Reducing the Pressure on Routing Resources of FPGAs with Generic Logic Chains

Hadi Parandeh-Afshar[†]
hadi.parandehafshar@epfl.ch

Grace Zgheib[‡]
grace.zgheib@lau.edu.lb

Philip Brisk[§]
philip@cs.ucr.edu

Paolo Ienne[†]
paolo.ienne@epfl.ch

[†]Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences, 1015 Lausanne, Switzerland

[‡]Department of Electrical and Computer Engineering
Lebanese American University, Byblos, Lebanon

[§]Department of Computer Science and Engineering
University of California Riverside, 900 University Ave., Riverside CA92521, U.S.A.

ABSTRACT

Routing resources in modern FPGAs use 50% of the silicon real estate and are significant contributors to critical path delay and power consumption; the situation gets worse with each successive process generation, as transistors scale more effectively than wires. To cope with these challenges, FPGA architects have divided wires into local and global categories and introduced fast dedicated carry chains between adjacent logic cells, which reduce routing resource usage for certain arithmetic circuits (primarily adders and subtractors).

Inspired by the carry chains, we generalize the idea to connect lookup tables (LUTs) in adjacent logic cells. By exploiting the fracturable structure of LUTs in current FPGA generations, we increase the utilization of the existing LUTs in the logic cell by providing new inputs along the logic chain, but without increasing the I/O bandwidth from the programmable interconnect. This allows us to increase the logic density of the configurable logic cells while reducing demand for routing resources, as long as the mapping tools are able to exploit the logic chains. Our experiments using the combinational MCNC benchmarks and comparing against an Altera Stratix-III FPGA show that the introduction of logic chains reduce the average usage of local routing wires by 37%, with a 12% reduction in total wiring (local and global); this translates to improvements in dynamic power consumption of 18% in the routing network and 10% overall, while utilizing 4% fewer logic cells, on average.

Categories and Subject Descriptors

B.6.1 [**LOGIC DESIGN**]: Design Styles—*Logic arrays, Combinational logic*; B.7.1 [**INTEGRATED CIRCUITS**]: Types and Design Styles—*Gate arrays*

General Terms

Design, Performance

Keywords

FPGA, Logic Chain, Routing Wire, Dedicated Connection, Generic Synthesis

1. INTRODUCTION

The programmable interconnect fabric dominates silicon area in modern high-performance FPGAs. The fraction of silicon dedicated to programmable routing increases with each successive technology generation, because transistors scale more effectively than wires. This trend directly impacts the performance and power consumption of FPGAs. Moreover, the feasibility of synthesizing a circuit onto an FPGA can be limited by the availability of routing resources, rather than programmable logic.

To reduce the negative impact of routing resources, a number of architectural innovations have been proposed in recent years. One approach, was to divide the routing network into global and local components [4, 22]. This enabled to have fast local routing within clusters of logic (e.g., *Logic Array Blocks, or LABs* in Altera's FPGAs) and reduced the demand for global routing resources between the clusters. The introduction of carry chains within logic clusters allowed for the efficient propagation of arithmetic carries along a fixed wire, native to the carry chain; consequently, these wires were completely moved out of the programmable interconnect.

This paper introduces a new logic cell architecture that reduces the demand for programmable routing resources when utilized effectively by the mapper. The key idea is to establish dedicated connection wires between adjacent logic

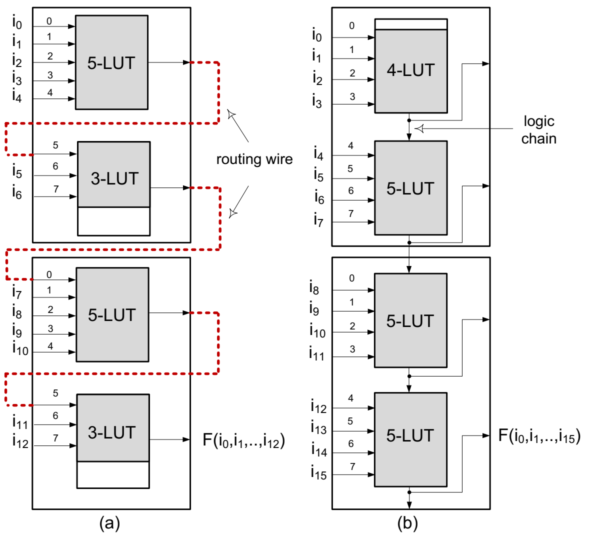

Figure 1: Key idea. (a) Two logic cells, each has 8 inputs and two base 5-LUTs. Many 13-input logic functions can be mapped to a linear cascade of the base LUTs; routing resources are required to connect adjacent LUTs in the cascade. (b) A dedicated logic chain between adjacent LUTs eliminates the overhead due to routing resources and increases the input bandwidth of logic cell. Many 16-input logic functions can be mapped with the same number of available LUTs.

cells; this so-called *Logic Chain* is similar in principle to a carry-chain, but connects programmable logic resources rather than fixed-function gates that can only perform carry-propagate addition. Through effective decomposition and mapping algorithms, it is possible to map logic functions of non-trivial size onto the dedicated logic chains, rather than using programmable interconnect. This reduces pressure on the routing network, as nets from the technology-mapped circuit are moved onto dedicated wires in the logic chain.

The other benefit of the logic chain is that it increases the utilization of LUTs in the logic cell by providing more bandwidth, but without increasing the I/O bandwidth of the cell in terms of connections to the programmable routing network. This allows us to improve the logic density of the logic cells by making better usage of the available resources. The logic chain is implemented to reuse as many of the circuit elements within the logic cell as possible; a small number of additional multiplexers and configuration bits must be added to augment the logic cell with the new logic chain.

The following section begins with a simple example to illustrate the main idea.

1.1 Key Idea

The motivation behind logic chains comes from the observation that many technology mapped circuits contain linear chains of LUTs after mapping. Based on our experiments, on average, 85% of the LUTs are chainable. Suppose that a circuit has been synthesized in such a manner that four LUTs are cascaded linearly and form a chain. Figure 1 illustrates how this circuit is differently implemented on two different FPGAs, one with a conventional logic cell and the other with the new logic cell that has the dedicated logic chain.

