MogaMap2: Multi-Objective Mapping Algorithm with parameter control for Optimize Area, Performance and Power Consumption in FPGA

V. L. Souza[1] and A. G. Silva-Filho[2]

[1] Souza, V. L., Informatics Center (CIn), Federal University of Pernambuco, Recife, Brazil
[2] Silva-Filho, A. G., Informatics Center (CIn), Federal University of Pernambuco, Recife, Brazil,
e-mail: vlss@cin.ufpe.br, agsf@cin.ufpe.br

ABSTRACT

This article presents a new technology mapper, MogaMap2, the second generation of the technology mapper, MogaMap, based on a hybrid approach that use evolutionary algorithm associated with specific heuristics of the problem in order to find better trade-off results among area, performance and power consumption. Different from MogaMap, the new approach includes a deterministic parameter control that, during the process, modifies the mutation probability. In a set of 20 large designs, we find that this adjust of parameter allow to reduce, in average, the LUT count in 2% and the edge count in 4%. In comparison to state-of-the-art technology mapping, our approach is able to reduce the LUT counts in 3% and the edges count in 10%. Placing and routing the resulting netlists leads to an 3% reduction in the complex logic blocks count, a 7% increasing in estimated operation frequency and 8% reduction in energy consumption.

.

Index Terms: Evolutionary Algorithm, FPGA, Technology Mapping.

I. INTRODUCTION

The FPGAs (Field Programmable Gate Arrays) were introduced in the 1980s with the purpose of provide reprogrammability, not available in ASICs (Application Specific Integrated Circuits). Initially, they were used only for fast realization of digital circuits. However, over the years, FPGAs have employed increasingly more complex logic blocks, including, memory blocks, Digital Signal Processing (DSP) blocks and even processors. Therefore, it became possible to develop more complex designs using this architecture. They have been used in aerospace and defense industry, medical solutions, wired and wireless communications and as hardware accelerators in scientific computing application.

Associated with an increase scale of application of FPGAs, there was also an increased need to improve their project metrics under some aspects: performance, area, and, mainly, power consumption, since FPGA are power inefficient compared to logically equivalent ASIC.

The quality of projects using FPGAs depends, not only, the hardware described by the designer but also the efficiency of the CAD tools that are responsible for converting the design onto a bitstream which is loaded onto the chip to program the various switches and obtain the desired functionality.

With this context in mind, the researchers are taking advantage of influence of the CAD flow on performance, area and energy minimization, to improve the algorithms in some of these purposes [1][2][3]. Advances are also present on the commercial tools. These are often optimized and the newest versions promise new capabilities, enhancements and performance improvements.

This article tries to contribute with this scientific demand by presenting the second generation of the technology map-per, MogaMap, presented in [4]. This new version is more efficient since, it adapts the mutation probability parameter, in order to adjust it to the different demands of evolutionary algorithm stages.

The rest of this article is organized as follows: Section II presents a background. The state of the art is shown in section III, section IV presents MogaMap2. The experimental results are presented in section V. The Sections VI show the conclusions.

II. BACKGROUND

A. FPGA

FPGA (Field Programmable Gate Array) is an architecture composed by I/O blocks, complex logic blocks (CLBs), routing channels, programmable routing architectures, beyond embedded blocks. A general view of FPGA Architecture can be seen in Fig. 1.

