the HLB. At the same time the complementary HLB fragment (numbered 2) composed of a single 4-input basic block is also created. Fragment number 3 is made by using one of the basic blocks that feeds the root block to implement a buffer. The deletion of the edge between the two basic blocks of fragment 1 yields a new HLB fragment (number 4), which is a two-input basic block.

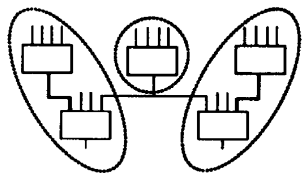
## 3 TEMPT MAPPING ALGORITHM

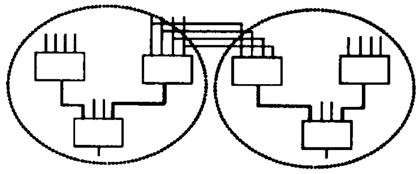
TEMPT uses two separate algorithms for optimizing the delay or area of the netlist of HLBs that implement the functionality of the input basic logic block network. The goal of the delay algorithm is to find a set of feasible HLB fragments that minimize the delay along the critical path. This delay-optimized set of fragments is then packed together into a minimum number of HLBs to reduce the area overhead. The goal of the area-optimization algorithm is to find the set of feasible HLB fragments that when packed together yields the minimum number of HLBs.

### 3.1 Delay Optimization

As in [Keut87], the DAG representing the input basic block network is first partitioned into a forest of trees. Each network tree is traversed in postorder fashion and the feasible HLB fragments at each node are found using tree matching. The optimal delay cover of feasible fragments for the tree is determined using dynamic programming [Keut87]. A straightforward merging of the trees to reproduce the final network would give sub-optimal delay performance because optimization across tree roots with fanout $> 1$ is prevented. This limitation is overcome by selectively replicating portions of the basic block netlist that fanout to two or more nodes. For example, Figure 3(a) shows a circuit that has been implemented in the HLB of Figure 2(a) when no replication across fanout is allowed. The circuit in Figure 3(a) has two HLBs along the critical paths and thus there are two programmable connections in the critical path. Replication of the basic block in the centre of Figure 3(a) into its fanouts leads to the faster implementation in Figure 3(b) with only one programmable connection delay. However, the number of basic blocks in the new network has increased from five to six.

The primary cost function in the dynamic programming and tree matching algorithm is the delay measured as the maximum number of HLBs (and hence the number of programmable connections) between the primary inputs and the node. If two feasible HLB fragment patterns lead to solutions with the same delay, the one with the lower height (or if the heights are also the same, the one with smaller size) is chosen. The height of an HLB fragment is the maximum number of basic logic blocks from the fragment inputs to the output. The size is the number of basic blocks in the HLB fragment. These secondary cost functions lead to an HLB fragment at the root of each tree with minimal delay with respect to the tree inputs and minimal height and size. Small root HLB fragments lead to fewer replicated basic blocks when replication at tree roots is needed.


(a) Circuit implementation without Replication


(b) Circuit implementation with Replication
Figure 3.

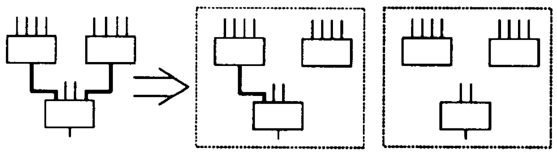


Figure 4: The L2-3 HLB and its Packing Sets

Each HLB fragment produced by the delay optimization algorithm can fit in an HLB. However, allotting one fragment per HLB is excessive, so the delay-optimization algorithm packs the HLB fragments into as few HLBs as possible using the algorithm outlined in Section 3.1.1.

#### 3.1.1 Packing HLB Fragments

The packing algorithm takes a list of HLB fragments and packs them into the smallest number of HLBs. A set of HLB fragments that can be packed together into a single HLB is called a *packing set* of HLB fragments and every packing set is a subset of a maximal packing set.