In Figure 1 (a), there are two FPGA logic cells, each having 8 inputs and a fracturable LUT structure. In this fracturable LUT structure, each logic cell has two base 5-LUTs and a bigger 6-LUT is formed using the two sub-LUTs followed by a multiplexer, which is not shown in this figure.

One drawback of this particular architecture is that the routing resources must be used to connect one sub-LUT to its successor in the cascade. The other drawback is that the second sub-LUT in each logic cell is under utilized due to the logic cell input bandwidth constraint. The first 5 inputs are used by the first sub-LUT and the remaining 3 inputs are used by the second sub-LUT. This means that a 5-LUT is used to implement a 3-input function.

Figure 1 (b) has the dedicated logic chains within the cluster so that the output of each sub-LUT connects directly to the input of the subsequent sub-LUT along the chain; in principal, this is similar to the interconnection structure of arithmetic carry chains in commercial FPGAs today. The introduction of these direct connections eliminates the need to use the global routing network to synthesize the cascade. This has several advantages: reduced pressure on the routing network, reduced critical path delay and reduced power consumption. Moreover, the logic chain provides a way that the available sub-LUTs are utilized more efficiently, since the input bandwidth of the logic cell is increased by the logic chain without any change in the local routing network of the logic array block.

Comparing these two figures, we see that with the new logic cell we can map bigger functions to the same number of logic cells. The logic cells in Figure 1 (a) can implement many functions with 13 inputs, while the logic cells in Figure 1 (b) can implement many 16-input functions.

2. RELATED WORK

Different and restricted types of LUT chains exist in some FPGA devices from both *Altera* and *Xilinx* families. Logic Cells (LC) in *Stratix* and *Cyclone* devices from *Altera*, have a local connection which connects the output of one LC's LUT to the input of the adjacent LC [4]. These connections allow LUTs within the same LAB to cascade together for wide input functions. Conceptually our proposed logic chain is similar to the mentioned local chains, but there are some fundamental differences. The main difference is that in contrast to the above FPGAs, we do not use the available input bandwidth of the LC to connect the logic output of the adjacent LC. This will increase the available bandwidth and hence wider functions can be implemented without any need to change the LC interface. The other difference is that the LC in current FPGA devices has a *fracturable* LUT structure and this allows to use the available LUT resources in a LC to implement larger functions considering our logic chain as the extra input.

Xilinx FPGAs also have a kind of local connections between the adjacent LCs, which goes through a number of multiplexers in each LC [22]. This local connection is mainly used for implementing carry look-ahead adders, but it can also be exploited for mapping of a limited number of generic functions. In its most general case, it can be used to implement the AND cascade of functions. For instance, a wide input AND function can be partitioned into some parts that are mapped to the LUTs and cascaded through the local connection. In contrast to such FPGAs, our logic chain is more general. The proposed logic chain goes through a LUT and forms the last input of the LUT; therefore, no specific logic constraint exist for cascading different functions.

Constructing bigger LUTs by cascading smaller ones is also possible in *Virtex-5 Xilinx* FPGAs. There are some multiplexers in the *Virtex-5* LC for this purpose and by using such multiplexers, we can build up to 8-input LUTs. However, the routing wires are required to connect smaller LUTs and also there is a concern about the feasibility and usefulness of synthesizing a circuit onto such big LUTs. Prior research [2] indicates that a LUT size of 4 to 6 provides the best area-delay product for an FPGA.

In [9], an FPGA chip was developed with the logic blocks that are comprised of cascaded LUTs. In this work, each logic block has three 4-input LUTs hard-wired together for high performance. In contrast to our design, the hard-wired connection is exclusively limited to the logic blocks internal and does not cross the logic blocks boundaries.

Other relevant ideas consist in introducing carry chains into modern high-performance FPGAs and developing advanced technology mapping algorithms that attempt to exploit carry chains.

The vast majority of carry chains that have been proposed are for different types of adders [8, 14, 17, 18, 19]; the carry chains on commercial FPGAs available from Xilinx and Altera also fall into this category. One interesting alternative is a carry chain that allows an Altera-style logic cell to be configured as a 7:2 compressor, which is used for multi-operand addition [20].

Similar to our work, one recent paper has presented a non-arithmetic carry chain in which two 2-LUTs are combined to form a 3-LUT [15]; however, it was based on a carry-select structure used in Altera's Stratix, which has since been deprecated. Starting with the Stratix II, Altera's carry chains have employed a ripple-carry structure.

There have been a handful of papers that have successfully mapped operations other than 2- or 3-input addition (or subtraction) onto the carry chains of commercial FPGAs. In a previous contribution [21], we mapped generalized parallel counters, also used for multi-operand addition, onto the carry chains used by Altera's Stratix III series FPGAs, which are still in use today. This approach target a limited set of logic functions (multi-operand addition) and cannot map general logic functions onto carry chains.

The ChainMap algorithm attempts to map arbitrary logic functions onto the carry chain of the Altera Stratix and Cyclone FPGAs [16]; as mentioned above, this carry chain has been deprecated and the authors readily admit that their algorithm is not applicable to newer Altera FPGAs or Xilinx FPGAs. Our chaining heuristic does share some principle similarities with ChainMap, but targets the logic chain that we have proposed rather than carry chains.

Traditional formulations of the technology mapping problem focus on converting a structural HDL implementation of a circuit into a network of K-cuts, where each K-cut can be mapped onto a single K-LUT. These formulations assume that the programmable routing network is used to connect the LUTs; it does not attempt to use carry chains, fracturable LUTs, embedded multipliers, or DSP blocks; likewise, these formulations could not account for the fixed wiring structure in the logic chain proposed here. Cong and Ding proved that minimizing the number of LUTs on the longest path can be done in polynomial time [10]; several others have proven that the decision problems corresponding to minimizing the total number of LUTs used in the covering and minimizing power consumption are NP-complete [12, 13]. Many heuristics to solve different variations of the technology mapping problem have been presented over the years; there are far too many to enumerate here.