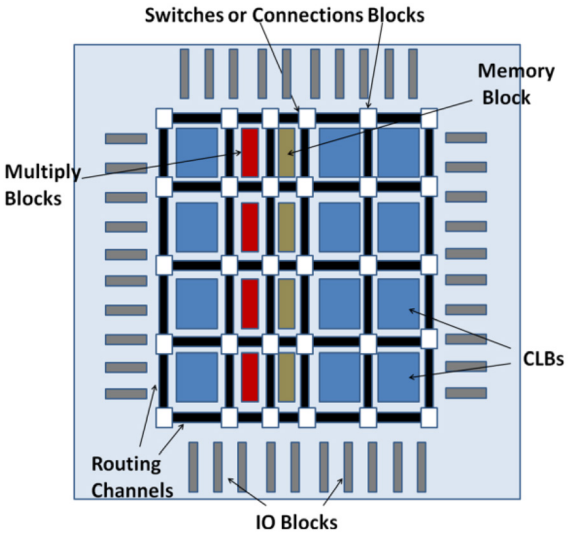

Figure 1. General view of FPGA Architecture

The CLBs represent the vast majority of blocks in the FPGA, and they are composed by a set of N basic logic elements (BLEs). Each BLE contains Look -Up Tables (LUTs) and registers. A K-Input LUT is able to implement any function of its K inputs. The modern architectures contain dual-output fracturable LUT instead of traditional single-output K LUT. The CLB have I inputs and N outputs (2N in case of fracturable LUTs). A simple architecture of BLE and the CLB composed by N BLEs is shown in Fig. 2.

The routing channels consist of wire segments that surround the CLBs from all sides. The input and output pins of a logic block can be connected to wire segments in the channels via a connection block. A routing switch block is located at the intersection of a horizontal channel and a vertical channel.

The embedded cores are special cores, like multiply blocks, memory blocks and, even, processors, that allow the efficient implementation of complex system on chip.

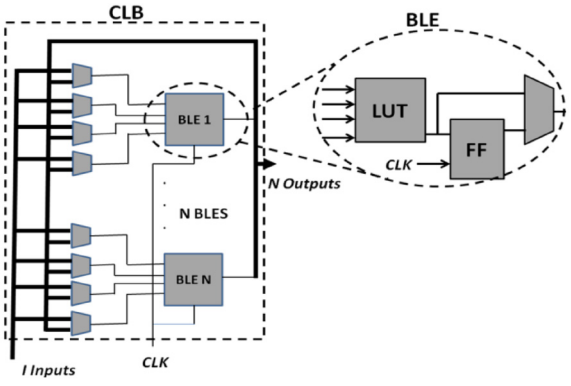

Figure 2. Architecture of BLE and CLB composed by N BLEs.

B. CAD Flow

The CAD tools are responsible for transform the designer logic described in hardware description language or as schematic, to a stream of "1"s and "0"s that program the FPGA during the configuration. The Fig. 3 illustrates the steps of a typical FPGA CAD flow.

The logic synthesis and optimization is a stage technology-independent. In this stage the logic functions are simplified and redundant logics are removed. The technology mapping is responsible for map the optimized user logic into the FPGA architectures composed by LUTs and flip-flops, according to explained in section II-A. After the mapping phase, the netlist of LUTs and flip-flops is converted into a netlist of complex logic blocks (CLBs), this process is executed in stage named packing. The placement stage is responsible for distribute the complex blocks among the physical blocks into the chip. Finally, the routing determines the resource routings that will be used to connect the blocks placed.

C. Technology Mapping

The technology mapping is treated as a covering problem. In order to solve this problem, a boolean network is represented by a direct acyclic graph (DAG), normally, this graph is an AIG (And-Inverter Graph). Technology mapping covers the circuit AIG with LUTs, each LUT in the mapped network implements a portion of the underlying AIG logic functionality.

A node in the graph represents a logic gate and an edge *(i,j)* exists only if *j* is fanout node of *i*. A node is a primary input (PI) if it does not have fanin and a node is a primary output if it does not have fanout. A fanin cone of a node *v* is a sub-network composed