The maximal packing sets are generated using the same deletion and buffering operations as in Section 2.1. Figure 4 illustrates how the edge deletion operation generates two maximal packing sets for the HLB from Figure 2(a). Each maximal packing set is enclosed in a dotted rectangle. The first edge deletion generates a maximal packing set with two HLB fragments, a two-basic block fragment

```
PackHLBfrags()
    PackedHLB_list ← $\phi$
    UnpackedFrags ← descendingSort(UnpackedFrags)
    while UnpackedFrags not = $\phi$
        packedHLB ← $\phi$
        while UnpackedFrags not = $\phi$
            addedFrag ← next (UnpackedFrags)
            packedHLB ← packedHLB ∪ addedFrag
            if packedHLB is a maximal packing set
                exit inner loop
            else if packedHLB is not a packing set
                return addedFrag to UnpackedFrags
            else if addedFrag is last of UnpackedFrags
                exit inner loop
            end if
        end while
        Packed_list ← Packed_list ∪ packedHLB
    end while
```

Figure 5: HLB Fragment Packing algorithm

plus a single 4-input basic block fragment. Deleting the edge from the two-basic block fragment generates the second maximal packing set, which consists of two 4-input basic blocks and one 2-input basic block. Note that the HLB fragments in a maximal packing set can be packed in the same HLB because every basic block output in an HLB is accessible via tapping buffers.

The pseudo-code for the packing algorithm is illustrated in Figure 5. Since smaller fragments can fit more easily into an HLB with other fragments it is more effective to pack them last; thus, the packing algorithm begins by sorting the unpacked HLB fragments into a descending list according to size. While the unpacked list is not empty, the outer loop of the packing algorithm constructs packed HLBs with the fragments from the list in a greedy manner.

A new packed HLB is initially empty. The inner loop scans the unpacked fragment list from largest to smallest, greedily adding any unpacked fragments that can fit in the unused part of the new packed HLB. The set of fragments can fit in a packed HLB if it is a subset of a maximal packing set. Unpacked fragments are added until one of three conditions holds: (1) The set of fragments form a *maximal packing set* and thus the packed HLB can hold no more fragments; (2) The unused capacity of the packed HLB is too small to accommodate any of the remaining unpacked fragments or (3) There are no remaining unpacked fragments. At this point the "filled" packed HLB is added to the packed HLB list and if there are remaining unpacked fragments, a new packed HLB is started.

```
FindMinimumAreaMapping()
    initially there are only single-block HLB fragments
    pack single-block fragments to find global area cost
    if global area cost satisfies lower bound
        exit mapping algorithm
    end if
    for every node $n$ in the basic block network
    traversed in pre-order
        for every feasible HLB fragment $f$ at $n$
            pack $f$ with all other HLB fragments
            if $f$ leads to lowest global area cost so far
                record $f$ as best frag. at $n$ and area cost
                if area cost satisfies lower bound
                    exit mapping algorithm
                end if
            end if
        end for
        if the lowest area-cost feasible HLB fragment
        pattern at $n$ is not a single-block fragment
            make the basic blocks in the lowest cost
            HLB fragment into a "fixed HLB"
        end if
    end for
```

Figure 6: Area-optimization algorithm

## 3.2 Area Optimization

The area-optimization mapping algorithm seeks to minimize the number of HLBs in the final HLB netlist. Each feasible HLB fragment at a node in the basic block network corresponds to a potential assignment of hard-wired links between basic blocks. The goal of the algorithm is to select the set of feasible fragments (and corresponding hard-wired links) that pack together to yield the fewest HLBs. A lower bound on the number of packed HLBs is $\left\lceil \frac{\text{number of basic blocks in network}}{\text{number of basic blocks in an } HLB} \right\rceil$ and whenever this lower bound is achieved, the mapping algorithm ends.

