# PROP: A Recursive Paradigm for Area-Efficient and Performance Oriented Partitioning of Large FPGA Netlists

Roman Kužnar[1] Franc Brglez[2]

[1]Dept. of Electrical & Comp. Eng., University of Ljubljana, Tržaska 25, 61000 Ljubljana, Slovenia
[2]CBL, Dept. of Electrical & Comp. Eng. Box 7911, North Carolina State University, Raleigh, N.C. 27695, U.S.A.
*(WWW: http://www.cbl.ncsu.edu/www/)*

**Abstract** – *In this paper, we introduce a new recursive partitioning paradigm* PROP *which combines partitioning, replication, optimization, to be followed by another recursion of partitioning, etc. We measure the quality of partitions in terms of total device cost, logic and terminal utilization, and critical path delay. Traditionally, the minimum lower bound into which a given netlist can be partitioned is determined by disregarding the logic interconnect while distributing the logic nodes into a minimum number of devices.* PROP *paradigm challenges this assumption by demonstrating feasible partitions of some large netlists such that the number of device partitions is smaller than minimum lower bounds postulated initially.*

*Overall, we report consistent reductions in the total number of partitions for a wide range of combinational and sequential circuit benchmarks while, on the average, reducing critical path delay as well.*

## I. Introduction

Much of research on partitioning algorithms considers minimization of the cutset. However, minimizing this objective alone does not necessarily lead to a good solution of the partitioning problem into multiple FPGAs. Iterative algorithms derive feasible solution by exchanging modules between partitions [1], [2]. Some of these techniques utilize clustering [3] or cell replication [4], [5], [6] to improve the results. Other techniques also exist which effectively solve the multi-way partitioning problem, such as spectral methods [7], [8], Boolean programming [9], geometric embeddings [10] placement techniques [11] and others [12], [13], [14].

Approaches that focus on FPGA partitioning, reporting benchmark results, have been proposed in [15], [16]. Contributions that address performance oriented partitioning of FPGA are only recent [17], [18]. This work addresses and extends the *FPGA partitioning problem* as defined in [6], [19], [20]. We introduce a new recursive partitioning paradigm *PROP* by combining *partitioning, replication, optimization*, to be followed by another recursion of *partitioning*, etc. While it can be shown that either *replication* or *optimization* can improve the quality of partitions, it is the combination of processes that consistently achieves the best results to date.

Following problem formulation, the paper illustrates a trend of monotonically decreasing number of partitions that clearly challenge bounds attainable with traditional partitioning techniques. After highlights of the proposed algorithm *PROP*, the paper concludes with a comprehensive report of the benchmarking experiment: from results with a simple bipartitioning strategy to results that invoke all phases of *PROP*. All experiments have been extensively documented for ease of comparison and reproducibility [21].

## II. Problem Formulation

We extend the notation from [6], [19], [20].

Hypergraph. The unpartitioned circuit is a *directed* hypergraph $\mathcal{H}_0 = (\{\mathcal{X}_0; \mathcal{Y}_0\}, \mathcal{E}_0)$. There are two node types: interior nodes $\mathcal{X}_0$ and *board terminal nodes* $\mathcal{Y}_0$; $\mathcal{E}_0$ is the set of edges. We refer to $\mathcal{Y}_0$ as primary input nodes (PIs) and primary output nodes (POs). After partitioning, nodes in $\mathcal{Y}_0$ connect to *device terminal nodes*. Interior nodes are multi-output combinational logic nodes, register nodes, and fanout nodes.

Combinational nodes describe a logic function of arbitrary complexity and are multi-output directed acyclic hypergraphs. Typically, they represent cells in a library and are generated during technology-mapping. All register nodes are D-flipflops implicitly synchronized with a single clock, with a single data input designated as a pseudo-primary output (PPO), and a single data output designated as a pseudo-primary input (PPI). Circuit delays are measured on *simple paths* between PIs, PPIs and POs, PPOs.

A fanout node in a directed hypergraph has a *single* input and any number of outputs. Subsequently, all nets in $\mathcal{E}_0$ can be represented as two-terminal wires. The notion of splitting fanout nodes during partitioning has been introduced in [22].

Partitioned Hypergraph. Consider the example in Figure 1a. The unpartitioned hypergraph $\mathcal{H}_0$ consists of 8 board terminal nodes and a total of 15 logic and register nodes, and 7 fanout nodes. Assuming that the largest available device can accept at most 6 terminal nodes and 10 logic/register nodes, we generate a *partitioned hyperpgraph* $\mathcal{H}_3$ consisting of 3 feasible partitions: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$. In general, a partitioned hyperpgraph $\mathcal{H}_k$ consists of k non-overlaping partitions $\mathcal{P}_j$ where each partition is itself a hypergraph $\mathcal{P}_j = (\{\mathcal{X}_j; \mathcal{Y}_j\}, \mathcal{E}_j)$. Unlike the set of board terminal nodes which is unique, the set of *device terminal nodes* is not; it is highly dependent on the algorithm used for partitioning.

As shown in Figure 1b, partitioning has been achieved by (1) splitting the fanout node $h$, (2) assigning board nodes to terminal nodes of 3 partitions, and (3) assigning cuts to 4 more two-wire connections such that the device terminal and logic capacity constraints of each partition were met. In this paper, we also consider additional degrees during partitioning: replication of logic nodes to reduce the size of the cutset along with optimization and resynthesis of each partition.