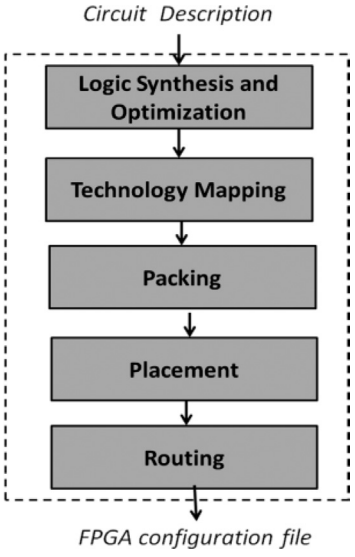

Figure 3. Typical FPGA CAD flow

by v and all the predecessors of v, such that any node u presents in the fanin cone has a path to v that lies entirely in the fanin cone. A cut is a partition (X_v, X_v') that split the fanin cone in two regions such that X_v' is a cone rooted at v. A cone rooted at v is a sub-network composed by v and some of its predecessors. A cut is K-feasible if the number of input edges in X_v' does not exceed K. In practice, a K-feasible cone can represent a K-inputs LUT. The level of a node is the length of the longest path from any PI to the node and the network depth is the largest level of an internal node in the network. A PI node has level equal to zero. The graph in Fig. 4 has 6 primary inputs (a to f), the fanin cone of the node i is circulated and the cut split this cone in two regions (X and X') and the cone rooted at i is composed by the nodes i and s. This cut is 3-feasible because it has 3 edges. The largest level of a node in the graph is 3, so the network depth is 3.

D. Evolutionary Algorithms

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution. The concept behind evolutionary algorithms comes from the idea of a population of individuals competing for limited resources in some environment. This process causes natural selection and causes a rise in the fitness of the population. The pseudo code of an evolutionary algorithm is given in Fig. 5.

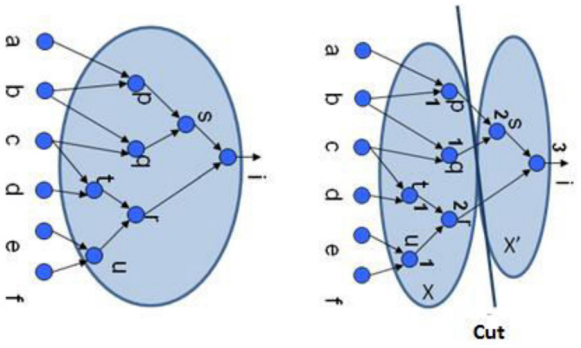

Figure 4. Graph representing a Boolean network

```
1:  Begin
2:   Initialize Population;
3:  Evaluate Individuals;
4:  While (termination condition is satisfied) do
5:       Select Parents;
6:          Recombine pairs of parents;
7:          Mutate the resulting offspring;
 8:         Evaluate new candidates;
 9:       Select Individuals for the next generation;
10:     end while;
11:  end;
```

Figure 5. Pseudo code of an evolutionary algorithm

The population is formed by a set of individuals, each individual encode a single possible solution to the problem. The evaluation process assigned to each individual is a fitness value. Individuals with high fitness represent better solutions than individuals with lower fitness.

The algorithm iterates until the termination condition is satisfied. Each iteration, the individuals in the current population are used to produce children using variation operators: recombination and mutation. Recombination is applied to two or more candidates and produce new candidates. The mutation is a perturbation applied to one candidate and results in one new candidate. The new candidates are evaluated and parents and children compete. The best one will survive and compose the new population. This process iterates until a candidate solution with sufficient quality is found or a set computational limit is reached.

III. STATE OF THE ART

In [5] is presented the mapper WireMap. WireMap uses an edge flow heuristic to improve the routability of a mapped design. The heuristic is applied during the iterative mapping optimization to reduce the total number of pin-to-pin connections (or edges). The WireMap proposes to associate the heuristics area flow and edge flow in a global optimization step, with specific heuristics of local area and local edge in a local optimization step. This technique, although being efficient for reducing the number of connections, does not use information about activity switching of the edges in order to reduce the power consumption.