In order to do a packing to determine global area cost every node in the basic block network has to be assigned to some HLB fragment. Thus, the algorithm described in Figure 6 starts with each basic block being mapped to a single-block HLB fragment, that is, the initial HLB network has no hard-wired links. We chose this initial assignment because small fragments can fit more easily into unused portions of an HLB during packing. The mapping algorithm then proceeds to add hard-wired links by choosing feasible multi-block HLB fragments at nodes in the basic block network that lead to a better packing. The nodes in the network are accessed in pre-order. Each feasible multi-block HLB fragment at a node is packed with the other previously constructed multi-block fragments and any remaining single-block fragments. Note that since there is no tree partitioning, matches for feasible fragments can occur across fanout. This leads to an increase in the number of feasible fragments at some nodes and can lead to better optimization. After packing, if the best global area cost so far is reduced then this feasible HLB fragment becomes the best fragment at the node. After all feasible HLB fragments for the node have been evaluated, the lowest area cost fragment is fixed permanently so that the nodes in the best fragment cannot be assigned to a different fragment. When all nodes have been visited the algorithm ends.

The primary global area cost function is the number of HLBs after packing. However, if two sets of HLB fragments have the same number of packed HLBs, the set with the largest HLB hole (i.e. number of unused logic blocks in an HLB) is chosen. If the two sets are still tied, then the one with fewer unpacked HLB fragments wins. The largest HLB hole cost function is used because a large hole can be more easily filled, during packing, by an unpacked fragment. Having fewer HLB fragments is also useful because it means a greater total number of hard-wired connections are in the fragments. Since the packing of fragments into the same HLB leads to the wastage of hard-wired connections, more hard-wired connections in the fragments mean that fewer are needed for packing them together.

## 4 RESULTS

In this section, we first compare TEMPT to the commercial technology mapper for the Xilinx 4000 (X4000) CLB [Hsie90], which is an FPGA with hard-wired connections. To illustrate how TEMPT can be used for FPGA architecture exploration we compare the X4000 CLB to other HLBs for area-efficiency. We also present a comparison of several HLB topologies when optimizing purely for speed. More extensive results comparing HLBs from a speed and area perspective are in [Chun91] [Sing91].

### 4.1 Comparison with Xilinx mapper

In this experiment, the goal was to minimize the number of X4000 CLBs used to implement a set of 15 MCNC benchmark circuits. The circuits were chosen because they could fit in the largest available X4000 CLB package. Each benchmark was optimized by mis2.2 [Bray86]. For TEMPT, the mis2.2-optimized circuits were first mapped by Chortle-crf [Fran91a] to the minimum number of 4-input LUTs and then TEMPT minimized the number of CLBs to cover the LUT network. The input to the Xilinx mapper, *PPR* was also the mis2.2-optimized circuits. The results quoted for PPR, are the *Packed CLB* utilization number from the PPR report file. The Packed CLB value is the CLB utilization if PPR were to pack the design into as small an area as possible. Table 1 lists the name of each benchmark circuit in Column 1. Columns 2 and 3 list the number of X4000 CLBs for PPR and TEMPT.

| Bench Cct | PPR CLBs | TEMPT CLBs |
|---|---|---|
| 9symml | 36 | 35 |
| alu2 | 71 | 69 |
| alu4 | 122 | 118 |
| apex7 | 38 | 38 |
| b9 | 20 | 20 |
| c1355 | 91 | 103 |
| c8 | 17 | 18 |
| cc | 8 | 10 |
| cm162a | 5 | 6 |
| comp | 17 | 16 |
| count | 21 | 16 |
| decod | 10 | 10 |
| mux | 5 | 7 |
| vda | 97 | 97 |
| z4ml | 3 | 3 |
| Totals | 561 | 566 |

Table 1: Xilinx mapper vs. TEMPT

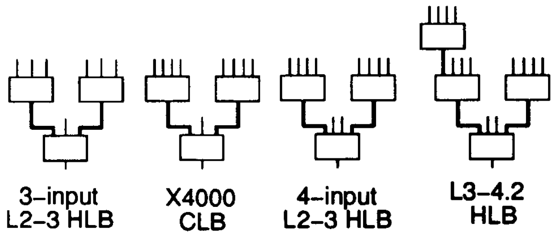


Figure 7: Area Study HLB Topologies

For the set of 15 benchmark circuits, the X4000 mapper required 561 CLBs and TEMPT used 566 CLBs. The two mappers are close in effectiveness even though TEMPT is not well-suited to mapping the X4000 CLB for the following reasons: (1) The X4000 CLB consists of both 3- and 4-input LUTs but since Chortle does the mapping to a homogeneous network of LUTs our methodology is better-suited for homogeneous LUT HLBs, and (2) The X4000 CLB violates the tapping buffer assumption since it allows only 2 tapping buffers for 3 LUTs. TEMPT had to be modified to allow this restriction.

