Direct Mapping of RTL Structures onto LUT-Based FPGA's

A. R. Naseer, M. Balakrishnan, and Anshul Kumar

***Abstract*—The problem of mapping synthesized RTL structures onto look-up table (LUT)-based field programmable gate arrays (FPGA's) is addressed in this paper. The key distinctive feature of this work is a novel approach to perform the mapping by utilizing the iterative nature of the data path components. The approach exploits the regularity of data path components by slicing the components and mapping slices of one or more connected components together. This is in contrast to other FPGA mapping techniques which start from Boolean networks. Both cost optimal and delay optimal mappings are supported. The objective in cost optimal mapping is to cover a given data path network with minimum number of CLB's. Similarly in delay optimal mapping, the objective is to reduce the number of CLB levels in the critical combinational logic paths. Implementation of these mapping techniques with LUT based FPGA's as target technology results in a significant reduction in cost (CLB count) and critical path delays (CLB levels).**

***Index Terms*—Data path, FPGA, look-up table, RTL structure, technology mapping.**

I. Introduction

The field programmable gate arrays (FPGA's) provide a new approach to application specific integrated circuit (ASIC) implementation that features both large scale integration and user programmability. Short turnaround times and low manufacturing costs have made FPGA technology popular for rapid system prototyping and low to medium-volume production. The increase in complexity of FPGA's in recent years has made it possible to consider implementation of data paths on FPGA's.

Most of the technology mapping approaches that have been reported for look-up table (LUT)-based FPGA's, start from a Boolean network (consisting of basic gates such as AND, OR, etc.). A variety of optimization objectives have been addressed in these mappers:

- minimizing the number of LUT's required to map a circuit, e.g., Chortle-crf [1] and MIS-PGA [2] for combinational circuits, and SIS-FPGA [3] for sequential circuits;
- minimizing the number of LUT's in the critical path, e.g., Chortle-d [4] and flowmap [5], or minimizing both configurable logic blocks (CLB) levels and wirelength, e.g., MIS-PGA_delay [6] and TechMap-D [7];
- maximizing the routability of the mapping solutions, e.g., Rmap [8] and RFR [9].

As already mentioned, all these approaches start from a gate level structure and generate CLB network. In contrast to this, the vendor's proprietary tools such as XACT/XBLOX [10] offer the possibility of starting with a structural design consisting of macro cells. The tools incorporate module generators which can expand and map these macro cells to LUT's.

Manuscript received May 13, 1996. This work was supported by the Department of Electronics (DOE), Government of India, under the IDEAS project and AICTE, Ministry of HRD, Government of India, under the QIP Program. This paper was recommended by Associate Editor A. Saldanha.

A. R. Naseer is with the Department of Computer Engingeering, KREC, Surathkal, Karnataka, India 574157.

M. Balakrishnan and A. Kumar are with the Department of Computer Science and Engineering, I.I.T. Delhi, New Delhi, India 110016 (e-mail: mbala@cse.iitd.ernet.im).

Publisher Item Identifier S 0278-0070(98)05198-7.

In this paper, we present a different approach which directly realizes an RTL data path in terms of FPGA's with the objective of minimizing cost or delay. The approach exploits the regularity of data path components. It involves dynamically slicing the components and considering slices of one or more connected components together for mapping. The main objective of this work is to integrate high-level synthesis with FPGA technology mapping. Our approach is fundamentally different from the previous approaches in the sense that we neither expand the RTL network into a Boolean network nor use module generators. In this process we retain the flexibility and generality of the approaches which start from Boolean networks while producing results that are comparable or superior to the specialized module generators. We have named this approach as FAST (an acronym for *FPGA targeted RTL Structure synthesis Technique*). FAST forms a backend to a Data path Synthesizer [11] and is integrated into IDEAS [12].

The rest of the paper is organized in seven sections. Section II presents the preliminary definitions and terms used in this paper. Cost and delay models used in the mapping are discussed in Sections III and IV. Algorithms for both cost optimal and delay optimal mapping of RTL structures onto FPGA's are presented in Section V. Examples illustrating the techniques used are described in Section VI. Results of technology mapping on XILINX devices for some high level synthesis benchmarks and conclusions are presented in Sections VII and VIII, respectively.

II. Definitions and Terminology

The input network is a data path RTL structure obtained from a high level synthesizer. This network is represented as a directed graph $G(V, E)$ where each node represents a *module* which could be either a register, a functional unit (such as ALU, Adder) or an interconnection element (such as MUX) and directed edges represent connections between the modules.

A *cell* is an indivisible part of a module that is iterated to form a *module*. While mapping a module to CLB's, it is divided into *slices*, where each *slice* is an array of contiguous cells of that *module*. A *cone* is a set of slices of interconnected nodes which lie on paths converging on a particular node called *apex* of the cone. It is often possible to map such sets of slices forming a cone onto a single CLB. A *realizable* cone is one that fits in a CLB. A cone is said to be *simple* or *compound* depending on whether it contains a single slice or multiple slices (of different nodes). Note that all slices are $simple$ form of cones.

Let $w(n)$ and $w(s)$ represent width (i.e., the number of cells) of a node n and slice s, respectively. In our approach, the slice width is not decided *a priori*, rather it is dynamically determined during the mapping process. We refer to this as *soft-slicing*. Let the maximum slice width of a node n which is a realizable cone by itself, be denoted by $msw(n)$.

Let n_k represent slice of node n with width k, then

$$msw(n) = \max\ k | n_k \text{ is realizable in a CLB.} \tag{1}$$

During the technology mapping process, we need to examine the data paths originating at primary inputs or register outputs and ending at primary outputs or register inputs. We call these paths d_paths. To facilitate examination of d_paths, we split each register node r of G into two nodes ro and ri, where

- ro represents a register output node which carries with it all the outgoing arcs of r;
- ri represents a register input node which carries with it all the incoming arcs of r.

Now every directed path from a source node to a sink node is a d_path.

III. Cost Model

Let $ns(s)$ denote the corresponding node for a slice s.