Additionally, several papers have tried to perform logical decompositions to optimize the structural circuit description in conjunction with technology mapping [7][11]; as logical optimization is NP-complete in the general case, this formulation of the problem is NP-complete as well, although the use of decomposition can significantly improve the quality of the technology mapping that can be achieved. In principle, this type of decomposition and of technology mapping approach would be appropriate for use with the logic chains proposed here; the decomposition could exploit the specific fixed interconnect structure between adjacent LUTs on the logic chain; this approach is likely to be more effective than what we have done here: searching for chainable candidates in a technology mapping solution that was produced by a more general technology mapping algorithm that was unaware of the presence of the logic chains.

3. NEW LOGIC CHAIN

Figure 2 (a) illustrates the structure of *Altera*'s *Stratix-III Adaptive Logic Module (ALM)* configured as two independent 5-LUTs, which have two shared inputs; the two shared inputs are necessary because the ALM has an input bandwidth constraint of 8. Consequently, if the user wishes to map two logic functions without shared inputs onto an ALM, the only possibilities are to use two 4-LUTs or a 5-LUT and a 3-LUT.

Figure 2 (b) illustrates a way that an ALM can realize a limited subset of 7-input logic functions: the 5-LUT is cascaded with the 3-LUT, however, the interconnection between the two requires the usage of the routing network. On the other hand, if the user wants to cascade two 5-LUTs with one another, then two ALMs are required, as shown in Figure 2 (c), once again using the routing network; if the two ALMs are placed within the same *Logic Array Block (LAB)*, then the fast local routing network could be used instead of the global routing network.

Altera's ALM is fracturable, meaning that several small LUTs (sub-LUTs) exist natively in the ALM and can be concatenated together, via multiplexers, to form larger LUTs. This paper uses this approach to build larger LUTs out of the sub-LUTs along the dedicated vertical connection that we call a *logic chain.* The basic idea is to re-use the current LUTs in the ALM and add a few multiplexer to provide a way to form larger LUTs with a fixed interconnection pattern. Figure 2 (d) illustrates the main idea. The modified logic cell now contains two 5-LUTs that are cascaded; one input of each of the 5-LUTs comes from the preceding 5-LUT in the logic chain; thus, only 8 input signals are provided from the routing network, keeping the design within the bandwidth constraints of the ALM. This allows the new logic cell to implement a subset of 9-input logic functions, without requiring the routing network and assuming that one of the inputs comes from the preceding logic cell along the logic chain; if the vertical input is unavailable, then it

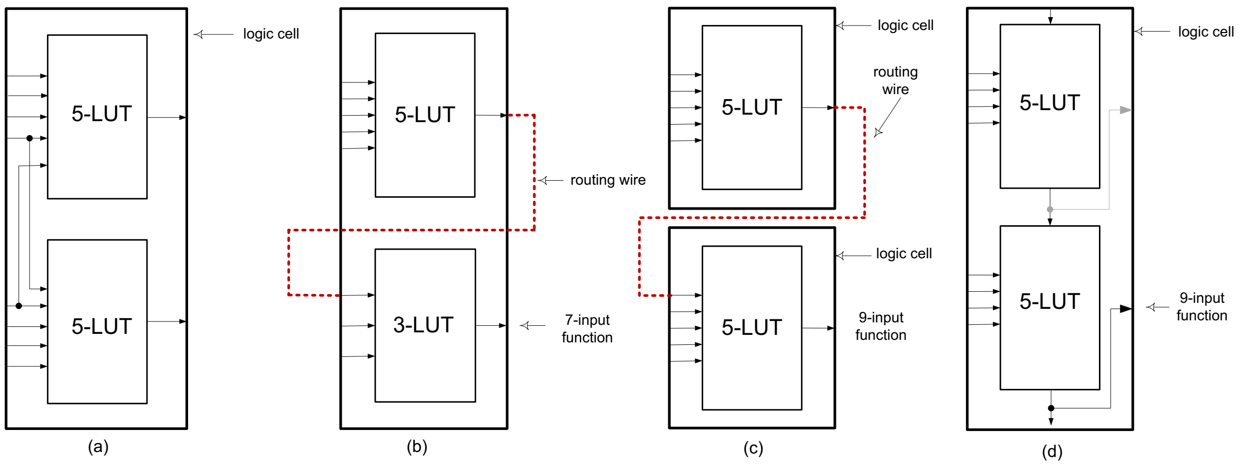

Figure 2: (a) Altera's ALM configured to implement two 5-input logic functions; the ALM imposes the constraint that the two functions must share two inputs. (b) Using fracturable LUTs, a subset of 7-input logic functions can be synthesized on an ALM, but this requires routing a signal from one sub-LUT to the next. (c) To implement two cascaded 5-input functions with no common inputs, two ALMs are required. (d) All three of the preceding logic functions can be synthezed on the proposed logic cell using the logic chain and without using the global routing network; moreover, the proposed cell can implement a subset of 9-input logic functions.

can still implement a subset of 8-input logic functions without using the routing network. This is significantly more powerful than the ALM, which can implement any 6-input logic function and limited 7-input crossbar switch without using the routing network.

The logic chain borrows many ideas from arithmetic carry chains, that also employ vertical connections between adjacent logic cells. The goal of carry chains, however, is to improve the resource usage and critical path delay of addition/subtraction operations, which are common, but limited. One of the key benefits of these carry chains was that carry propagation was performed along the vertical connections and therefore did not enter the routing network. This reduced contention for routing resources and also reduced critical path delay and power consumption, as the wires in the vertical connection are shorter and do not have additional delays caused by configuration elements placed periodically along them. By integrating LUTs into the vertical connections instead, it is possible to synthesize a wide variety of operations, including addition/subtraction, onto the logic chains.

Figure 3 shows how the fracturable structure of ALMs is used to embed the logic chain and form bigger LUTs along the logic chain. Each half-ALM contains two 4-LUTs which can form a 5-LUT using a multiplexer controlled by a fifth input; all inputs between the two 4-LUTs are shared. This design is effectively a Shannon decomposition. The shaded area of the figure illustrates the logic chain. Using a similar idea, we instantiate a vertical multiplexer, which is controlled by the logic chain, at the outputs of each pair of 4-LUTs; this forms a new 5-LUT along the logic chain. This provides us with the option to form either a horizontal or a vertical 5-LUT in each half-ALM. The output of the vertical 5-LUT propagates along the logic chain.