The mapper proposed in [6] is based on an algorithm that was developed taking into consideration the power consumption. The algorithm run in three steps: In the first step are generated, for each node, the set of all K-feasible cones. The second step computes a cost function based on power estimation and depth for each K-feasible cut and in the third step, using the set of cost function, is determined the power-aware minimum depth mapped network. It is not possible to measure the reduction on power consumption obtained by the technique, because the experimental results do not make comparisons with the state of the art.

The SVmap-2 presented in [1] uses techniques, namely, global duplication cost adjustment, input sharing, and slack distribution. The algorithm uses cut-enumeration in order to generate all K-feasible cuts of each node. During the cut selection stage, the representative cut of a node that is not on a critical path is picked as long as it will not violate the timing constraint and will produce a better power in the same time. This work does not present a comparison with

algorithms that use efficient techniques to reduce area, like area flow.

The MogaMap was presented in [4], it is the only mapping approach based on evolutionary algorithm. MogaMap associates the flow heuristics with an evolutionary algorithm in order to find better trade-off results among area, performance and power consumption. The results presented in [4] show that it is quite efficient in reduce pos-route metrics. However, MogaMap does not use any parameters control, so that the probabilities associated with each operator are constant.

In order to better technical performance and finding better reduction results, especially of number of edges, we propose modify the algorithm and associate a deterministic parameter control that changes the mutation probability parameter along the process. It is necessary because the search process is characterized by exploration in the early generations and exploitation in the later stages. The lack of exploitation prevents the algorithm evolve and prevents best least be found.

IV. MOGAMAP2

The MogaMap2 is the second generation of the MogaMap [4]. The MogaMap is a hybrid mapping algorithm inspired by the traditional nondominated sorting genetic algorithm II (NSGA-II) [7] and that uses an evolution specific heuristics of the problem in order to reduce the search space and to drive the search to a promising region. It is an algorithm based in cuts selection. The metrics area flow [8], edge flow [5] and switching flow [9] are used to select the cuts of each node that are more useful to the desired goals. These metrics are remembered in equations 5, 6 and 7.

During this section we present the details evolving each step of the MogaMap2. This include from the form of individual representation until the detailed pseudo-code.

A. The individual representation

In MogaMap2, each individual, that represents one possible solution to the problem, is represented by an array of size *n*, where *n* is the number of nodes into the AIG, the array elements (genes) are arranged in topological order. The Fig. 6 illustrates an example of a network whose nodes *a* to *f* are primary inputs.

Each node in AIG has a quantity of valid cuts (called domain) and each cut has an integer number associated. For example, considering 3-LUTs, the possible cuts of each node in the network are shown in Table I. The node *w* has 3 valid cuts, thus, the domain of node *w* is 3. For the first valid cut *(u,v)* was attributed the integer 1, for the second one *(u,n,x)* was attributed the number 2 and so on.

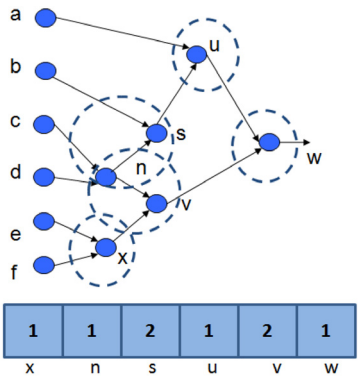

Figure 6. Example of network and individual representation

Table I. Possible cuts and domain for each node

Node	Domain	Possible Cuts	Integer number associated
x	1	(e,f)	1
n	1	(c,d)	1
s	2	(b,n)	1
		(b,c,d)	2
v	3	(n,x)	1
		(c,d,x)	2
		(n,e,f)	3
u	2	(a,s)	1
		(a,b,n)	2
w	3	(u,v)	1
		(u,n,x)	2
		(a,s,v)	3

In Fig. 6, the chosen cut for each node is represented by the dashed lines and the individual representation for this mapping is shown below the network. According to Fig. 6 and Table I, the chosen cut of node *w* was 1, therefore, the node *w* was mapped with the cut *(u,v)*.The node *v* was mapped with cut 2 *(c,d,x)*, the node *u* was mapped with cut 1 *(a,s)*, the node *s* was mapped with cut 2 *(b,c,d)* and the nodes *x* and *n*, that have domain 1, were mapped with cuts *(e,f)* and *(c,d)*.