Minimum number of instances of slice s required to cover $ns(s)$ is given by

$$cnt(s) = \left\lceil \frac{w(ns(s))}{w(s)} \right\rceil. \tag{2}$$

Minimum number of CLB's required to realize a node n is given by

$$\text{CLB\_}cnt(n) = \left\lceil \frac{w(n)}{msw(n)} \right\rceil = cnt(n_k) \tag{3}$$

where k is maximum slice width and n_k denotes slice of n of width k.

As our intention is to minimize the number of CLB's required to realize the graph, we start with an upper bound on CLB's required. This can be easily found by adding the minimum number of CLB's required for realizing each of the nodes in the node set V

$$\text{CLB\_}upper\_bound = \sum_{n \in V} \text{CLB\_}cnt(n). \tag{4}$$

Our algorithm is based on packing slices from multiple nodes into a single CLB. This is achieved by identifying *compound* cones. Further, we consider only those cones which are *realizable* and *beneficial* i.e., those which reduce the number or the levels of CLB's required.

Let c be a realizable compound cone consisting of slices of nodes which form a set denoted by $V(c)$. Let $CA(c)$ denote the cost of realizing the nodes of cone c individually, i.e., using *simple* cones. It can be computed by simply summing the CLB_cnt of the individual nodes that make up the cone c

$$CA(c) = \sum_{n \in V(c)} \text{CLB\_}cnt(n). \tag{5}$$

The slices in c can possibly be of different widths and have different cnt values. Therefore, in general, the number of instances of cone c is given by the minimum cnt of its slices. That is

$$\min\_cnt(c) = \min_{s \in c} cnt(s). \tag{6}$$

Let $CB(c)$ denote the cost of realizing the nodes in $V(c)$, with *compound* cone c formed. As each cone is realized by a CLB, $\min\_cnt(c)$ gives the number of CLB's realizing cone of type c. Due to differences in bit width of nodes as well as slice width of slices in c, the nodes may not be covered completely by *compound* cones. The remaining part of nodes are covered by the maximum width slices.

The width of that part of $ns(s)$ which is not covered by the *compound* cone c is denoted as $uncvr\_w(ns(s))$ and is given by

$$uncvr\_w(ns(s)) = w(ns(s)) - \min\_cnt(c) * w(s). \tag{7}$$

Now, the cost of realizing that part of $ns(s)$ which is covered by slices (*simple* cones) of width $msw(ns(s))$ is denoted by $cvr\_slices(ns(s))$ and can be given by

$$cvr\_slices(ns(s)) = \left\lceil \frac{uncvr\_w(ns(s))}{msw(ns(s))} \right\rceil. \tag{8}$$

Therefore, $CB(c)$ is given by

$$CB(c) = \min\_cnt(c) + \sum_{s \in c} cvr\_\text{slices}(ns(s)). \tag{9}$$

Now we can quantify the gain due to *compound* cone c as the difference between these two costs:

$$cost\_\text{gain}(c) = CA(c) - CB(c). \tag{10}$$

We define a set of cones C as complete if it covers all the nodes in the graph. In this formulation, we consider only those cone sets in which *compound* cones do not overlap.

Cost of a cone set C denoted by CB_tot is

$$CB\_tot(C) = \sum_{c \in C} CB(c). \tag{11}$$

IV. Delay Model

To analyze delay of a node in graph G, we consider the network of cells corresponding to the RTL component represented by the node n. We define cell_levels(n) to be the number of cells in the critical path in this network. When the RTL components are connected to form a data path, we need to take into account the direction of signal propagation within the components. For example, signal propagation through the cells is from LSB to MSB in case of adders, from MSB to LSB in case of some comparators and no horizontal chaining of cells in multiplexers.

To facilitate the computation of delays of the d_path containing different types of nodes, we split the delay of a node n into two parts: vertical delay $vd(n)$ representing the delay of a single cell and horizontal delay $hd(n)$ representing the additional delay encountered due to signal propagation within the node as follows:

$$vd(n) = D_{\text{cell}}(n) \tag{12}$$

$$hd(n) = \left(\left\lceil \frac{|\text{cell\_levels}(n)|}{msw(n)} \right\rceil - 1 \right) \cdot D_{\text{cell}}(n) \tag{13}$$

where $D_{\text{cell}}(n)$ is the cell delay of node n which is an integer multiple of D_{CLB},[1] the delay of a CLB which is nothing but the delay of the function generators or LUT's. It is assumed that the delay from any input to any output in a CLB is identical and is denoted by D_{CLB}. The two parts of the node delay combine differently when path delays are computed.

The minimum delay of dp, denoted by $\min\_\text{del}(dp)$, is obtained by adding the horizontal and vertical parts of the minimum delays of nodes in $V(dp)$ appropriately. All the vertical delays are added unconditionally, whereas the horizontal delays are added conditionally, depending upon the directions of signal propagation in the adjoining nodes. Fig. 1 shows an example in which the horizontal delays of only nodes 1, 3, and 4 contribute to the path delay

$$\min\_\text{del}(dp) = \sum_{n \in V(dp)} (vd(n) + hd(n) * p(n, dp)) + \alpha(dp) \tag{14}$$

where $\alpha(dp)$ represents the sum of the source delay (either the input pad delay or register propagation delay) and the sink delay (either the output pad delay or register setup time). $p(n, dp)$ is a 0-1 flag which determines whether the horizontal delay of a node n is to be included in the d_path delay computation.

A d_path is critical if it has the largest $\min$_del among all the d_paths. The critical path delay of a graph G is given by

$$\text{critical\_del}(G) = \max_{dp \in DP} \min\_\text{del}(dp) \tag{15}$$

where DP is the set of all d_paths. Let $V(C)$ denote the set of nodes whose slices are included in a *compound* cone C. Due to differences in widths of slices in C, some of the nodes in $V(C)$ may not be completely covered by that cone. The uncovered parts of

[1]This is because in some of the components each basic cell may occupy more than one CLB.