Device Library. For continuity, we use the library introduced in [6], [19], [20]. Without loss of generality, we label library devices as a 5-tuple $\mathcal{D}_i = (c_i, t_i, d_i, l_i, u_i)$, representing the number of elementary logic units contained in the device, the number of terminals, the price, and the lower and upper bounds on the utilization of elementary circuit units. A hypergraph $\mathcal{P}_j = (\{\mathcal{X}_j; \mathcal{Y}_j\}, \mathcal{E}_j)$ meets the constraints of the device $\mathcal{D}_{i_j}$, denoted as $\mathcal{P}_j \models \mathcal{D}_{i_j}$, if and only if $l_i c_i \leq |\mathcal{X}_j| \leq u_i c_i$ and $|\mathcal{Y}_j| \leq t_i$. A $k$–way partition is called *feasible* if each $\mathcal{P}_j$ satisfies the relation $\mathcal{P}_j \models \mathcal{D}_{i_j}$ for some $i_j$, $j = 1, 2, ..., k$. We extend the library by associating the propagation delays with

Roman Kužnar was supported by Slovenian Ministry of Research and Technology under grant S28-0781-030/11535/93. Franc Brglez was supported by contracts from the Semiconductor Research Corporation (94-DJ-553), SEMATECH (94-DJ-800), and ARPA/ARO (P-33616-EL/DAAH04-94-G-2080). XACT tools from Xilinx Inc. facilitated reporting results of our experiments.

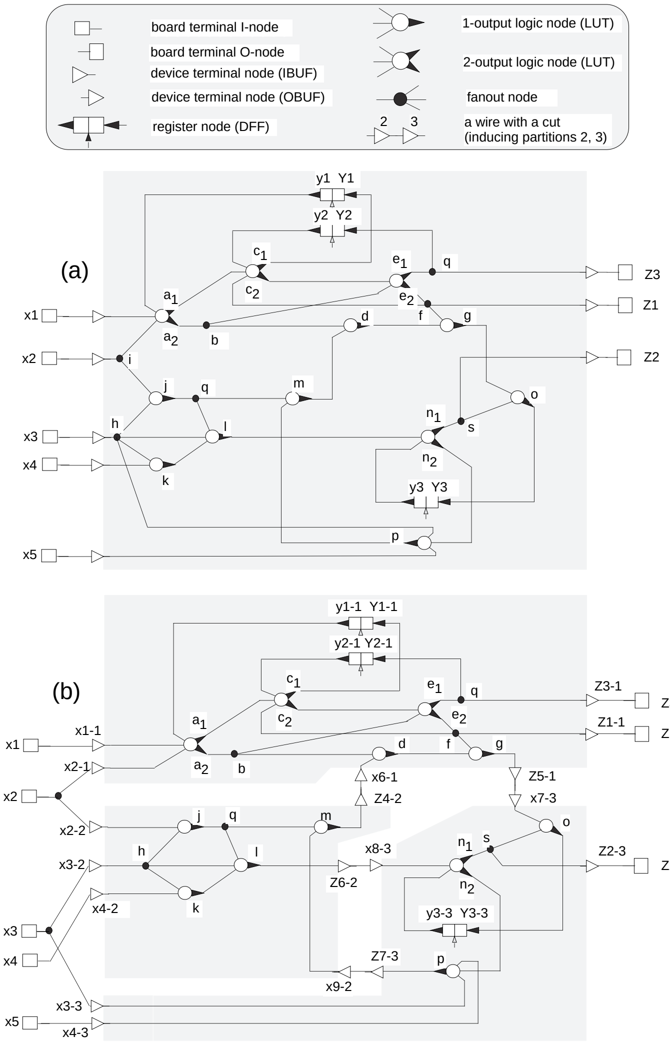

Fig. 1. (a) initial and (b) partitioned directed hypergraph.

the node types as used by the XACT tool [23]. The values are summarized in the Table I. By convention, the propagation delay of a D-flipflop is associated with its PPO.

TABLE I
A SUBSET OF THE XILINX XC3000 DEVICE LIBRARY

| Device<br>type | $c_i$<br>(CLB) | $t_i$<br>(IOB) | $d_i$<br>(N\$) | $l_i$ | $u_i$ |
|---|---|---|---|---|---|
| XC3020x-x | 64 | 64 | 1.00 | 0.8 | 0.9 |
| XC3030x-x | 100 | 80 | 1.36 | 0.8 | 0.9 |
| XC3042x-x | 144 | 96 | 1.84 | 0.8 | 0.9 |
| XC3064x-x | 224 | 110 | 3.15 | 0.8 | 0.9 |
| XC3090x-x | 320 | 144 | 4.83 | 0.8 | 0.9 |

| Node type | Propagation delay | |
|---|---|---|
| | (XC3000xxx-150) | (XC3100xxx-3) |
| 1-output, 2-output LUT | 4.6ns | 3.2ns |
| D-flipflop | 4.0ns | 4.0ns |
| IBUF (unclocked) | 2.8ns | 2.2ns |
| OBUF (unclocked) | 19.5ns | 9.0ns |

*Delay Modeling.* We need to characterize the *delay* of individual partitions $\mathcal{P}_j$ as well as of the partitioned hypergraph $\mathcal{H}_k$. We measure the delays on *simple paths* that start either at PI or PPI and terminate at PO or a PPO. The topological critical path is defined and evaluated as described in [24]. In this work, we make no attempt to identify false paths.

Critical path delay in a hypergraph and each of its partitions can be conveniently represented in a matrix form. Given the example in Figure 1a, we find

$$\tau_{\mathcal{H}_0} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ y_1 \\ y_2 \\ y_3 \end{array} \left[ \begin{array}{cccccc} Z_1 & Z_3 & Z_2 & Y_1 & Y_2 & Y_3 \\ 36.1 & 36.1 & - & 25.2 & 20.6 & 29.8 \\ 36.1 & 36.1 & 36.1 & 25.2 & 20.6 & 43.6 \\ - & - & 36.1 & - & - & 43.6 \\ - & - & 36.1 & - & - & 43.6 \\ - & - & - & - & - & 29.8 \\ 35.3 & 33.3 & - & 22.4 & 17.8 & 27.0 \\ 28.7 & 28.7 & - & 20.6 & 16.0 & 22.4 \\ - & - & 24.1 & - & - & 13.2 \end{array} \right]$$