In Figure 3 there is no way to access the output of the LUTs that are placed in the logic chain; this severely limits the ability to use the logic chain when a LUT placed in the chain has a fanout that exceeds one. The FPGA already contains several multiplexers on its output: one to select between the LUT and carry chain outputs; another to optionally select the flip-flop's output, allowing for sequential circuits. We add an additional multiplexer, as shown in Figure 4 to select between the carry chain output and the logic chain output. The shaded area in the figure indicates the additional logic that we add to the half-ALM to support the logic chains. The additional multiplexer that we have added will not increase the critical path delay of the non-arithmetic modes of the ALM, since it does not lie along those paths.

To estimate the area overhead of the new logic, we count the number of transistors in the logic that we added and compare it coarsely with the number of transistors in a simplified ALM. We have added four multiplexers and two configuration bits (SRAMs); based on the components that are known to exist already in the Stratix-III ALM architecture [4], we are confident that the area overhead is less than 3%.

4. CHAINING HEURISTIC

The objective of the mapping heuristic is to identify chains of logic having the maximum possible lengths. The input is a *Direct Acyclic Graph(DAG)*, where each node represents a logic function and each edge represents the input and output dependencies among the functions. The DAG is generated after technology mapping, so each node is a prospective function that can map onto the LUTs. The number of inputs (K) of each node in the DAG cannot exceed the number of LUT inputs; each node has K child nodes and one or more parents based on the fanout of the node output.

The mapping heuristic visits the DAG nodes in *Depth First Search (DFS)* order, starting from the outputs and

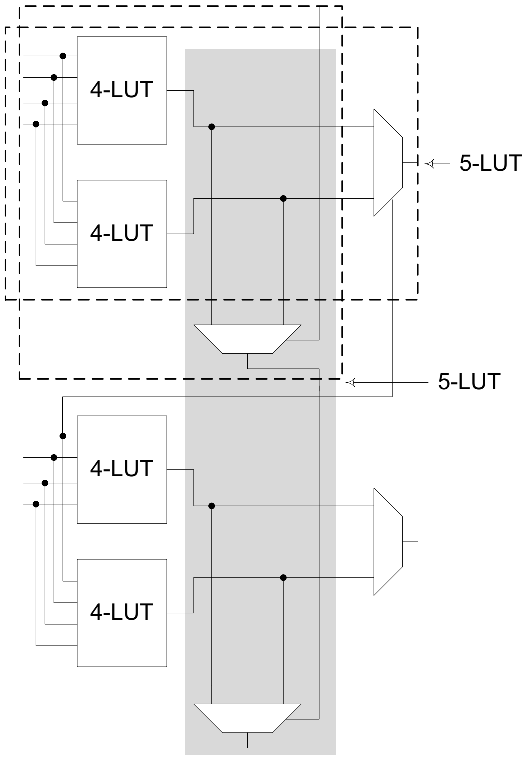

Figure 3: Integrating the logic chain into the structure of an ALM. The shaded area indicates the logic chain. Existing 4-LUTs are used to form vertical 5-LUT on the logic chain using a multiplexer. The fifth input is the vertical logic chain. This figure also shows how a 5-LUT is built in an ALM using smaller LUTs. The key point is that the ALM input bandwidth remains the same, therefore two cascaded 5-LUT with no shared inputs can be mapped to the new cell.

working back toward the inputs. The heuristic recursively assigns a depth to each node in the DAG.

Definition 4.1. *A node is* ***chainable****, if it has at most K inputs and is not part of another chain.*

Definition 4.2. *The* ***depth*** *of a node is the number of chainable nodes that can be accessed consecutively through that node; the* ***depth*** *of an internal node is the maximum depth among all input nodes from which it is reachable.*

Once a depth has been assigned to all nodes, then we can select specific nodes to map onto the logic chain. In particular, we search for the longest chain of nodes in the DAG, which is a chain whose head node has the maximum depth. This chain is then mapped onto a logic chain in the FPGA; the head of the chain can be either a DAG output or a child of a node that is not chainable.

Figure 5 (a) provides a simple example. In this figure, there are two chain candidates, each having a length of 5; the chains intersect at node *N2*. Since each node can be part of one chain, *N2* is arbitrarily chosen for one of the chains.

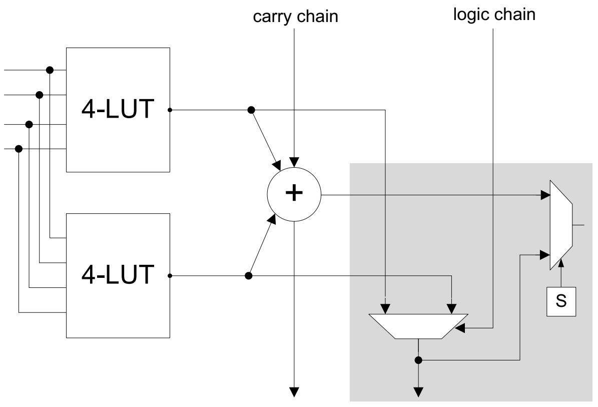

Figure 4: The logic chain integrated with the carry chain. In addition to the vertical multiplexer, a horizontal multiplexer selects between the output of the full adder and the logic chain fanout; this multiplexer is required when the logic chain has fanout.

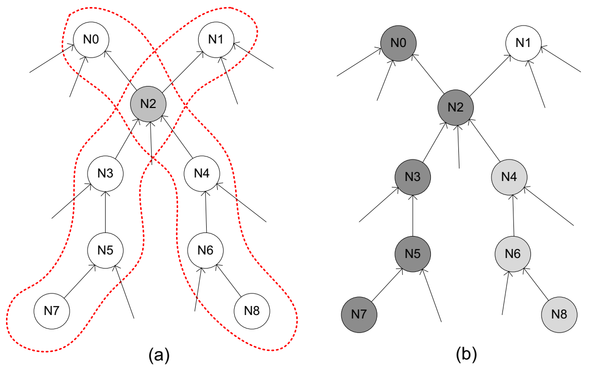

Figure 5: (a) Two chains intersecting at a shared node. (b) The shared node is assigned to one of the chains, breaking the other chain into two smaller sub-chains.