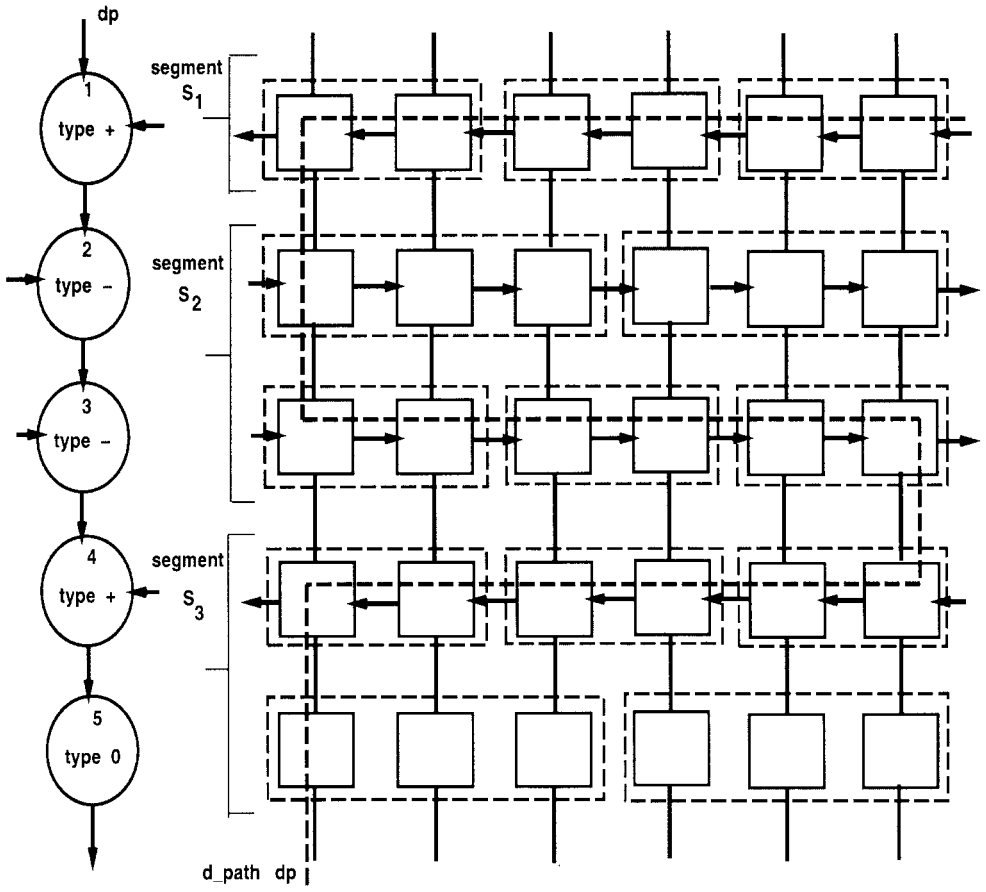

Fig. 1. (a) A portion of a 6-bit data path. (b) Its cell level structure.

partially covered nodes are considered to be covered by maximum width slices, forming *simple* cones, as described in context of the cost model [refer to (7) and (8)].

Let dp/C denote the portion of the path dp intersected by *compound* cone C [i.e., $V(dp/c) = V(dp) \cap V(c)$].

Let us denote by del(dp, CS) the delay of the path dp after selection of cone set CS. Then delay of a d_path dp before selecting any *compound* cone is given by

$$\text{del}(dp, \phi) = \min\_\text{del}(dp). \tag{16}$$

Let del_gain(C, dp, CS) denote the gain in terms of delay reduction due to covering of path dp/C by *compound* cone C. This can be used to update the delay of the path after cone formation as follows:

$$\text{del}(dp, CS + C) = \text{del}(dp, CS) - \text{del\_gain}(C, dp, CS) \tag{17}$$

del_gain(C, dp, CS) can be expressed in terms of changes in vertical and horizontal delays along dp due to cone C

$$\text{del\_gain}(C, dp, CS) = \sum_{n \in V(dp/C)} \Delta d(n, C, CS, dp) - D_{\text{CLB}} \tag{18}$$

$$\Delta d(n, C, CS, dp) = \Delta vd(n, C, CS) + \Delta hd(n, C, CS) \cdot p(n, dp) \tag{19}$$

where Δvd is the change in vertical delay and Δhd is the change in horizontal delay in the path dp due to covering by *compound* cone C.

The cones are selected on the basis of their potential gain, given by

$$\text{cone\_gain}(C) = \max_{dp \in \text{critical paths}} \text{del\_gain}(C, dp, \phi). \tag{20}$$

V. Fast Mapping Algorithms

A. Cost Optimal Mapping Algorithm

The algorithm described in Fig. 2 shows the major steps involved in cost optimal mapping [13] of RTL structure onto FPGA's. Step 1 computes the CLB upper bound and Step 2 traverses the network and generates cones. We traverse the network backward starting from sink nodes and generate *realizable* cones with nonnegative cost_gain by considering various soft slicing options and merging them till no more merger is feasible or source nodes are reached. The feasibility of these cones are checked as they are generated and only *realizable* ones are retained. This is described in detail in Section V-C. Step 3 finds a cover which minimizes the CLB count. In the present implementation we have used a greedy approach for covering the nodes with the cones.