i.e. the topological critical path in the unpartitioned hypergraph $\mathcal{H}_0$ is $< x_4 - k - l - n_2 - p - m - d - g - o - Y3 >$ and has the delay of 43.6 ns. Additional buffer nodes introduced by the cuts on the same path in the partitioned hypergraph $\mathcal{H}_3$ in Figure 1b increase the delay to 132.8 ns:

$$\tau_{\mathcal{H}_3} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ y_1{-}1 \\ y_2{-}1 \\ y_3{-}3 \end{array} \left[ \begin{array}{cccccc} Z_1 & Z_3 & Z_2 & Y_1{-}1 & Y_2{-}1 & Y_3{-}3 \\ 36.1 & 36.1 & - & 25.2 & 20.6 & 52.1 \\ 36.1 & 36.1 & 58.4 & 25.2 & 20.6 & 132.8 \\ - & - & 58.4 & - & - & 132.8 \\ - & - & 58.4 & - & - & 132.8 \\ - & - & - & - & - & 96.7 \\ 33.3 & 33.3 & - & 22.4 & 17.8 & 49.3 \\ 28.7 & 28.7 & - & 20.6 & 16.0 & 44.7 \\ - & - & 24.1 & - & - & 13.2 \end{array} \right]$$

We traversed partitions $\mathcal{P}_2$ and $\mathcal{P}_3$ twice in this evaluation. Similarly, we traverse node $c$ twice in the partition $\mathcal{P}_1$, first through the output $c_2$, and the second time through the output $c_1$. Critical path delays of partitions $\mathcal{P}_2$ and $\mathcal{P}_3$ are:

$$\tau_{\mathcal{P}_2} = \begin{array}{c} \\ x_2{-}2 \\ x_3{-}2 \\ x_4{-}2 \\ x_9{-}2 \end{array} \left[ \begin{array}{cc} Z_4{-}2 & Z_6{-}2 \\ 31.5 & 31.5 \\ 31.5 & 31.5 \\ - & 31.5 \\ 26.9 & - \end{array} \right] \quad \tau_{\mathcal{P}_3} = \begin{array}{c} \\ x_3{-}3 \\ x_5{-}3 \\ x_7{-}3 \\ x_8{-}3 \\ y_3{-}3 \end{array} \left[ \begin{array}{ccc} Z_2{-}3 & Z_7{-}3 & Y_3{-}3 \\ - & 26.9 & - \\ - & 26.9 & - \\ - & - & 11.4 \\ 31.5 & 31.5 & 16.0 \\ 24.1 & 28.7 & 13.2 \end{array} \right]$$

*Performance Measures.* Let $n_i$ be the number of devices of type $i$ to be used in the $k$–way partition, and $c_{\mathcal{P}_j}$, $t_{\mathcal{P}_j}$, be the number of elementary logic units and the number of terminals, respectively. The topological critical path delay $\tau_k$ is based on the partitioned hypergraph $\mathcal{H}_k$. Then:

$$k \triangleq \sum_{i=1}^{q} n_i \qquad \text{(Total number of partitions)} \tag{1}$$

$$\$_k \triangleq \sum_{i=1}^{q} d_i n_i \qquad \text{(Total device cost)} \tag{2}$$

$$\overline{c}_k \triangleq \sum_{j=1}^{k} c_{\mathcal{P}_j} / \sum_{i=1}^{q} c_i n_i \ \text{(Average logic utilization)} \tag{3}$$

$$\overline{t}_k \triangleq \sum_{j=1}^{k} t_{\mathcal{P}_j} / \sum_{i=1}^{q} t_i n_i \ \text{(Average terminal utilization)} \tag{4}$$

$$\tau_k \triangleq \max_{\forall\ paths} (\tau_{\mathcal{H}_k}) \ \text{(Topological critical path delay)} \tag{5}$$

The optimization objective in this paper is simply minimization of $\$_k$ for the set of all feasible partitions, while also reporting values of $\overline{c}_k$, $\overline{t}_k$, and $\tau_k$.

Lower Bounds on the Device Cost. Given the formulation in (2), we generate a lower bound on the total device cost $\$_k$ by solving an ILP problem [20]. This bound is based on the simplifying assumption of disregarding all circuit nets. In practice, one would therefore expect the bound to be very conservative; i.e. total number of actual partitions into library devices may significantly exceed the bound.

This paper demonstrates that the number of circuit partitions of *digital logic* need not necessarily be considered bounded from below even by such conservative lower bounds. To simplify presentation and experiments, we consider partitions of equal size only. Then, the nominal bound on the cost of a partition for a given device $\mathcal{D}_i$ is simply

$$LB_i = \lceil max(\frac{|\mathcal{X}_0|}{u_i c_i}, \frac{|\mathcal{Y}_0|}{t_i})\rceil \qquad (6)$$

For example, take the circuit c6288 with 833 logic nodes and a device type XC3020 as described in[19]. With 90% logic utilization, we can partition the circuit c6288 into at most 15 devices of type XC3020. The smallest number of XC3020 devices that make up a partition of this circuit has been reported at 16, with average logic utilization of $\overline{c}_k$ ranging from 0.81 to 0.86. However, as shown in Table II, we now can partition the same circuit into only 12 devices of type XC3020, with average logic utilization of $\overline{c}_k = 0.87$.

## III. Motivation

Consider three versions of a simple, logically equivalent, circuit in Figure 2. The circuit in Figure 2a shows a netlist of three cells that were generated with a Xilinx tool [23] upon submitting a netlist of simple 2-input gates shown in the same figure. Assume, that the maximum size capacity of each partition is at most two cells. Currently, the size of the cutset $\{B, C, D, Y3\}$ is 5. Moving the middle cell across the cutline eliminates nets $\{D, E, Y3\}$ from the cutset while adding nets $\{X3, A\}$ to the cutset, decreasing the cutsize by 1 net. Moving the top cell across the cutline introduces nets $\{W, A\}$ to the cut set, increasing the size of the cut set by 2 nets. The most effective operation is a functional replication of the middle cell shown in the Figure 2b, reducing the cutset by 2 nets: $\{Q, Y3\}$. Each functional replication increases logic cell count by 1 cell. For the case shown here, replication can also introduce redundant logic. Cell outputs $\{Y3, X3r\}$ are unobservable and thus any logic, observable at these outputs only, is redundant. By resynthesizing each of the partitions independently of the other, we can remove all redundant logic as well. The newly optimized technology-independent representation of each partition may now be remapped again – and possibly into a smaller number of cells. As shown in Figure 2c, we have now reduced the size of *each* partition by 1 cell.