A population is composed by *N* individuals. The Fig. 7 shows an example of population with five individuals for the previous network.

B. Recombination Operation

The recombination operation used in this work is the one-point crossover. After selecting two parents by binary tournament, the crossover point is drawn and the children are produced through the combining genetic material from the parents. The Fig. 8 shows an example of the crossover operator action. The decoding of the individuals representing the parents and children in the crossover operation is shown in Fig. 9.

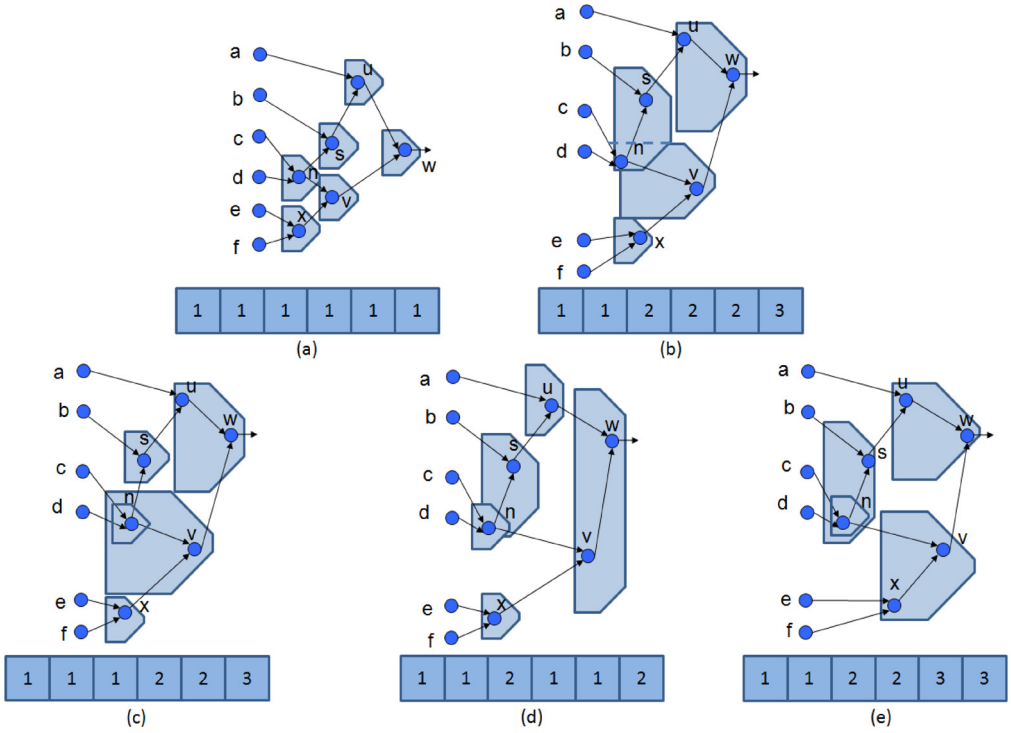

Figure 7. Example of population with five individuals

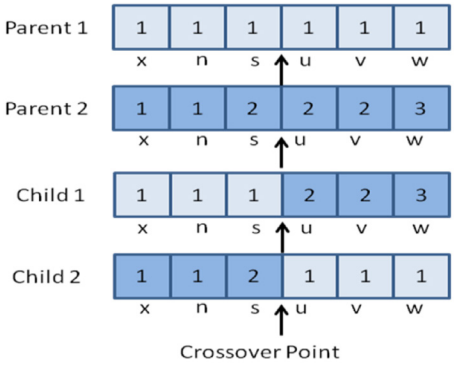

Figure 8. Example of one point crossover operator

In this Figure, it is possible to verify the effect of the combination of genetic material from the parents.