As shown in Figure 5 (b), the second chain is then broken into a chain of 3 nodes and a singleton node, which itself is not part of a chain

Figure 6 presents the chaining algorithm. The loop in the main function recursively traverses the DAG using the DFS, starting from the outputs, and computes the depth of each node. In this approach, first the depths of each node's inputs are computed and then the maximum depth value is increased by one and assigned as the node depth. For the nodes that are either primary inputs or non-chainable, zero is considered as the depth. Figure 7 illustrates an example, in which each node is marked by its depth. The shaded nodes are not chainable, since they are already assigned to other chains.

The depth information then allows the heuristic to identify the logic chains using the *sort_chains* function. The head node of a chian is the one that has the highest depth. All of the nodes in the selected chain are marked as *CHAINED*, to avoid placing a node in more than one chain. This process

```
Chaining_Heuristic(pDAG) {
  while(!termination_condition) {
    for(i=0 to nDAGOutputs) {
      Find_Node_Depth_Rec(pDAG->out[i])
    sort_chains();
    mark_nodes_in_longest_chain();
  }
}
//-----------------------------------------------------------------------
int Find_Node_Depth_Rec(pNode) {
   for(i = 0 to nLeaves-1) {
      if(pNode->Leaves[i] == DAGInput) {
         pNode->depth[i] = 0;
         break; // go to the next leaf
      }
      else
         pTmpNode = pNode->Leaves[i];

      depth = Find_Node_Depth_Rec (pTmpNode );
      if(pNode->chainable)
         pNode.depth[i] = depth + 1;
      else
         pNode.depth[i] = 0;
   } // end of for loop
   pNode->max_depth = pNode->Find_Max_Depth();
   return  pNode->max_depth;
}
```

Figure 6: Pseudo-code of the Chaining Heuristic.

repeats until the maximum length of all remaining chains is less than some threshold value. The time complexity of the heuristic is $O(nh)$, where n is the number of nodes in the DAG and h is the depth of the DAG.

5. TOOL CHAIN FLOW

Figure 8 presents the tool chain flow that we use for our experiments. First, we synthesize each benchmark using *Quartus-II*, a commercial tool provided by Altera; Quartus-II generates a *Verilog Quartus Mapping (VQM)* file netlist; parsing the VQM file yields a DAG-based circuit representation that is fed to the chaining heuristic. Next, the *Verilog* writer routine is called to generate an atom-level netlist for the new mapped circuit. In terms of logic, the Verilog netlist is equivalent to the original VQM netlist; however, the mapping of some cells have bee changed. Lastly, the new netlist is placed and routed by Quartus-II targeting an Altera Stratix-III FPGA. The number of resources that are utilized are obtained for each benchmark. Additionally, the timing revision tool analyzes the timing report produced by Quartus-II and delay numbers are extracted. The *PowerPlay Early Power Estimator* tool is employed to extract the power consumption. Details of the key steps are presented in the following subsections.

5.1 DAG Generator

Quartus-II synthesizes the benchmarks and maps them onto LUTs. Quartus-II's synthesizer generates the VQM file in ASCII text, which contains a node-level (or atom-level) netlist. Since our proposed logic cell is a modified version of the Stratix-III ALM, we felt that Quartus-II was the most appropriate mapper to use. We also considered the possibility of using Berkeley's ABC synthesis tool [1]; however, ABC

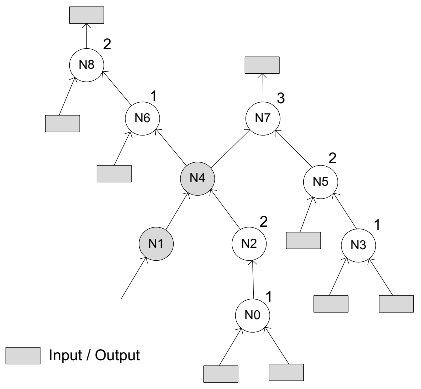

Figure 7: The depths of different nodes in a sample DAG. The shaded nodes are part of other chains and hence not chainable.

is a more general technology mapper and does not consider many ALM-specific features, such as fracturable LUTs.

We implemented a VQM parser in C++, which produces a DAG corresponding to the netlist. Each node in the DAG corresponds to an FPGA cell in the VQM netlist and the edges between nodes in the DAG represent data dependencies. Each DAG node is a C++ class object. Some of the class members are initialized when the VQM file is parsed and others are initialized by the chaining heuristic: the depth of a node; whether a node has been assigned to a chain; the id of the chain to which a node has been assigned; and the order of the node in the chain.

5.2 Placement and Routing

Once the circuits have been mapped to the new FPGA logic cells using the logic chain, we need to place-and-route the mapped circuit to obtain accurate estimates of the area, critical path delay and power consumption. Our area metric includes the number of logic blocks that are used and also the amount of local and global routing resources required to realize the circuit.

We considered the possibility of modifying VPR [5][6] as our experimental platform; however, two problems caused us to look for other alternatives. Firstly, VPR's architectural model does not include carry chains and its packing, placement and routing algorithms would be unable to handle their presence if they were supported architecturally; secondly, a comparison between VPR and Quartus-II is not meaningful, as these are two completely different frameworks; it is difficult to say whether a disparity in favor of either the baseline FPGA or our modified FPGA would be due to architectural superiority or differences between Quartus-II and VPR.