This example illustrates that *optimizing* each partition during partitioning may be as important as minimizing the size of the cutset between partitions. Notably, the concept of functional replication such as in [6], [19] not only reduces the size of the cutset beyond the traditional approach but may also raise the amount of reconvergence. Compared to the lower partition in the Figure 2a which has no signal reconvergence at output $X2$, we now have a reconverging signal at the output $X2$ in Figure 2b. Partitions based on signal reconvergence have been shown to lead to much improved optimization results for large circuits [22].

We report on large scale partitioning experiments with *PROP* in Table II. Figure 3 depicts the trend of monotonically reducing the number of devices towards the easily computed lower bounds of 172 and 81 when partitioning into XC3020 and XC3042 devices, respectively. The trend is unmistakeable only after increasing the sophistication of the partitioning algorithm: from *(p, p)*, *(p, r, p)* as reported in [20], [19] to *(p, o, p)* and *(p, r, o, p)* as reported in this paper. Notably, these results are based on *total device counts in all partitions.* As pointed out earlier and shown in Table II, we can demonstrate *special* cases of benchmarks where partitions consist of *less* devices than predicted with the lower bound.

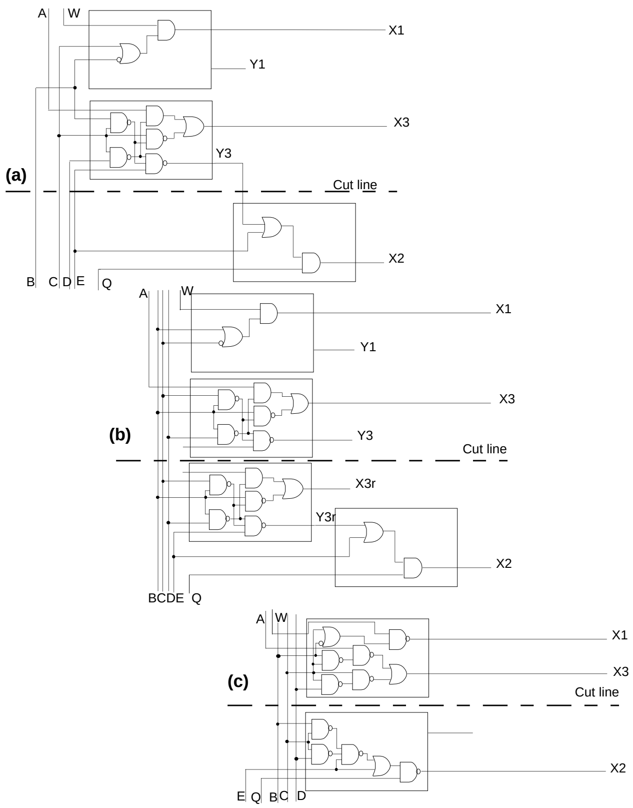

Fig. 2. On reducing the cutset during bipartitioning.

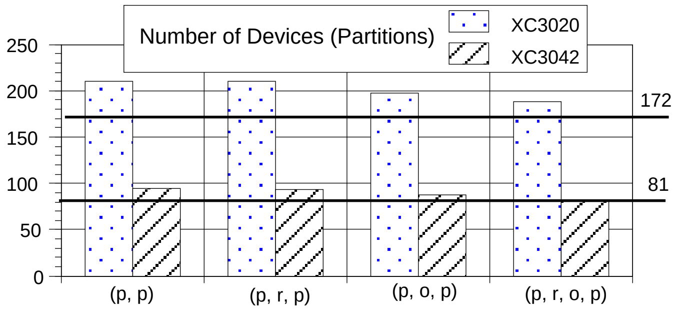

Fig. 3. Impact of distinct partitioning strategies.

## IV. Algorithm *PROP*

*PROP* is a greedy heuristic which combines logic resynthesis with replication-based partitioning until each *trial* partition becomes feasible. We present an overview of the algorithm, including some of the more important choices and decisions. Next, we discuss procedure for technology-independent optimization, technology-mapping, and dynamic balancing criteria. We conclude with an example to illustrate the progression of the algorithm for a large benchmark example.

*Overview.* As in [6], [19], *PROP* makes use of the the recursive paradigm using functional replication. For a given circuit, it applies a highly optimized bipartitioning procedure to extract the first feasible partition that fits a given device, and then repeats the procedure on the circuit remainder until the remainder fits at least one device from the target library.