B. Delay Optimal Mapping Algorithm

The major steps of the algorithm for optimal delay mapping [14] of RTL structures onto FPGA's are described in Fig. 3. In Step 1, for each node in the graph, we compute the $\min$_del i.e., the number of

```
Input  :  RTL Data path structure
Output :  Cost optimal interconnected CLB map
```

1. Computation of CLB upper bound
- 1.1 *for* each node $n \in$ node set V of graph G
  - 1.1.1 compute (i) $msw(n)$ and (ii) $CLB\_cnt(n)$
- 1.2 compute CLB_upper_bound

2. Realizable Cone generation
- 2.1 *for* all *realizable* slices s of nodes in V *do*
  - 2.1.1 generate all candidate cones with s as *apex* using *softslicing*
  - 2.1.2 identify *realizable* cones by decomposition
  - 2.1.3 compute *cost_gain* (eqn. 10)
  - 2.1.4 Select *realizable* cones with *cost_gain* ≥ 0

3. Cost Optimal cone cover
- 3.1 Generate complete cone sets with minimum cost

Fig. 2. Algorithm for cost optimal mapping of RTL structures onto FPGA's.

```
Input  :  RTL Data path structure
Output :  Delay optimal interconnected CLB map
```

1. Computation of Delay upper bound
- 1.1 *for* each node $n \in$ node set V of graph G
  - 1.1.1 compute (i) $msw(n)$ (ii) $no\_of\_levels(n)$ and (iii) $min\_del(n)$
- 1.2 compute d_path delays
- 1.3 identify the critical paths

2. Realizable Cone generation
- 2.1 *for* all *realizable* slices s of nodes in V *do*
  - 2.1.1 generate all candidate cones with s as *apex* using *soft_slicing*
  - 2.1.2 identify *realizable* cones by decomposition
  - 2.1.3 compute *cone_gain* (eqn. 20)
  - 2.1.4 Select *realizable* cones with *cone_gain* ≥ 0

3. Delay optimal cone cover
- 3.1 Generate complete cone sets which covers the entire network G with minimum delay

Fig. 3. Algorithm for delay optimal mapping of RTL structures onto FPGA's.

CLB levels required to realize slices in the critical path of the nodes. Next we determine the d_path delays and select set of nodes in the critical paths. Step 2 generates and identifies realizable and beneficial cones similar to algorithm FAST_CMAP using (20) to compute delay gain of a cone. In Step 3, a cover is found which minimizes the CLB levels or delay in the critical path.

C. Realizable Cone Generation

The *realizable cone generation* algorithm is shown in Fig. 4. This algorithm considers each node from the nodes of G and generates variable-width slices of width varying from one to maximum slice width and checks whether each slice of that node can be merged with a slice of the node at its fanin to form a cone. If the resulting cone is *realizable* and *beneficial*, then starting with this newly generated cone, it further checks whether it can be merged with slices of the node at its fanin. This process is repeated until no more slices can be packed into the cone. For each cone generated, it computes the reduction in CLB count in case of cost optimal mapping and reduction

1. $Cone\_set = \phi$
2. *for* all $n \in V$ *and* $i = 1$ to $msw(n)$ *do*
   - 2.1 $C = \{ n_i \}$
   - 2.2 $Cone\_set = Cone\_set + C$
   - 2.3 $grow\_cone(C)$

procedure grow_cone(C)

1. *for* all $u \in fanin\_set(C)$ excluding register o/p's and primary i/p's *and* $j = 1$ to $msw(u)$ do
   - 1.1 $C^1 = merge(C, u_j)$
   - 1.2 *if* C^1 is *realizable* and *beneficial then*
     - 1.2.1 $Cone\_set = Cone\_set + C^1$
     - 1.2.2 $grow\_cone(C^1)$

Fig. 4. Algorithm for realizable cone generation.

in CLB levels (delay) in case of delay optimal mapping and rejects those for which no gain occurs.

During cone generation an important check to be performed is whether a cone is realizable or not. A CLB is characterized by a fixed number of inputs, outputs, and flipflops. Every *realizable* cone should have number of inputs, outputs and flipflops less than or equal to those present in a CLB. But for realizability this check is not sufficient because a CLB (unlike an LUT) cannot realize any arbitrary function of all of its inputs. Therefore, the boolean function of a cone may have to be decomposed into two or three parts (depending on the internal structure of the CLB) to be mappable onto a CLB.

Among the decomposition techniques employed by FPGA mapping systems, Roth–Karp [15] is the most versatile but suffers from high computation complexity. The complexity arises due to the fact that all possible combinations of variables have to be exhaustively checked for decomposition. We have developed heuristics for fast decomposition [16], which is based on checking some simple necessary conditions before checking the sufficiency conditions for *feasible* decomposition. Thus during the decomposition process a large number of candidate solutions are quickly rejected to achieve a speedup.

Apart from functional decomposition, we also explore splitting of a multi-output node into multiple nodes, each with a single output. This sometimes results in packing of a larger slice in a CLB.

D. Cost Optimal Cone Cover

A greedy approach is being followed at present for finding a cone cover. The procedure actually involves generating several cover sets of cones and finally retaining the best one. It begins by sorting the list of cones in the decreasing order of the gain. Initially a cover set containing the first cone of the cone_list is formed and the CLB_upper_bound is taken as the optimal cost for covering the entire network. Next each cone other than the first cone is taken from the cone_list and checked to see whether it overlaps with the cover sets already generated. If the cone overlaps with all the cover sets already generated, it creates a new cover set with this cone. Otherwise, it is added to all the nonoverlapping cover sets and optimal cost of realizing the network is made equal to the minimum of CLB_upper_bound and cost of newly formed complete cover sets. All cover sets exceeding this optimal cost are rejected.

E. Delay Optimal Cone Cover

Presently a greedy approach is being followed for delay cone cover as well. We start by choosing a cone from the set of *beneficial* cones generated in Step 2 (Fig. 3), which reduces the longest path delay

by maximum value while covering uncovered or partially covered nodes. We update delays on all the paths which contain the nodes forming this selected cone. Next, we determine the new critical path in the updated graph. This process of choosing a *delay beneficial* cone, updating the path delays and determining the new critical path is repeated until all the nodes in the graph are completely covered.

VI. Illustrative Examples

A. Cost Optimal Mapping Example

We illustrate the cost optimal mapping technique using a RTL structure obtained from IDEAS Data Part Synthesizer [11] which takes a behavioral description of greatest common divisor (GCD) high-level synthesis benchmark as input. Fig. 5(a) shows the GCD RTL structure, and Fig. 5(b) gives the CLB map of this structure for XILINX XC3000.[2] In Fig. 5(b) the dotted rectangles enclosing the nodes indicate that they can be realized using single CLB's and the number in the small square box associated with each node indicates the width of the slice of that node packed in a CLB.