C. Mutation Operation

The mutation operation is responsible for changing the content of each gene, which means, changing the chosen cut as representative cut of each node. Typically, in the mutation process, each gene is visited and it is decided whether the gene will be changed and the new value assigned to it. For our approach, it is not necessary to visit all genes, only genes that make part of the current solution are visited. Genes that are encapsulated in cuts are not visited. For example, for the solution represented by the individual in Fig 7 (b), only the genes that represent the nodes w, s, v and x will be visited. The genes u and n are encapsulated and do not make part of the solution.

In this process, the nodes are visited in topological reverse order. If a node is visited, the inputs of its representative cut are also visited, until the primary inputs are reached. The Fig. 10 illustrates a mutation in a node and shows its effect in the mapping.

D. Objective Functions

The objective functions treated in this problem are area, number of edges and activity switching of the edges. The area (f_A) is the LUT count in the mapping, the number of connections (f_E) is the quantity of connections among the LUTs and the activity switching (f_S) is the summation of switching activities of all connections between LUTs. Therefore, each individual has an objective vector $f = (f_A, f_E, f_S)$.

During the process, all individuals are evaluated and the objective function is employed to determine the dominance levels.

E. Parent Selection

The parent selection is carried out using the binary tournament, similar to proposed in NSGA-II. In binary tournament, a pair of individuals is randomly chosen from the population. The chosen individuals are compared using the operator $(\geq_n)$. The individual which win the tournament will become one of parent.