The distinguishing feature of *PROP* is that during each recursion $k$, it bipartitions the remainder circuit $\mathcal{R}_k$ into two partitions: a *trial partition* $\mathcal{P}_{k+1} \models D_i|u_{k+1}$ with the utilization bounds $[l_{k+1}, u_{k+1}]$ set *above upper utilization bound* $l_i$ of the target device $D_i$, and a *remainder* partition $\mathcal{R}_{k+1}$. For each $k$, the algorithm produces a bipartition with at least one feasible partition $\mathcal{P}_{k+1}$ at the end of the iteration. The process repeats for as long the remainder partition does not meet the constraint of the device: i.e. $\mathcal{R}_k \not\models D_i$. Key steps of *PROP* are illustrated in Figure 4. We outline procedures *Optimize($\mathcal{P}_{k+1}$)* and *Update($\beta$)* next.

```
Input: $\mathcal{H}_0(\{\mathcal{X}_0;\mathcal{Y}_0\},\mathcal{E}_0)$, $\{D_i\}$
Output: $k$, $\$_k$, $\mathcal{P}_1, \mathcal{P}_2, .., \mathcal{P}_k$
$k = 0$; $\mathcal{R}_0 = \mathcal{H}_0$;              /* initialize k, remainder */
$Initialize(\beta, \gamma, u_{k+1}, l_{k+1})$; /* init. optim. param.   */
while ($\mathcal{R}_k \not\models D_i$) begin
    if ($\mathcal{R}_k \models D_i|u_{k+1}$) then
        $Optimize(\mathcal{R}_k)$;        /*optimize remaider */
    if ($\mathcal{R}_k \models D_i$) then
        $\mathcal{P}_{k+1} \leftarrow \mathcal{R}_k$;
    while ($\mathcal{P}_{k+1} \not\models D_i$)  begin
        $Partition\ \mathcal{R}_k = \{\mathcal{P}_{k+1};\mathcal{R}_{k+1}\}$
                 subject to $\mathcal{P}_{k+1} \models D_i|u_{k+1}$;
        $Optimize(\mathcal{P}_{k+1})$;    /* optimize partition */
        if ($\mathcal{P}_{k+1} \not\models D_i$) then
            $Update(\beta)$;
            $u_{k+1} = \beta u_{k+1}$; $l_{k+1} = \beta l_{k+1}$;
        end
    end while;
    $Output(\mathcal{P}_{k+1})$;
    $k = k + 1$;
end while
```

Fig. 4. Top view of algorithm *PROP*.

*Procedure Optimize($\mathcal{P}_{k+1}$).* The main task of optimization is to minimize the amount of logic within the partition since each trial partition initially exceeds the bounds of the target device. If the optimization produces a feasible partition, e.g. a partition which meets the constraints of the target device ($\mathcal{P}_{k+1} \models D_i$), we store the solution and continue partitioning the remainder. If the optimizer fails to produce a feasible partition, we decrease utilization bounds and repeat bipartitioning, starting with a new trial partition. By maximizing the number of over-utilized *trial* partitions that become feasible after optimization, we take full advantage of the functional replication and reduce total interconnect between the partitions while reducing the cell count $|\mathcal{X}_k|$ in the remainder circuit. Experimental results demonstrate that the reduction of the remainder circuit also tends to reduce the total device count.

Since the size of each feasible partition is measured *after* technology mapping, the number of logic modules (e.g. Xilinx Configurable Logic Blocks, CLBs) must not exceed the target utilization of a given FPGA device. We separate optimization steps into *technology independent logic optimization* which is performed with *sis* [25] and *technology mapping* of the logic into Configurable Logic Blocks, which is perfomed with the technology mapper *xnfmap* [23]. Although *sis* can perform technology mapping into XC3000 family for combinational circuits, we use the technology mapper *xnfmap* for the following reasons: (1) a netlist mapped with *sis* is usually harder to route in XC3000 family, (b) *sis* technology mapping produced inferior results compared to *xnfmap* results in terms of CLB count for a set of combinational circuits, and (3) *sis* cannot be used for technology mapping of *sequential* circuits into the XC3000 family.

Currently, we only perform the combinational optimization for combinational and sequential benchmark circuits. For each CLB in the partition, we extract the functionality of Look-Up Tables (LUTs) in order to translate the netlist into *blif* format for technology-independent optimization by *sis*. All flip-flops are replaced with PPIs and PPOs. The initial netlist which is passed to *sis* is based on the XC3000 device netlist and is thus bounded to have at most $n = 5$ inputs. However, upon minimization of logic nodes in *sis*, we do not use operations for vertex minimization such as *xl_cover* and *xl_partition* since this may lead to poor technology mapped results with *xnfmap*. We obtain better results by using a script which in the last step performs the decomposition into simpler local functions instead of performing vertex minimization. To date, we find that the following script produces the best results:

```
eliminate 2; simplify; xl_part_coll; xl_coll_ck;
simplify; decomp -q; full_simplify;
```

For the combinational circuits, the netlist from *sis* is immediately mapped by *xnfmap*. For sequential circuit, the combinational part of the circuit is merged with the flip-flops, i.e. all PPIs and PPOs are eliminated from the combinational netlist, and the netlist with flip-flops is then mapped by *xnfmap*.

*Procedure Update($\beta$).* The question arises how to choose the 'best' balancing constraints when generating over-utilized trial partitions. Setting the utilization bounds $[l_{k+1}, u_{k+1}]$ too high may prevent the optimizer from producing a feasible partition. In this case, we have to backtrack and repeat partitioning and optimization until a feasible partition, which meets the constraints, is found. Since optimization of partitions can be CPU-intensive, repeating several optimizations during each bipartitioning iteration may drastically increase the execution time of the partitioning process. Conversely, too low an utilization of trial partitions may not effectively reduce the size of the remainder partitions, resulting in a suboptimal solution.

We balance the size of the over-utilized trial partitions in each bipartition step according to the following criteria:

$$\beta\gamma c_i\,(1-\xi) \le \ |\mathcal{P}_{k+1}| \le \ \beta\gamma c_i\,(1+\xi) \qquad if\ |\mathcal{R}_k| > 2\gamma c_i(1+\xi),$$
$$|\mathcal{P}_{k+1}| \le \beta\frac{|\mathcal{R}_k|}{2}(1+\xi) \qquad otherwise \qquad (7)$$

where $c_i$ represents the number of logic modules per device $i$, $\gamma$ reflects the expected ratio of the size reduction achieved with the optimization, and the $\xi$ is a balancing factor which permits maximum deviation in size of feasible partition. Here, $\beta$ is the scaling factor by which the utilization bounds decrease if optimization fails. Initially, in each new bipartitioning iteration, $\beta = 1$. If the optimization fails, the scaling factor gets updated: $\beta_{new} = \beta_{old}\frac{(1-\xi)}{(1+\xi)} < 1$.