We traverse the GCD network starting from a register node *rega* and generate realizable cones by merging *rega* with slices of nodes at its fanin, i.e., *muxa*. Table I shows the realizable cones rooted at *rega*, slices of nodes associated with these cones and CLB count.

It is evident from the table that cone C_{22} is most beneficial as it requires only eight CLB's, whereas C_{11} and C_{22} consume 16 and 12 CLB's, respectively, and hence the latter are rejected.

Starting with this newly generated cone C_{22}, we further check whether it can be merged with the slices of the nodes at its fanin. Since no further merger is possible, this cone is added to the cone list. As it can be seen from the figure, a one-bit slice of the *alu* node has four inputs and two outputs and it cannot be merged with any other node and hence it forms a separate cone. Similarly, traversing the network from *regb* toward the primary inputs generates the next beneficial cone containing two-bit slices of *regb* and *muxb*. Next the traversal is continued from primary outputs toward register inputs. The comparator node *cmp* has five inputs and three outputs and cannot be realized by a CLB, and hence it is decomposed into three subnodes, one-bit slices of first two nodes occupy a single CLB whereas two-bit slices of the third node get mapped onto one CLB.

B. Delay Optimal Mapping Example

For illustration purpose, we consider a portion of the critical path dp_1 node set from AR-filter data path example comprising three nodes, node A (16-bit Adder), node B (two-input 16-bit wide MUX), and node C (16-bit register) as shown in Fig. 6(a). Node A has 16 one-bit slices in its critical path, whereas nodes B and C have single one-bit slice in their respective critical paths. In the case of mapping of this part of the structure onto XC3000 type CLB's, one-bit slice of node A requires a CLB whereas two-bit slices of nodes B and C each require a CLB, if considered individually. Or in other words, we can say that cell_levels$(A) = 16$, cell_levels$(B) = 1$ and cell_levels$(C) = 1$. Therefore, $vd(A) = 1, hd(A) = 15; vd(B) = 1, hd(B) = 0, vd(C) = 1$ and $hd(C) = 0$. Now, the delay of the path dp_1 is given by $\min\_\text{del}(dp_1) = 3 + 15 = 18 * D_{\text{CLB}}$. As node A is a multi-output node which cannot be combined with any other nodes at its fanout, it is split into sum $(A1)$ and carry generator $(A2)$ nodes as shown in Fig. 6(b). At node $A1$, after cone generation, we have three choices of cones A_{c_1}, A_{c_2} and A_{c_3} as shown in Fig. 6(c). For cone A_{c_1}, cone_gain$(A_{c_1}) = 0$ and del_gain$(A_{c_1}, dp_1) = 0$; for cone A_{c_2}, cone_gain$(A_{c_2}) = 1$ and del_gain$(A_{c_2}, dp_1) = 1$; whereas for cone A_{c_3}, cone_gain$(A_{c_3}) = 2$ and del_gain$(A_{c_3}, dp_1) = 2$. From the values of delay gains for cones at node $A1$, it is clear that cone A_{c_3} is the most beneficial cone as it reduces the critical path delay from 18 to 16 CLB levels. The number of CLB's required to cover these nodes in this case is 32, whereas cost optimal mapping on this

[2] A CLB in XC3000 has five inputs and two outputs.

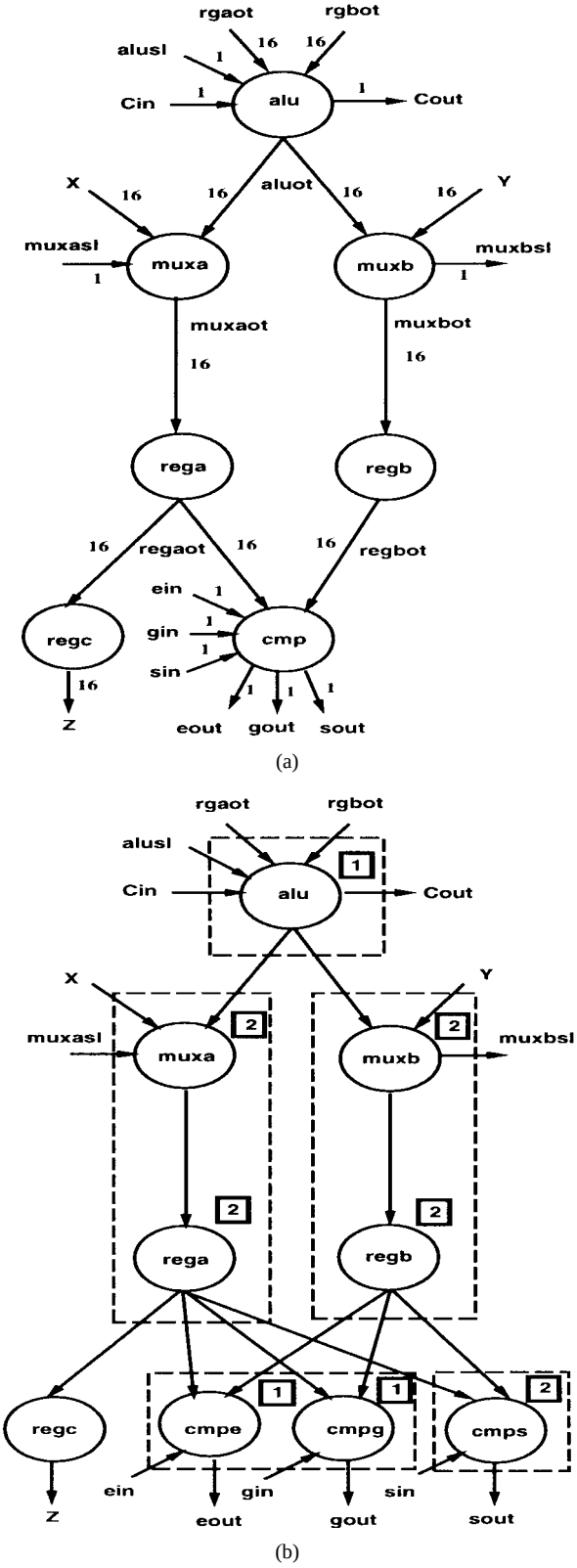

Fig. 5. RTL structure and CLB map for GCD example.