Fortunately, we discovered a way to let Quartus-II model our new FPGAa logic cell; this allowed us to use Quartus-II's placement and routing algorithms, rather than developing our own algorithms to better exploit the modified ALM architecture that we propose. The basic idea is to leave the ALM structure alone and to simply "pretend" that existing carry chains represent the logic chains that we want to introduce. This is possible because the carry output of the adder is a function of the carry input and four ALM inputs, which is similar in principle to our proposed logic chain in terms of connectivity. Therefore, we can assume that the output

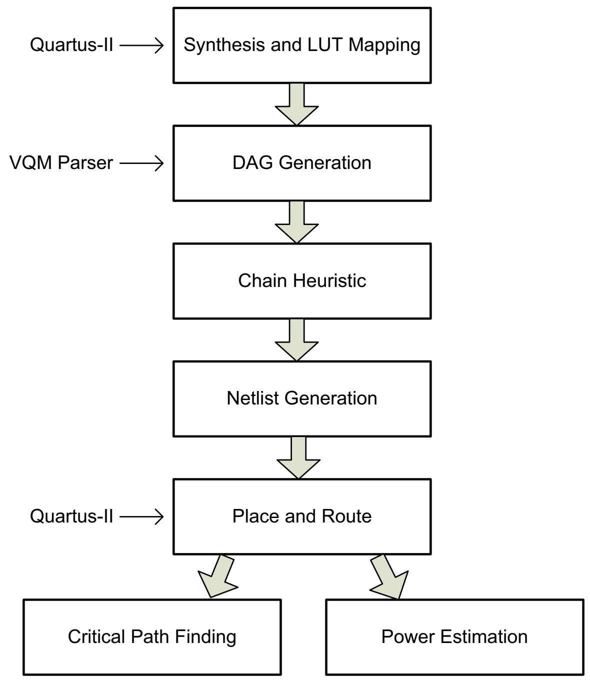

Figure 8: Tool chain flow used for the experiments.

of the vertical 5-LUT in Figure 3 is the carry output of the adder; when we have a fanout that exceeds one, we can take the sum output of the adder as the fanout connection.

Consequently, it is necessary to configure the ALM in a way that the nodes on the logic chain are mapped to the carry chain in a corresponding fashion. To do so, we instantiate the ALM cells and provide the connections and configurations as explained previously; for the other nodes that are not on any chain, we write back the original cell description that was obtained after synthesis to the final netlist. The output, therefore, is a netlist of ALM cells, similar to the original VQM file; nodes that use the logic chain are configured in arithmetic mode, while others are mapped using the ALM's normal mode. Quartus-II then proceeds to place and route the resulting netlist. This gives precise estimates of the usage of logic blocks and routing resources; however, some additional work is required in order to accurate model the critical path delay. To obtain the most dense implementation, we lock the logic in rectangular regions and shrink the region size to the extent that the tool is not able to fit the logic. This guarantees that we have the most packed implementation for both original netlist and the modified one.

5.3 Timing Analysis

As described in the preceding section, we effectively use the carry chains that are present in the Stratix-III FPGA to mimic the logic chains we have proposed for the purposes of placement and routing; however, the critical path delays that are obtained from Quartus-II are based on the delays of the full adders on the carry chains, rather than the multiplexers that we introduced in the logic chain. Therefore, the delay of each adder in the carry chain should be replaced with the delay of the multiplexer instead. Our experiments revealed that the delay between the ALM inputs and the outputs of the adder are significantly greater than the delay of a 5-LUT in the same ALM. Consequently, we take the delay of the normal 5-LUT in the ALM to be the delay of the 5-LUT that is realized in the logic chain; as the result, this reduces the overall delay compared to the timing report produced by Quartus-II.

This analysis has to be performed on a path-by-path basis. The critical path, as identified by Quartus-II's timing report, may no longer be critical once the delays of the logic chain have been properly accounted for. To solve this problem, we repeatedly adjust the delays of subsequent critical paths until we identify a path that includes no nodes mapped onto the logic chains. We wrote a script to perform this task for a specific number of critical paths; the termination condition is to stop when the first path that does not include an adder on the carry chain is found. The paths are then sorted based on their adjusted critical path delays and the maximum is returned as the critical path delay of the circuit synthesized on an Stratix-III style FPGA that has been modified to include our proposed logic chain.

5.4 Power Estimation

To estimate the dynamic power consumption of the mapped circuits, we use *PowerPlay Early Power Estimator* [3] provided by *Altera*. This tool obtains the amount of resources used by each benchmark, the clock frequency after synthesis, the average fanout, the device type and the toggle rate of the wires and estimates the dynamic power consumption of the circuit. The power that is reported is broken down into routing power, logic block power and total power. Here we assume that the dynamic power of the new logic block is approximately equal to the dynamic power of the standard ALM. One potential source of error could be the difference between the toggle rate of the adder output that we use for modeling compared to the toggle rate of the logic in our carry chain. To observe the difference, we modeled the real cell and the ALM in VHDL and applied several stimulause vectors to each and computed the average toggle rates. Our results validated our assumption that toggle rates are approximately equal, on average.

6. EXPERIMENTAL RESULTS

We evaluate the modified ALM-style logic cell with the new logic chain; we consider several factors, including critical path delay, ALM usage, routing resource usage and dynamic power consumption. We used the MCNC benchmarks for our experiments and selected the combinatorial benchmarks exclusively; to synthesize the sequential circuit benchmarks, it is necessary to separate the combinational cones of logic that are placed between the registers and apply the chaining heuristic to each cone; this is left open for future work.

The Stratix-III ALM can be configured with logic functions having up to 7-inputs; any 6-input logic function can be mapped onto the ALM, along with a subset of 7-input logic functions. Table 1 shows the distribution of functions in terms of the number of inputs for the different benchmarks. The majority of the functions have 5 inputs; we have selected logic functions having at most 5 inputs for mapping onto the chains; on average, 85% of logic functions are chainable.

Table 1: LUT mapping and chaining heuristic statistics.

Benchmark	3-LUT	4-LUT	5-LUT	6-LUT	7-LUT	Chainable	Chained	Max Chain	Ave Chain
alu4	163	89	413	32	6	94%	39%	12	5.2
pdc	431	214	538	139	5	89%	53%	9	5.8
misex3	179	99	376	43	1	93%	42%	9	5.1
ex1010	166	148	336	419	1	60%	47%	8	5.3
ex5p	149	89	234	60	0	88%	46%	7	5.2
des	134	89	159	110	0	77%	20%	4	3.1
apex2	206	101	284	108	2	84%	39%	8	4.9
apex4	127	74	354	119	1	82%	59%	8	4.3
spla	258	229	719	114	2	91%	46%	11	5.3
seq	205	150	356	96	2	88%	43%	6	4.9
Average	201	128	376	124	2	85%	44%	8.2	4.9

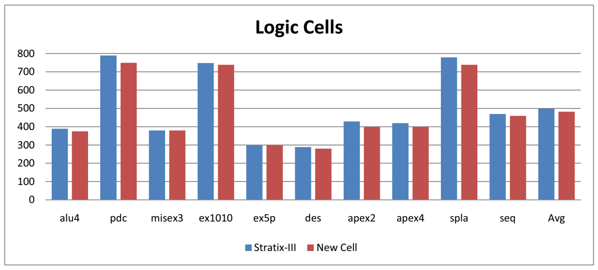

Figure 9: Number of logic cells (ALMs) that are used in each method. On average, the introduction of our logic chain reduces ALM usage by 4%.