In all experiments, we set the parameters $\gamma = 0.95$, $\xi = 0.05$ and $\beta = \{1.00, 0.90\}$ for first and second optimization attempt in one bipartitioning step. These values are based on experimental observation that optimization in most cases

```
PROP Version 1.1
================

Device Library = XC3042

     F E A S I B L E       ||     R E M A I N D E R
-------------------------------------------------------
Step.   #of CLB    #of IOB ||  #of CLB   #of IOB    LB
    (p,r) (p,r,o,p)
=======================================================
0           ---      --    ||    842       102       7
1   (125)   107      74    ||    739        56       6
2   (143)   119      27    ||    605        79       5
3   (129)   115      37    ||    491        92       4
4   (132)   101      85    ||    373        96       3
5   (143)   108      82    ||    235        84       2
6   (108)    93      65    ||    129        83       1
7   (129)    93      83    ||    ---       ---
=======================================================
TOTAL:(909)   736     453  ||    ---       ---
=======================================================
```

Fig. 5. Progression of *PROP* for circuit s15850.

reduces the size of the partitions on average by 15% and that the average percentage of the replicated cell over all benchmark circuits is around 5% [6]. Such parameters also ensure that feasible partitions which meet the constraints of the target device are found at most in the second optimization attempt in each bipartitioning step for 90% target device utilization.

*Illustrative Example.* We illustrate the progression of the *PROP* algorithm by partitioning the benchmark circuit s15850 into a set of XC3042 devices. The maximum utilization bound of the device is set to $u_i = 0.9$. For the given circuit the number of interior nodes $|\mathcal{X}_0| = 842$, the number of terminal nodes $|\mathcal{Y}_0| = 102$, and the calculated lower bound on the device count is

$$LB_{s15850} = \lceil max(\frac{|\mathcal{X}_0|}{(c_i u_i)}, \frac{|\mathcal{Y}_0|}{t_i})\rceil = \lceil max(\frac{842}{(144x0.9)}, \frac{102}{96})\rceil = 7.$$

During the first five bipartitioning steps, the bounds of the over-utilized feasible partition are calculated as $|\mathcal{X}_j| \in \{[130, 144], [122, 130]\}$ respectively (first and second optimization attempt), and to $|\mathcal{X}_j| \leq 129$ during the last bipartitioning step.

Figure 5 illustrates the progression of the partitioning process. Columns show the iteration step, size and terminal count of the feasible partition and the remainder partition, and the calculated lower bound for the remainder partition. Values of the logic modules count under *(p, r)* and *(p, r, o, p)* are given for the non-optimized over-utilized trial partitions, and optimized partitions, respectively. Data shown in Figure 5 indicates that terminal constraint was never an active constraint; I/O terminal count of feasible partitions does not approach the I/O constraints of 96 terminals per device. We can conclude that functional replication was quite effective in reducing the interconnect between partitions. However, total number of logic modules in *non-optimized* partitions increased from the initial 842 to 909 logic modules. Simple calculation shows that with 909 logic modules the lower bound on total device count for XC3042 devices $LB = \lceil\frac{909}{144x0.9}\rceil = 8$, indicating that using functional replications without optimization of each partition leads to suboptimal implementation. Using optimization on each of the partitions, we not only maintain the lower bound of 7 devices but also reduce total logic module count from an initial 842 to 736 CLBs, decreasing the average density of logic per device.

## V. Experimental Results

Charaterization of partititioning benchmarks and FPGA libraries introduced in [20] and [19] has been extended to include a simple delay model in [21]. This section summarizes a subset of extensive experiments with *PROP* that have been archived for easy access, comparison, and reproducibility, in directories accessible from [21].

The main objective of current experiments is strictly the reduction of the total device count ($\$_k$, $k$) while reporting *all* of the additional performance measures introduced in (3-5): $\bar{c}_k$, $\bar{t}_k$, and $\tau_k$. In Table II we report results of partitioning into either XC3020 or XC3042 device family. The columns labeled as $\tau_0$ and $LB_i^*$ are fundamental reference points. The critical path delay $\tau_0$ (in nanoseconds) refers to the *unpartitioned* CLB-level netlists, using XC3000xxx-150 delay parameters in Table I. The lower bound $LB_i^*$ is based on a simple formula in the footnote of the table for each device family. Except for the delay $\tau_k$, columns under $(p,p)$ and $(p,r,p)$ are copies of the best results from [19]. These results can now be directly compared to the best results, in columns under $(p,o,p)$ and $(p,r,o,p)$, attainable with *PROP* as presented in this paper.

The chart in Figure 3 relates the total number of devices in all partitions in Table II to the respective total lower bound of 172 and 81. There is a clear trend of consistent *reduction* in total device count when we move from the earlier strategies based on $(p,p)$ and $(p,r,p)$ to the newly proposed strategy based on $(p,o,p)$ and $(p,r,o,p)$. While no attempt has been made to reduce the total delay across all partitioned devices, it is encouraging that the current strategy of reducing the total device count *has not* impacted total delay in any major way. Overall, there is only a slight increase in the total delay when moving from $(p,o,p)$ to $(p,r,o,p)$. In both cases, the total delay is better than one reported under $(p,p)$ and $(p,r,p)$.

In addition to a consistent trend overall, data in Table II depicts several instances of interesting and surprising results. Consider the circuit c6288. With 90% logic utilization, we can partition this circuit into at most 15 devices of type XC3020. While results based on $(p,p)$ and $(p,r,p)$ report partitions of 16 devices with average logic utilization of $\bar{c}_k$ ranging from 0.81 to 0.86, a strategy based on $(p,o,p)$ and $(p,r,o,p)$ partitions the same circuit into only 12 devices with $\bar{c}_k = 0.87$. In addition, there is a noticeable reduction in delay when we achieve a 12-device partition, compared to the 15-device partition. Similarly, when partitioning largest benchmark s38584, only the reduction of total device count using $(p,r,o,p)$ results in a reduction of the critical path delay across all partitions!