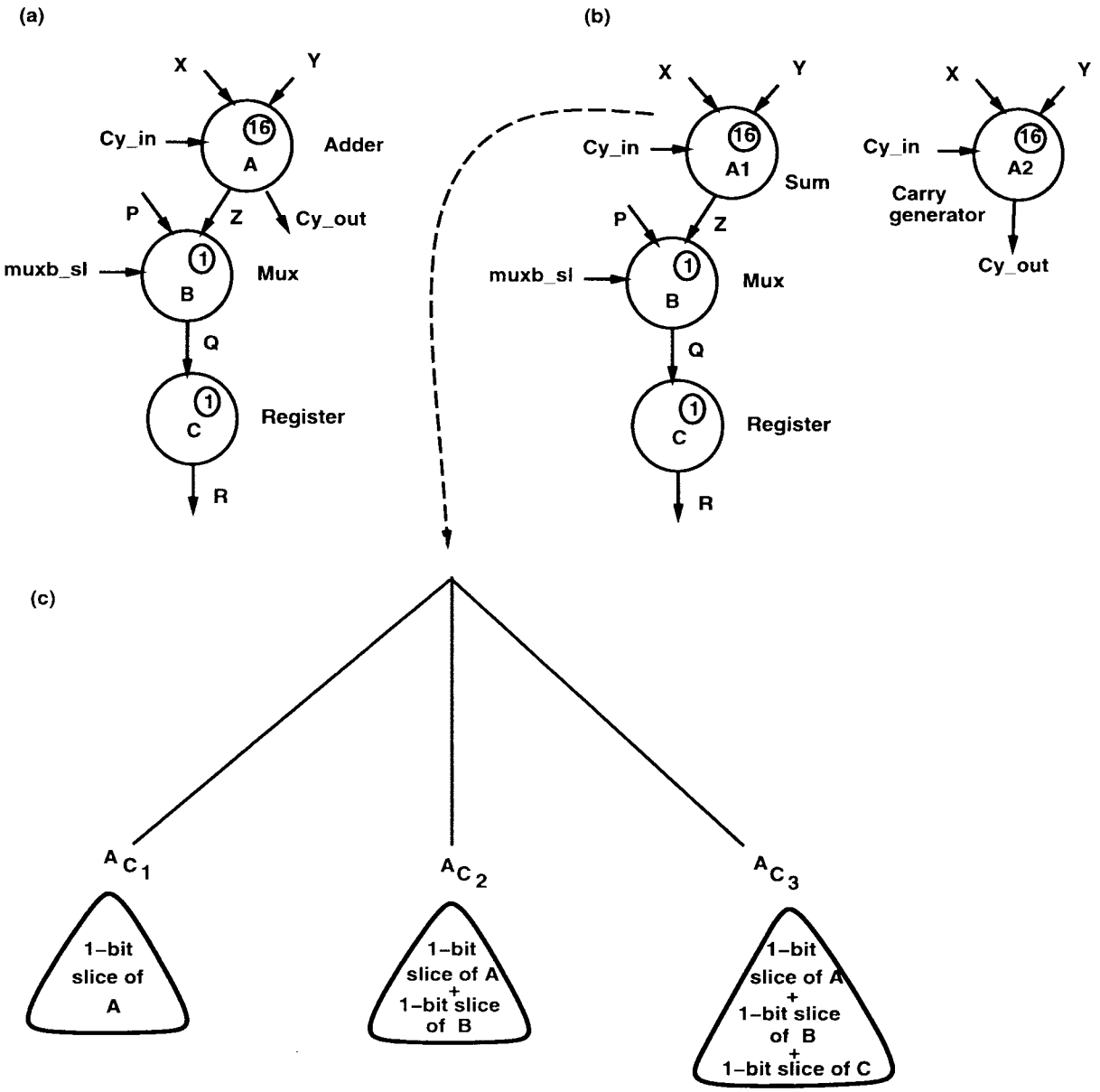

Fig. 6. (a) A critical path node set. (b) Cone choices at node A.

TABLE I
PARTIAL CONE LIST

Cone	slice sizes	CLB Count
C_{11}	rega: 1-bit + muxa: 1-bit	16
C_{21}	rega: 2-bit + muxa: 1-bit	12
C_{22}	rega: 2-bit + muxa: 2-bit	8

structure would have resulted in a CLB count of 24 CLB's with path delay of 17 CLB levels.

VII. RESULTS AND DISCUSSION

The programs for cost optimal (FAST_CMAP) and delay optimal (FAST_DMAP) mappings have been implemented on SUN workstation. Results for the mapping of several RTL structures corresponding to high-level synthesis benchmarks [17] onto XC2000, XC3000, and XC4000 devices using these programs and comparison with XILINX proprietary tools are reported in Tables II–IV. For benchmarks containing multipliers, we have assumed that **multipliers are external to the design and are realized separately**. XC4000 device CLB's contain internal carry logic apart from the LUT's and flip-flops. We have accounted for these features in our CLB model in order to compare our results with XACT and XBLOX.

A. Cost Optimal Mapping Results

Results from FAST_CMAP are shown in Table II which lists the total number of CLB's required by FAST, XACT,[3] and XBLOX[4] for realizing the network after the mapping process. Table II also indicates the complexity of the datapath as it shows the set of RTL components in each case. The cost optimal mapping technique results in a CLB count reduction of upto 16.7% over XACT and upto 11.8% over XBLOX.

[3] XACT is a proprietary product of XILINX and interfaces with a schematic capture tool for mapping onto XILINX devices.

[4] XBLOX is a product from XILINX and supports MSI level module library.