### 4.2 Area-efficiency Results

In this section, we demonstrate how TEMPT may be used to evaluate the area-efficiency of the different HLB architectural alternatives illustrated in Figure 7. The X4000 CLB is an example of an L2-3 HLB topology with two 4-input LUTs and one 3-input LUT. The other L2-3 HLBs investigated are composed of three 3-input LUTs and three 4-input LUTs respectively. Comparing these three L2-3 HLBs is interesting because the functionality of the L2-3 HLB with 3-input LUTs and the functionality of the L2-3 HLB with 4-input LUTs bracket the functionality of

| Bench Cct | 3-inp L2-3 | X4000 CLB | 4-inp L2-3 | 4-inp L3-4.2 |
|---|---|---|---|---|
| 9symml | 40 | 36 | 26 | 19 |
| alu2 | 77 | 71 | 48 | 37 |
| alu4 | 126 | 122 | 82 | 61 |
| apex7 | 44 | 38 | 27 | 20 |
| b9 | 22 | 20 | 16 | 12 |
| c1355 | 131 | 91 | 80 | 59 |
| c8 | 21 | 17 | 15 | 11 |
| cc | 17 | 8 | 9 | 7 |
| cm162a | 8 | 5 | 5 | 4 |
| comp | 27 | 17 | 14 | 13 |
| count | 21 | 21 | 14 | 10 |
| decod | 10 | 10 | 8 | 8 |
| mux | 10 | 5 | 5 | 5 |
| vda | 96 | 97 | 70 | 52 |
| z4ml | 3 | 3 | 3 | 2 |
| **Tot HLBs** | 653 | 561 | 421 | 320 |
| **LUT Bits** | 15672 | 22440 | 20208 | 20480 |
| Ratios | 0.70 | 1 | 0.90 | 0.91 |
| **HLB pins** | 6530 | 6171 | 5473 | 5440 |
| Ratios | 1.06 | 1 | 0.89 | 0.88 |

Table 2: Area Measures of Different HLBs

the X4000 CLB. The L3-4.2 HLB topology was also evaluated. This topology is interesting because the three 4-input LUTs nearest the root LUT form an L2-3 HLB and the fourth LUT makes a fast three basic-block level path.

Table 2 compares the HLBs in Figure 7 when Chortle and TEMPT were used in area-optimizing mode to map the benchmark circuits. For the X4000 CLB, the PPR numbers in Table 1 were used. Each row in the table corresponds to individual circuit data and the last 4 columns gives the number of HLBs needed by each HLB topology. The last five rows of Table 2 summarize the individual data. The first summary row of Table 2 shows the total number of each HLB required to implement the set of benchmark circuits. The second and fourth row are interesting architectural measures derived from the number of HLBs. The second row compares the total number of bits in the LUTs of the HLBs needed to implement the benchmark set. This is a measure of the active silicon area required to implement the logic. The fourth row is the total number of pins (one pin for each LUT input and tapping buffer output in the HLB) for the required HLBs. A previous architectural study [Rose90] demonstrated a correlation between the total number of pins and the routing area. The third row shows the ratio of the number of bits relative to the X4000 CLB. The last row shows the ratio of the number of pins relative to the X4000 CLB.

These results show that the L2-3 HLB composed of only 4-input LUTs (Column 4) uses fewer LUT bits and fewer pins than the X4000 CLB (Column 3). These architectural measures indicate that this HLB will have superior logic density. Also, the L2-3 HLB composed of 3-input LUTs (Column 2) uses much fewer bits and slightly more pins than the X4000 CLB. The architectural measures for the L3-4.2 HLB (Column 5) show it is about as area-efficient as the L2-3 HLB composed of 4-input LUTs. However, the delay results given in Section 4.3 and [Chun91] show that, when optimizing for delay, the L3-4.2 architecture has on average 12% fewer programmable connnections along the critical path than the L2-3 HLB, and so the L3-4.2 architecture is the better overall architecture.