## VI. Conclusions

Experimental results reported in this paper and in [21] show the greatest reduction in the total number of devices under the column $(p,r,o,p)$ in Table II; i.e. when we use *PROP* to combine logic re-synthesis with replication-based partitioning until each *trial* partition becomes feasible.

Results under the column $(p,r,o,p)$ also show that we currently achieve the average utilization of devices in each device at 0.70 for both device sets. This utilization is below the target utilization of 80%-90%. Such result suggest that some partitions may have greater *minimization* and *resynthesis* potential, an issue not addressed in the current implementation. A new version of *PROP* will be guided by estimating the resynthesis potential of each feasible partition before it is passed to the partitioner, improving the decisions during the progression of the partitioning process.

In addition, analysis of delay performance of partitions reported in this paper indicates that it should be possible to reduce critical path delay further.

TABLE II
Partitioning into sets of XC3020 and XC3042 devices

| | | Partitioning into XC3020 devices | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *Best from [19]* | | | | | | | | *Best with PROP* | | | | | | | | |
| | | *(p,p)* | | | | *(p,r,p)* | | | | *(p,o,p)* | | | | *(p,r,o,p)* | | | | |
| Circuit | $\tau_0$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $LB_i^*$ |
| c3540 | 128 | 0.74 | 0.90 | 216 | 6 | 0.80 | 0.79 | 213 | 6 | 0.64 | 0.91 | 211 | 6 | 0.66 | 0.80 | 189 | 6 | 5 |
| c5315 | 78 | 0.65 | 0.86 | 144 | 9 | 0.73 | 0.90 | 144 | 8 | 0.55 | 0.81 | 149 | 9 | 0.62 | 0.90 | 154 | 8 | 7 |
| c6288 | 440 | 0.81 | 0.50 | 708 | 16 | 0.86 | 0.45 | 708 | 16 | 0.87 | 0.60 | 573 | 12 | 0.87 | 0.55 | 662 | 12 | 15 |
| c7552 | 73 | 0.76 | 0.77 | 206 | 10 | 0.77 | 0.81 | 184 | 10 | 0.64 | 0.79 | 167 | 9 | 0.62 | 0.77 | 176 | 9 | 9 |
| s5378 | 64 | 0.54 | 0.87 | 153 | 11 | 0.68 | 0.81 | 176 | 10 | 0.49 | 0.82 | 126 | 11 | 0.61 | 0.78 | 171 | 9 | 7 |
| s9234 | 67 | 0.79 | 0.58 | 167 | 10 | 0.79 | 0.58 | 167 | 10 | 0.72 | 0.75 | 144 | 9 | 0.72 | 0.65 | 173 | 9 | 8 |
| s13207 | 87 | 0.75 | 0.63 | 265 | 23 | 0.75 | 0.67 | 252 | 23 | 0.63 | 0.84 | 216 | 21 | 0.68 | 0.69 | 265 | 19 | 16 |
| s15850 | 87 | 0.69 | 0.76 | 281 | 19 | 0.78 | 0.67 | 281 | 19 | 0.72 | 0.77 | 291 | 17 | 0.75 | 0.69 | 274 | 16 | 15 |
| s38417 | 58 | 0.75 | 0.68 | 241 | 46 | 0.83 | 0.50 | 215 | 48 | 0.73 | 0.64 | 221 | 44 | 0.81 | 0.53 | 201 | 44 | 39 |
| s38584 | 81 | 0.76 | 0.70 | 227 | 60 | 0.81 | 0.62 | 277 | 60 | 0.74 | 0.69 | 238 | 60 | 0.77 | 0.61 | 180 | 56 | 51 |
| | 1163 | 0.72 | 0.73 | 2615 | 210 | 0.80 | 0.68 | 2616 | 210 | 0.67 | 0.76 | 2336 | 198 | 0.70 | 0.70 | 2445 | 188 | 172 |

*Lower bound on partitions, given 90% utilization of each device $LB_i = \lceil(\text{Total CLBs in the circuit})/(0.9 \times 64)\rceil$

| | | Partitioning into XC3042 devices | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *Best from [19]* | | | | | | | | *Best with PROP* | | | | | | | | |
| | | *(p,p)* | | | | *(p,r,p)* | | | | *(p,o,p)* | | | | *(p,r,o,p)* | | | | |
| Circuit | $\tau_0$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $\overline{c}_k$ | $\overline{t}_k$ | $\tau_k$ | $k$ | $LB_i^*$ |
| c3540 | 128 | 0.66 | 0.74 | 194 | 3 | 0.71 | 0.68 | 194 | 3 | 0.66 | 0.75 | 225 | 2 | 0.69 | 0.89 | 212 | 2 | 3 |
| c5315 | 78 | 0.51 | 0.77 | 122 | 5 | 0.53 | 0.78 | 122 | 5 | 0.54 | 0.87 | 109 | 4 | 0.53 | 0.83 | 109 | 4 | 4 |
| c6288 | 440 | 0.85 | 0.89 | 642 | 7 | 0.85 | 0.45 | 686 | 7 | 0.79 | 0.59 | 545 | 6 | 0.86 | 0.55 | 537 | 5 | 7 |
| c7552 | 73 | 0.83 | 0.49 | 184 | 4 | 0.85 | 0.89 | 184 | 4 | 0.49 | 0.84 | 149 | 5 | 0.67 | 0.90 | 127 | 4 | 4 |
| s5378 | 64 | 0.53 | 0.88 | 153 | 5 | 0.73 | 0.83 | 153 | 4 | 0.48 | 0.86 | 148 | 4 | 0.57 | 0.89 | 139 | 4 | 3 |
| s9234 | 67 | 0.79 | 0.73 | 190 | 4 | 0.85 | 0.52 | 156 | 4 | 0.70 | 0.78 | 111 | 4 | 0.72 | 0.65 | 142 | 4 | 4 |
| s13207 | 87 | 0.58 | 0.84 | 243 | 11 | 0.76 | 0.65 | 229 | 10 | 0.61 | 0.75 | 207 | 9 | 0.69 | 0.76 | 225 | 8 | 8 |
| s15850 | 87 | 0.73 | 0.75 | 327 | 8 | 0.78 | 0.69 | 322 | 9 | 0.66 | 0.82 | 251 | 8 | 0.67 | 0.73 | 282 | 7 | 7 |
| s38417 | 58 | 0.77 | 0.64 | 178 | 20 | 0.76 | 0.46 | 165 | 20 | 0.76 | 0.67 | 184 | 20 | 0.81 | 0.41 | 178 | 19 | 18 |
| s38584 | 81 | 0.75 | 0.62 | 251 | 27 | 0.84 | 0.35 | 205 | 27 | 0.76 | 0.67 | 189 | 25 | 0.80 | 0.56 | 176 | 25 | 23 |
| | 1163 | 0.70 | 0.74 | 2507 | 94 | 0.77 | 0.63 | 2416 | 93 | 0.65 | 0.75 | 2118 | 87 | 0.70 | 0.72 | 2127 | 82 | 81 |