TABLE II
RESULTS OF COST OPTIMAL MAPPING

HLS Benchmark		GCD	Diff_eqn	AR_filter	Elliptic	Tseng
Allocation		1 ALU 1 Comp 3 Reg 2 Mux	1 ALU 6 Reg 8 Mux	2 ALU 5 Reg 8 Mux	1 ALU 12 Reg 11 Mux	2 ALU 2 Comp 17 Reg 18 Mux
FPGA	Mapper	Total # of CLBs for HLS Benchmarks				
XC2000	FAST	96	320	240	656	630
	XACT	104	338	264	688	662
	% Reduc.	7.7	5.3	9.1	4.7	4.8
XC3000	FAST	56	160	120	328	317
	XACT	62	168	126	344	328
	% Reduc.	9.7	4.8	4.8	4.7	3.6
XC4000	FAST	40	120	104	256	294
	XACT	48	142	113	278	316
	% Reduc.	16.7	15.5	8.0	7.9	7.0
	XBLOX	40	136	104	272	304
	% Reduc.	0.0	11.8	0	5.9	3.3

TABLE III
RESULTS OF DELAY OPTIMAL MAPPING: SIMPLE VERSUS COMPOUND MAPPING

Bench-mark example	Allocation	FPGA device family	CLB Count		Critical Path Delay	
			simple Map	*comp.* Map	*simple* Map	*comp.* Map
GCD	$\{1alu, 1cmp\}$	XC2000	160	80	18	17
		XC3000	80	56	18	16
		XC4000	72	48	10	9
Diff_eqn	$\{1+, 1<, 2*, 1-\}$	XC2000	468	324	19	18
		XC3000	282	210	19	17
		XC4000	234	178	11	9
	$\{1+, 1<, 1*, 1-\}$	XC2000	432	320	22	20
		XC3000	280	216	22	19
		XC4000	200	152	14	11
Elliptic Filter	$\{3+, 2*\}$	XC2000	784	576	22	20
		XC3000	504	360	22	19
		XC4000	352	280	14	11
	$\{3+, 1*\}$	XC2000	688	544	22	21
		XC3000	440	328	22	20
		XC4000	320	272	14	12
	$\{2+, 1*\}$	XC2000	688	496	23	20
		XC3000	424	336	23	19
		XC4000	296	232	15	11
AR_filter	$\{2+, 4*\}$	XC2000	752	512	21	20
		XC3000	480	336	21	18
		XC4000	320	256	13	10
	$\{1+, 2*\}$	XC2000	672	448	22	20
		XC3000	400	280	22	18
		XC4000	296	228	14	11
	$\{1+, 1*\}$	XC2000	544	368	21	20
		XC3000	312	224	21	18
		XC4000	256	176	13	9

TABLE IV
RESULTS OF DELAY OPTIMAL MAPPING: FAST VERSUS XBLOX MAPPING

Benchmark example	Allocation	CLB Count		CLB Levels	
		FAST	XBLOX	FAST	XBLOX
GCD	$\{1alu, 1cmp\}$	48	39	9	12
Diff_eqn	$\{1+, 1<, 2*, 1-\}$	178	174	9	11
	$\{1+, 1<, 1*, 1-\}$	152	178	11	13
Elliptic Filter	$\{3+, 2*\}$	280	334	11	14
	$\{3+, 1*\}$	272	292	12	13
	$\{2+, 1*\}$	232	272	11	13
AR_filter	$\{2+, 4*\}$	256	298	10	11
	$\{1+, 2*\}$	228	264	11	12
	$\{1+, 1*\}$	176	220	9	10
Average % Reduc.		12		14.7	

B. Delay Optimal Mapping Results

Delay optimal mapping of four RTL structures synthesized with different operator allocations have been performed and results are reported in Table III. The allocation used in each example is listed in the second column of the table. The CLB count and critical path delays obtained using *simple* and *compound* mappings are given in the last four columns of the table. It is evident from the table that the delay optimal mapping technique using compound cones results in a substantial reduction in CLB count as well as in the CLB levels in the critical path as compared to mapping with simple cones. These structures have also been mapped using XBLOX to XC4000 devices and results are compared in Table IV. Our techniques result in an average reduction of 12.0% in CLB count and 14.7% in critical path CLB levels.

VIII. CONCLUSION

To conclude, we have presented approaches for cost optimal and delay optimal mapping of RTL structures onto FPGA's. To the best of our knowledge, this is the first attempt to map RTL structures directly onto FPGA's. The techniques are primarily meant for realizing data path and effectively utilize iterative structure of the data path components. The slices of connected components are generated and are called cones which are mapped onto CLB's. The approach is flexible and can handle various families of XILINX FPGA's. Results on many examples show its effectiveness in both cost and delay reduction. Further, the synthesized CLB boundaries correspond to RTL component boundaries which can be helpful in handling testability and ease delay simulation.

Our technique handles mapping starting from RTL structures and does not require expanding it to the gate level. As a result the data size required to be handled is relatively small. A benefit of this is that our algorithms can easily generate good quality solutions. The delay modeling is restricted to CLB levels and does not take into account either the interconnection delays or the variations in delays between various CLB input–output pairs.

ACKNOWLEDGMENT

The authors are thankful to T. V. Kumar for his assistance in comparing their approach with XBLOX.

REFERENCES

[1] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for look-up table-based FPGAs," in *Proc. 28th DAC*, June 1991, pp. 227–233.
[2] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate arrays," in *Proc. 27th DAC*, June 1990, pp. 620–625.
[3] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential synthesis for table look up programmable gate arrays," in *Proc. 30th DAC*, June 1993, pp. 224–229.
[4] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of lookup table-based FPGA's for performance," in *Proc. ICCAD'91*, Nov. 1991, pp. 568–571.
[5] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table-based FPGA designs," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1–12, Jan. 1994.
[6] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance directed synthesis for table look up programmable gate arrays," *in Proc. ICCAD'91*, Nov. 1991, pp. 572–575.
[7] P. Sawkar and D. Thomas, "Performance directed technology mapping for look-up table-based FPGA's," in *Proc. 30th DAC*, June 1993, pp. 202–212.
[8] M. Schlag, J. Kong, and P. K. Chan, "Routability-driven technology mapping for lookup-table-based FPGA's," in *Proc. ICCD*, Oct. 1992, pp. 86–90.