The absolute effectiveness of the area-optimization algorithm can be evaluated by comparing the results in Table 2 to the lower bound in Section 3.2. The number of HLBs using TEMPT is within 3% and 4% of this lower bound for the L2-3 and L3-4.2 HLBs consisting of 4-input LUTs. For the L2-3 HLB consisting of 3-input LUTs and the X4000 CLB, the differences are 13% and 33% respectively. However, it should be noted that the lower bound is optimistic for the X4000 CLB because it consists of two 4-input and one 3-input LUT and the lower bound assumes all HLB LUTs have the same functionality.

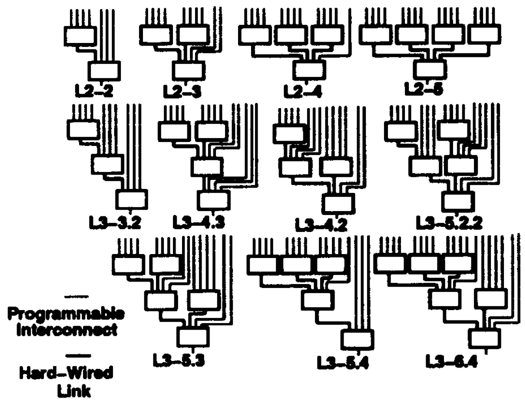


Figure 8: Delay Study HLB Topologies

## 4.3 Pure Delay Optimization Results

In this section, we demonstrate how TEMPT may be used to evaluate the speed improvement for different HLB topologies. A sampling of the many HLB topologies considered are illustrated in Figure 8. Each HLB consists of several 4-input lookup tables hard-wired together. The depth-optimized LUT networks were generated by Chortle-d [Fran91b] and the HLB netlists were generated using the delay-optimization algorithm of TEMPT.

The total critical path delay, $D_{tot}$, is given as $D_{tot} = N_{LB} \times D_{LB} + N_R \times D_R$. The first product in this expression is the delay due to the basic blocks along the critical path. Here $N_{LB}$ is the number of basic logic blocks and $D_{LB}$ is the delay of each basic block. For our $1.2\mu m$

| Logic Block | $\overline{N_R}$ | % decr in $\overline{N_R}$ | $\overline{D_{tot}}$ (ns) | % decr in $\overline{D_{tot}}$ |
|---|---|---|---|---|
| K4 | 5.4 | 0 | 30 | 0 |
| L2-2 | 4.2 | 22 | 26 | 13 |
| L2-3 | 3.4 | 37 | 22 | 27 |
| L2-4 | 3.1 | 43 | 21 | 30 |
| L2-5 | 3.0 | 44 | 21 | 30 |
| L3-3.2 | 4.0 | 26 | 25 | 17 |
| L3-4.2 | 3.0 | 44 | 21 | 30 |
| L3-4.3 | 3.1 | 43 | 21 | 30 |
| L3-5.2.2 | 3.1 | 43 | 21 | 30 |
| L3-5.3 | 3.0 | 44 | 21 | 30 |
| L3-5.4 | 2.9 | 46 | 20 | 33 |
| L3-6.4 | 2.8 | 48 | 20 | 33 |

Table 3: Delay Performance of Different HLBs

CMOS technology, the delay for a 4-input lookup table, $D_{LB} = 1.7ns$. The second product is the routing delay. Here $N_R$ is the number of programmable connections in the critical path and $D_R$ is the average delay per routing connection. $D_R$ varies widely according to interconnection architecture and in [Chun91] and [Sing91] we explored the effects of various values of $D_R$. For Table 3 an intermediate value of $D_R = 4\ ns$ was chosen. In Table 3 the % decrease in $\overline{N_R}$ and $\overline{D_{tot}}$ are relative to K4, an FPGA with 4-input LUT basic blocks with no hard-wired connections.