*Lower bound on partitions, given 90% utilization of each device $LB_i = \lceil(\text{Total CLBs in the circuit})/(0.9 \times 144)\rceil$

## References

[1] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.

[2] C. W. Yeh and C. K. Cheng. A general purpose multiple way partitioning algorithm. In *In $28^{th}$ DAC ACM/IEEE*, pages 421–426, 1991.

[3] J. Cong and M. Smith. A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design. In *30th DAC, ACM/IEEE*, pages 755–760, June 1993.

[4] C. Kring and A. R. Newton. A Cell-Replicating Approach to Mincut-Based Circuit Partitioning. In *IEEE ICCAD-91*, pages 2–5, November 1991.

[5] J. Hwang and A. El Gamal. Optimal Replication for Min-Cut Partitioning. In *IEEE ICCAD*, pages 432–435, November 1992.

[6] R. Kužnar, F. Brglez, and B. Zajc. Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect. In *ACM/IEEE 31st DAC*, pages 238–243, June 1994.

[7] P.K. Chan, M.D.F Schlag, and J.Y. Zien. Spectral K-Way Ratio-Cut Partitioning and Clustering. In *30th DAC, ACM/IEEE*, pages 749–754, June 1993.

[8] C. J. Alpert and So-Zen Yao. Spectral Partitioning: The More Eigenvectors, The Better. In *32th DAC, ACM/IEEE*, pages 195–200, June 1995.

[9] M. Shih and E.S. Kuh. Quadratic Boolean Programming for Performance-Driven System Partitioning. In *30th DAC, ACM/IEEE*, pages 761–765, June 1993.

[10] C. J. Alpert and A. B. Kahng. Multi-Way Partitioning Via Spacefilling Curves and Dynamic Programming. In *31th DAC, ACM/IEEE*, pages 652–657, June 1994.

[11] B. M. Reiss, K. Doll, and F.M. Johannes. Partitioning Very Large Circuits Using Analytical Placement Techniques. In *31th DAC, ACM/IEEE*, pages 646–651, June 1994.

[12] G. Saucier, D. Brasen, and J. P. Hiol. Partitioning with Cone Structures. In *IEEE ICCAD*, pages 236–239, November 1993.

[13] H. Yang and D. F. Wong. Efficient Network Flow Based Min-Cut Balanced Partitioning. In *IEEE ICCAD-94*, pages 50–55, November 1994.

[14] J. Cong, W. Labio, and N. Shivakumar. Multi-Way VLSI Circuit Partitioning Based on Dual Net Representation. In *IEEE ICCAD-94*, pages 56–62, November 1994.

[15] N.-S. Woo and J. Kim. An Efficient Method of Partitioning Circuits for Multiple- FPGA Implementation. In *30th DAC, ACM/IEEE*, pages 202–207, June 1993.

[16] N.C. Chou, L.T. Liu, C.K. Cheng, W.J. Dai, and R. Lindelof. Circuit Partitioning for Huge Logic Emulation Systems. In *31th DAC, ACM/IEEE*, pages 244–249, June 1994.

[17] P. Sawkar and D. Thomas. Multi-Way Partitioning For Minimum Delay For Look-Up Table Based FPGAs. In *32th DAC, ACM/IEEE*, pages 201–205, June 1995.

[18] L. T. Liu, M. T. Kuo, C. K. Cheng, and T. C. Hu. Performance-Driven Partitioning using a Replication Graph Approach. In *32th DAC, ACM/IEEE*, pages 206–210, June 1995.

[19] R. Kužnar, F. Brglez, and B. Zajc. A Unified Cost Model for Min-Cut Partitioning with Replication Applied to Optimization of Large Heterogeneous FPGA Partitions. In *EURO-DAC '94*, September 1994.

[20] R. Kužnar, F. Brglez, and K. Kozminski. Cost Minimizations of Partitions into Multiple Devices. In *ACM/IEEE 30st DAC*, pages 315–320, June 1993.

[21] R. Kužnar. Latest Partitioning Results Update and Directories. Available from http://www.cbl.ncsu.edu/k̃uznar/, August 1995. For an autoreply on up-to-date access to all benchmark directories, send e-mail to benchmarks@cbl.ncsu.edu.

[22] S. Dey, F. Brglez, and G. Kedem. Circuit Partitioning for Logic Synthesis. *IEEE Journal of Solid-State Circuits*, 26(3):350 – 363, March 1991.

[23] Xilinx. *User Guide and Tutorials.* Xilinx Incorporation, 2100 Logic Drive, San Jose, California, 1991.

[24] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw Hill, New York, 1994.

[25] E. M. Sentovich et. al. *SIS: A System for Sequential Circuit Synthesis.* Dept. of Electrical Engineering and Computer Science, University of California, Berkeley,CA 94720, 1992.