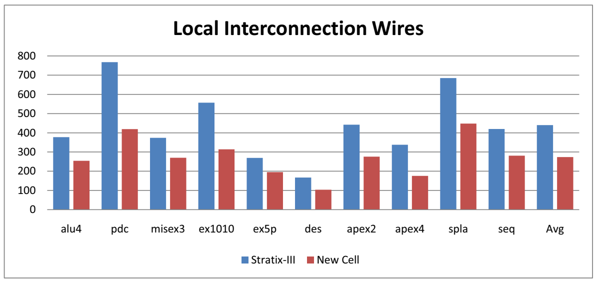

Figure 10: The number of local interconnection wires (i.e., within a LAB) used for each benchmark. On average, the introduction of the logic chain reduces the number of local wires used by 37%.

The last three columns in Table 1 summarize the chaining heuristic. The column labeled "Chained" reports the percentage of eligible functions that are placed onto the logic chain, which is 44%, on average; we set the minimum length chain to 4 for all benchmarks, except for *apex4* and *sec*, where we allowed minimum chain lengths of 3. The last two columns of the table report the maximum and average chain lengths for each benchmark.

Quartus-II is used to place-and-route each circuit. To evaluate the new logic cell, we compare against the Stratix-III FPGA as a baseline; as described earlier, we take a netlist that has been mapped onto the Stratix-III, identify logic chains and re-map them onto our new logic cell that includes vertical logic chains.

Figure 9 reports the number of logic cells that are used; our chaining heuristic was able to reduce the number of logic cells used for all benchmarks other than *ex5p* and *misex3*; on average, our approach uses 4% fewer logic cells than the Stratix-III. It is important to note that the Stratix-III ALM is used most effectively when it is configured to implement two 5-input logic functions with shared inputs, as shown in Figure 2 (a); in such cases, which are actually quite common, the introduction of our logic cell with the chaining heuristic cannot offer a significant improvement in terms of logic density.

The real benefit of using our logic block is its ability to reduce the usage of routing resources, as reported in Figures 10 and 11. On average, our logic cell and chaining heuristic reduces the usage of local wires by 37%, with a maximum savings of 45%. On average, we reduce the usage of global and local wires by 12%, with the local wires contributing a reduction in 3%. The minimum saving on total wiring was 7% (*apex2*) and the maximum was 22% (*seq*). To account for different horizontal and vertical wires with different lengths, we have scaled the wires based on their length and we reported their sum in Figure 11.

Reducing the usage of interconnect noticeably improves dynamic power consumption. A large fraction of dynamic power is consumed in the routing network; therefore, replacing a net that is routed on programmable interconnect with a direct connection using the logic chain can help to reduce dynamic power consumption. Figure 12 compares estimates of the dynamic power consumption of the Stratix-III to our proposed FPGA that includes logic chains. On average, we reduce dynamic power consumption in the routing network by 18%. The most dramatic improvement was observed for *seq*, which had the greatest savings in total interconnect as reported in Figure 11.

Figure 13 reports the dynamic power consumption for each benchmark, which includes power consumption of logic resources; on average, the introduction of the logic chain reduces dynamic power consumption by 10%. It is important here to note that the power estimation methodology used here is far from precise; however, our circuits are too large to use a much more accurate methodology such as SPICE simulation. Although we do not trust the exact numbers reported in Figures 12 and 13, we consider them a good indication of the type of savings that is possible using our logic cell.

Lastly, we measure the logic chain's impact on critical

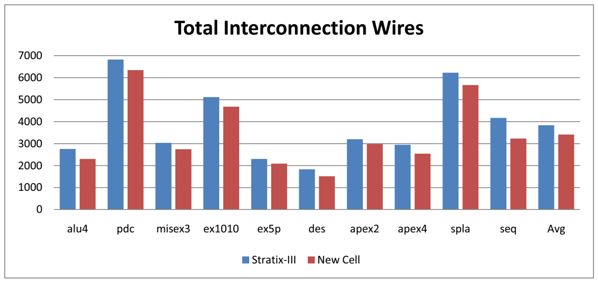

Figure 11: The number of global and local interconnection wires used for each benchmark, scaled by the length of the wires. On average, the introduction of the logic chain reduces the total number of wires used by 12%.

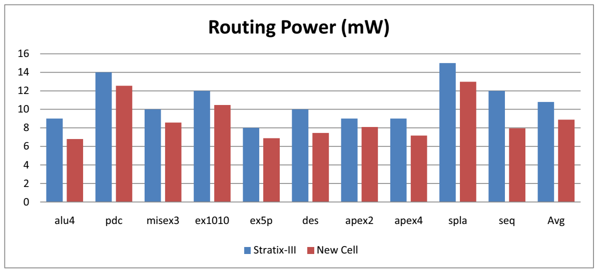

Figure 12: Dynamic power consumption estimates for the routing network; as the logic chain reduces the number of programmable wires used, an average savings of 24% is obtained.

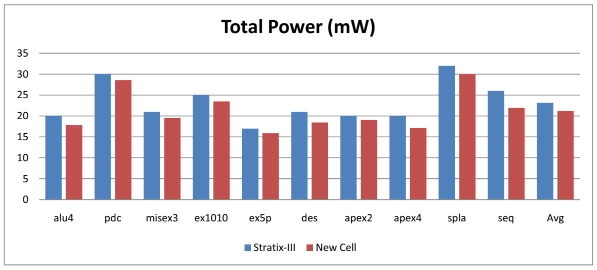

Figure 13: Total (logic plus routing network) power consumption estimates; the logic chain reduces total power consumption by 10%, on average.