From Table 3, the L2-3 HLB reduces $\overline{N_R}$ to 3.4, while L3-4.2 HLB reduces $\overline{N_R}$ to 3.0, a difference of 12% in the average number of programmable connections in the critical path. Compared to an FPGA without hard-wired links, the L2-3 HLB and L3-4.2 HLB reduce $\overline{N_R}$ by 37% and 44% respectively and reduce average total critical path delay by 27% and 30%. The greatest speed gains per added basic block are realized by the L2-2 and L2-3 HLB and the speed gains are very small, per added basic block, for HLBs with more than four basic blocks. In general, for HLBs with a specific number of basic blocks, symmetric topologies attained superior speed than asymmetric topologies. This is likely because speed-optimized basic block networks tend to exhibit more balanced tree-structures.

## 5 CONCLUSION

A technology mapping algorithm that can be used for exploring FPGA architectures with hard-wired connections has been presented. It is capable of optimizing for speed or area and can map to any tree-topology hard-wired logic block FPGA. The mapper is comparable to a commercial CAD tool when mapping to the Xilinx 4000 CLB, a commercial FPGA with hard-wired connections.

The utility of this mapper for FPGA architectural exploration was shown using two studies, one that compared the area-efficiency of several HLB topologies and another that compared the speed performance of several HLB topologies. In the area-efficiency study, it was demonstrated that a hard-wired logic block consisting of three 4-input LUTs is superior to the Xilinx 4000 CLB in terms of both logic and routing area. The mapper was also used to show that another topology, the L3-4.2 HLB topology consisting of four 4-input LUTs, was similar in area-efficiency to the best of the 3-LUT HLB topologies. Also, the L3-4.2 HLB topology was shown to be faster than the 3-LUT HLBs in the speed performance study.

A future improvement to TEMPT is to integrate the delay and area algorithms to allow trade-offs between delay and area. Placement costs will also be considered in the packing step. More architectural studies on HLBs with basic blocks other than 4-input LUTs will be performed.

ALGORITHMS
ARCHITECTURES

## REFERENCES

[Bray86] R. Brayton, et al., "Multiple-Level Logic Optimization System," *Proc. of Int. Conf. on CAD 1986 (ICCAD-86)*, Nov. 1986, pp. 356-359.

[Chun91] K. Chung, S. Singh, J. Rose, P. Chow "Using Hierarchical Logic Blocks to Improve the Speed of Field-Programmable Gate Arrays." in *FPGAs - Proc. of 1st Int. Workshop on Field Programmable Logic and Applications*, Sept. 1991, pp. 103-113.

[ElGa89] A. El Gamal, et. al, "An Architecture for Electrically Configurable Gate Arrays," *IEEE Jour. of Solid State Ccts. (JSSC)*, Vol. 24, No. 2, Apr. 1989, pp. 394-398.

[Erco91] S. Ercolani, G. De Micheli, "Technology Mapping for Electrically Programmable Gate Arrays," *Proc. of 28th Design Automation Conf. (DAC-28)*, June 1991, pp. 234-239.

[Fran91a] R. Francis, J. Rose, Z. Vranesic "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. of DAC-28*, June 1991, pp. 227-233.

[Fran91b] R. Francis, J. Rose, Z. Vranesic "Technology Mapping of Lookup Table-Based FPGAs for Performance," *Proc. of ICCAD-91*, Nov. 1991, pp. 568-571.

[Hsie90] H. Hsieh, et al., "Third-Generation Architectures Boosts Speed and Density of Field-Programmable Gate Arrays," *Proc. of Custom IC Conf. 1990 (CICC-90)*, May 1990, pp. 31.2.1-31.2.7.

[Keut87] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," *Proc. of DAC-24*, June 1987, pp. 341-347.

[Rose90] J. Rose, R. J. Francis, D. Lewis, P. Chow, "Architectures of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE JSSC*, Vol. 25, No. 5, Oct. 1990, pp. 1217-1225.

[Sing91] S. Singh, J. Rose, D. Lewis, K. Chung, P. Chow "Optimization of Field Programmable Gate Array Logic Block Architecture for Speed," *Proc. of CICC-1991*, May 1991, pp. 6.1.1-6.1.6.