[9] M. Pedram and N. Bhat, "Layout driven technology mapping," in *Proc. 28th DAC*, June 1991, pp. 99–105.
[10] *Xilinx Programmable Gate Array Users' Guide*. Xilinx, Inc., 1993.
[11] M. V. Rao, M. Balakrishnan, and A. Kumar, "DESSERT: Design space exploration of RT level components," in *Proc. IEEE/ACM 6th Int. Conf. VLSI Design'93*, Jan. 1993, pp. 299–303.
[12] A. Kumar, V. Kashyap, S. D. Sherlekar, G. Venkatesh, S. Biswas, A. Kumar, P. C. P. Bhatt, and S. Kumar, "IDEAS: A tool for VLSI CAD," *IEEE Design and Test of Computers*, vol. 6, pp. 50–57, Oct. 1989.
[13] A. R. Naseer, M. Balakrishnan, and A. Kumar, "An efficient technique for mapping RTL structures onto FPGA's," in *Proc. 4th Int. Wkshp. Field Programmable Logic and Applicat.*, Sept. 1994, pp 99–110.
[14] ____, "Delay minimal mapping of RTL structures onto-LUT-based FPGA's," in *Proc. 5th Int. Wkshp. Field Programmable Logic and Applicat.*, Sept. 1995.
[15] P. J. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM J. Res. Development*, vol. 6, no. 2, pp. 227–238, Apr. 1962.
[16] A. R. Naseer, M. Balakrishnan, and A. Kumar, "FAST: FPGA targeted RTL structure synthesis technique," in *Proc. IEEE/ACM 7th Int. Conf. on VLSI Design'94*, pp. 21–24, Jan. 1994.
[17] E. Detjens, *Workshop on High-Level Synthesis*, Orcas Island, WA, Jan. 1988.

A Simple, Continuous, Analytical Charge/Capacitance Model for the Short-Channel MOSFET

Raymond S. Winton and William R. Bandy

***Abstract*—A charge/capacitance model of a simple form that is continuous across the linear and saturation regimes is developed. The model is based on a conductance analysis of the MOSFET which incorporates velocity saturation at a first-principles level [1]. By relating charge layers within the device to characteristics of the conductance, the charge model not only is able to characterize C–V behavior but to also incorporate velocity saturation. Since the basic conductance form is a hyperbola, the model is mathematically simple and robust and yields MOSFET capacitances and charges which are continuous and of infinite differentiability over the linear and saturation regimes of device operation.**

***Index Terms*—MOS charge/capacitance modeling, MOS device modeling.**

I. Introduction

Simulation of MOS very large scale integration (VLSI) circuits requires that the MOS device have a model for the charge/capacitance characteristics that makes use of parameters and physical basis consistent with the conduction model. This is often not an easy task when the conduction characteristics are not linearly related to charge, as in the case of the short-channel MOS transistor. In addition, conditions for robust circuit simulation demand: 1) mathematical simplicity, 2) accuracy, and 3) physical basis. This paper describes a new approach and a new MOSFET charge/capacitance model that meets these conditions, developed from a continuous hyperbolic conductance model form [1].

Manuscript received June 4, 1996. This paper was recommended by Associate Editor Z. Yu.

R. S. Winton is with the Department of Electrical Engineering, Mississippi State University, Mississippi State, MS 39762 USA.

W. R. Bandy is with the Microelectronics Research Laboratory, Columbia, MD 21044 USA.

Publisher Item Identifier S 0278-0070(98)05196-3.

The nonlinearity of the conductance behavior for the MOSFET is primarily due to the phenomenon of velocity saturation [2], [3]. Since the effect is dominant only for high fields, it is possible to make approximations in which the mobility is taken as constant, as is usually done for the sheet-charge models [4]–[9]. But the models strain to meet satisfactory mathematical behavior in the short-channel limits, often manifested by difficulty in linking the regimes above and below the saturation knee. In an alternative approach [1], a hyperbolic conduction form for the conductance characteristics has been assumed which models the complete conduction characteristics as a hyperbola, eliminating the gap between different regimes. This approach allows velocity saturation to be included in a fundamental, first-principles way. The effects may be extended into the charge-control domain by use of the relationship between channel conductance and the charge layers. The result is a set of charge and capacitance equations for the device that span both the linear and the saturation regimes in single equation form with velocity saturation implicitly included. The charge/capacitance model is of simple mathematical form, uniformly continuous across the linear-to-saturation transition knee, and analytically well behaved. Therefore, like the conduction model, it provides a good platform for modeling higher order behavior such as the indefinite admittance matrix [10]. Furthermore, a hierarchy of model equation complexity emerges, the lowest-order being of the same form as the traditional Ward–Meyer models [11]–[13], but with the effect of velocity saturation implicitly retained. As with the conductance model, both short- and long-channel behavior is included in a manner consistent with physical operation, as characterized by conventional device measurements and parameters.

The key feature of the charge/capacitance model is that, unlike traditional approaches to MOSFET modeling in which the conduction characteristics of the device are developed from a charge-control analysis, it reverses the process and develops the charge-control behavior from the conduction model. The advantage of this approach is that it yields a self-consistent method for including velocity saturation in both conduction and charge models of the MOSFET.

II. The Charge/Capacitance Model

At any point within the channel the charge controlled by the gate field consists of two components [14], [15]: 1) a channel inversion layer q_C and 2) a depletion layer q_B. For normal operating conditions, with the device in a conducting state, the density of these charge layers each varies monotonically from source to drain, with the magnitude of q_I decreasing and that of q_B increasing.

When the MOSFET is called in a circuit simulation process, the integral of charge controlled by each terminal is used to evaluate the device response. In the conducting state the integrals of the layers for q_C and q_B, respectively, are

$$Q_C = W \int_0^L q_C \, dy \tag{1a}$$

$$Q_B = W \int_0^L q_B \, dy. \tag{1b}$$

The total charge controlled by the gate, Q_G, is the sum of Q_C and Q_B.

A. Inversion Charge

The source and drain terminals independently act under transient conditions to charge and discharge the conductive charge layer Q_C.