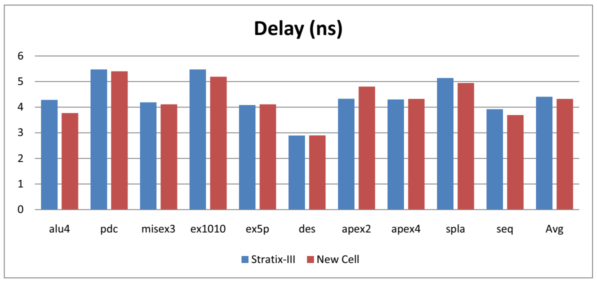

Figure 14: Critical path delay of each benchmark; the introduction of the logic chain marginally improves the critical path delay of most benchmarks.

path delay; an improvement is observed for all benchmarks other than *apex2* and *apex4*. The overall improvement in delay is minimal; however, the logic chain was introduced primarily to reduce interconnect and logic block usage, not to improve delay. We do believe that there is potential to further improve the critical path delay, which would require much more aggressive synthesis algorithms that are specific to the new logic chain. In particular, this would require a new logic decomposition algorithm that recognizes the cascaded structure of the logic chain; such an algorithm could be integrated with a technology mapper to make better use of the logic chains than the relatively naive and greedy chaining heuristic described here; this is an important research direction that is currently left open for future work.

7. CONCLUSION

This paper has introduced the concept of logic chains as a way to improve routing resource utilization in modern high-performance FPGAs. The dedicated wires between logic cells in the chain reduce pressure on the routing network. The key idea is based on the observation that many technology mapped circuits contain linear chains of LUTs after mapping; the basic idea is to add a direct connection between LUTs in a cluster that provide a natural mapping target for these chains. Having the fracturable structure of LUTs in modern FPGAs, we form larger LUTs by adding a multiplexer which is controlled by the direct output of a preceding LUT along the chain. This enables us to increase the input bandwidth and logic density of the logic cells without adding any additional inputs. Our experimental results have shown that the proposed logic cell with logic chains can reduce the total number of routing resources required by 12%. This reduction saves 10% of the total dynamic power consumption. Moreover, the number of logic cells used is reduced by 4% and the critical path delay is improved marginally as well.

In this paper, we simply searched for logic chain candidates in a netlist that was produced by a technology mapper that was unaware of the logic chains. We believe that more chains can be found and that pressure on the routing network can be reduced further through the development of logic decomposition and technology mapping algorithms that are aware of and can exploit the logic chains. We intend to pursue the development of these algorithms as future work.

8. REFERENCES

[1] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. In *http://www.eecs.berkeley.edu/˜alanmi/abc/*.

[2] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. In *IEEE Transactions on VLSI Systems*, volume 12, pages 288–298, 2004.

[3] Altera Inc. *PowerPlay Early Power Estimator User Guide*. Available online: http://www.altera.com/literature/ug/ug_epe.pdf.

[4] Altera Inc. *Stratix, Cyclone, Stratix II, III, and IV*

device handbooks. Available online: http://www.altera.com/.

[5] V. Betz and J. Rose. VPR: a new packing, placement, and routing tool for FPGA research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL)*, pages 213–222, Sept. 1999.

[6] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep Submicron FPGAs*. Kluwer Academic Publishers, Feb. 1999.

[7] G. Chen and J. Cong. Simultaneous logic decomposition with technology mapping in FPGA designs. In *Internation Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 48–55, Feb. 2001.

[8] D. Cherepacha and D. Lewis. DP-FPGA: an FPGA architecture optimized for datapaths. In *VLSI Design*, volume 4, pages 329–343, 1996.

[9] P. Chow, S. Seo, D. Au, B. Fallah, C. Li, and J.Rose. A 1.2um cmos fpga using cascaded logic blocks and segmented routing. In *Workshop on Field Programmable Logic and Applications*, pages 91–102, Sept. 1991.

[10] J. Cong and Y. Ding. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 13, pages 1–12, Jan. 1994.

[11] T. S. Czajkowski and S. D. Brown. Functionally linear decomposition and synthesis of logic circuits for FPGAs. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 12, pages 2236–2249, Dec. 2008.

[12] A. H. Farrahi and M. Sarrafzadeh. Complexity of the lookup-tale minimization problem for FPGA technology mapping. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 13, pages 1319–1332, Nov. 1994.

[13] A. H. Farrahi and M. Sarrafzadeh. FPGA technology mapping for power minimization. In *4th International Workshop on Field-Prog. Logic and Applications*, pages 66–67, Sept. 1994.

[14] M. T. Frederick and A. K. Somani. Multi-bit carry chains for high-performance reconfigurable fabrics. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 1–6, Aug. 2006.

[15] M. T. Frederick and A. K. Somani. Non-arithmetic carry chains for reconfigurable fabrics. In *International Conference Computer Design (ICCD)*, pages 137–143, Oct. 2007.

[16] M. T. Frederick and A. K. Somani. Beyond the arithmetic constraint: depth-optimal mapping of logic chains in LUT-based FPGAs. In *Internation Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 37–46, Feb. 2008.

[17] S. Hauck, M. M. Hosler, and T. W. Fry. High-performance carry chains for FPGAs. In *IEEE Transactions on VLSI Systems*, volume 2, pages 138–147, 2000.

[18] A. Kaviani, D. Vranseic, and S. Brown. Computational field programmable architecture. In *IEEE Custom Integrated Circuits*, pages 261–264, May 1998.

[19] K. Leijten-Nowak and J. L. van Meerbergen. An FPGA architecture with enhanced datapath functionality. In *Internation Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 195–204, Feb. 2003.

[20] H. Parandeh-Afshar, P. Brisk, and P. Ienne. A novel FPGA logic block for improved arithmetic performance. In *Internation Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 171–180, Feb. 2008.

[21] H. Parandeh-Afshar, P. Brisk, and P. Ienne. Exploiting fast carry-chains of FPGAs for designing compressor trees. In *Proceedings of the 19th International Conference on Field-Programmable Logic and Applications*, pages 242–49, Prague, Aug. 2009.

[22] Xilinx Inc. *Virtex-5 User Guide*. `http://www.xilinx.com/`.