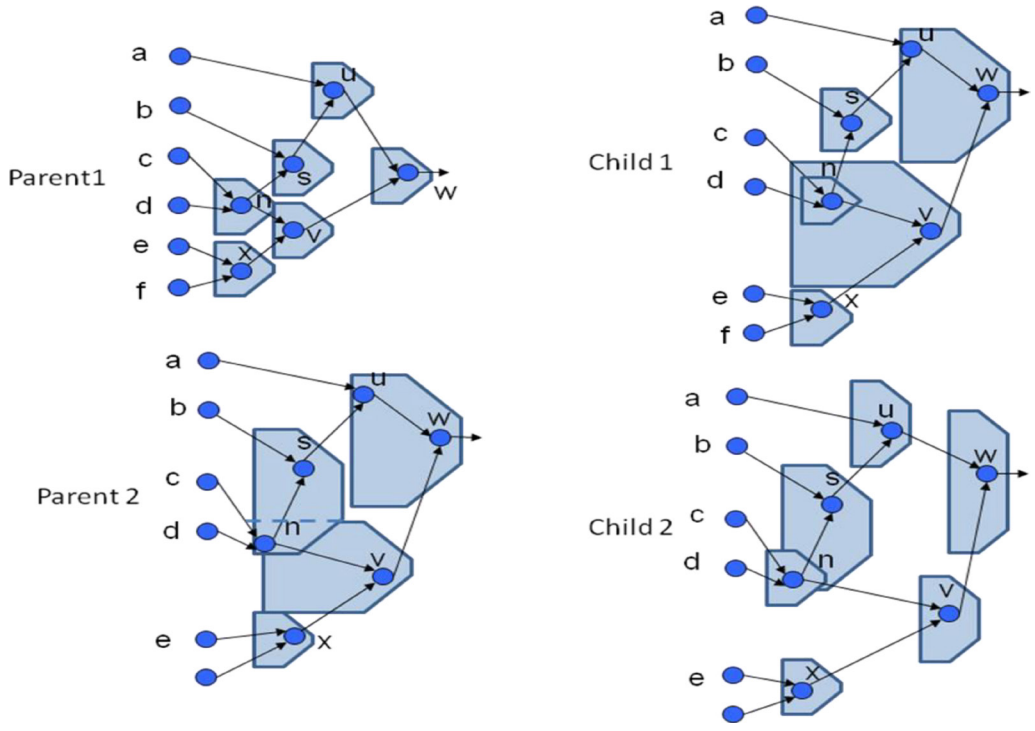

Figure 9. Decoding of parents and children in crossover operator

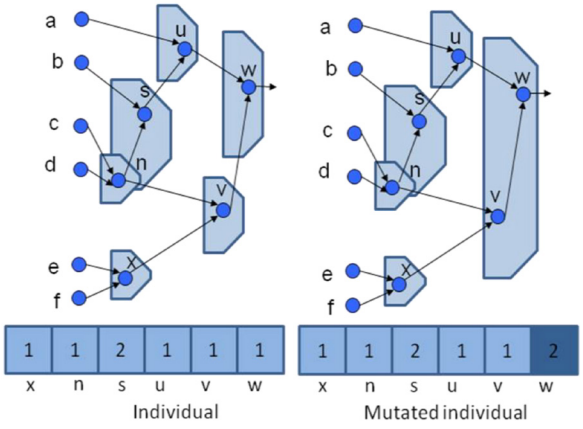

Figure 10. Mutation in a gene of the individual and its effect in the mapping

The process is repeated and the second parent is determined. The choices are performed with replacement, thus, in the same generation, the individual can be chosen more than once to be parent.

F. Survivor Selection

The criterion used in the survivor selection is determined by NSGA-II. According to this criterion, parents and offspring compete to compose the new population. The individuals are selected using the non-domination level and whenever necessary, the crowding distance is employed.

G. Specific knowledge about the problem

The complexity level of the mapping problem depends on the network structure, the network size and the number of LUT inputs. The last one influences directly in the number of possible cuts for each node. On the other hand, it is known that a subset of this cuts are not interesting from the standpoint of the expected metrics to the problem. Therefore, adding specific knowledge about the problem can be used in order to reduce the search space and lead the genetic algorithm to promising regions. The idea is evaluating the cuts according to their potentiality and invalidating those which are uninteresting.

We are interested in cuts with some characteristics. In order to evaluate the cuts, we use consolidated heuristics in the literature that allow us to measure these characteristics and pre-select the cuts, discarding those one which violate our constraints.

The first pre-selection is performed by excluding cuts whose depth exceed the minimum depth of the network. If a cut C is chosen to be a representative cut of the node i, then the minimum depth of C is defined as in (1).

$$MinimumDepth\,(C) = 1 + \max_{\forall Input(C)} (MinimumDepth\,(Input\,(C)) \quad (1)$$

The depth of a primary input node is zero and the minimum depth of a node i (non primary input)

is determined according to (2). In the other words, the minimum depth of a node *i* is the smallest depth among all the k-feasible cuts of the node *i*.

$$MinimumDepth\ (i) = \min_{\forall C \in cuts(i)} (MinimumDepth\ (C)) \quad (2)$$

The minimum depth of the network is the largest depth among all primary outputs. The depth metrics are determined during the cut-enumeration process.

Once the minimum depth of the network is determined, the network is traversed in topological reverse order and, for each node, is assigned a limit depth. The limit depth of a primary output is the minimum depth of the network, and the limit depth of the other nodes is determined according to (3). Cuts whose minimum depth meets the constraint of limit depth are validated, otherwise, they are discarded.

$$LimitDepth\ (i) = \min_{\forall Cli \in inputs\ (C)} (1 - LimitDepth(Node\ root\ of\ C)) \quad (3)$$

The second pre-selection is based on a weight assigned to each cut. The weight of a cut C rooted at a node *v* is determined according to (4).

$$Weight\ (C) = \alpha_M \bullet AreaFlow_n\ (C) + \beta_M \bullet EdgeFlow_n\ (C) + \gamma_M \bullet SwitchFlow_n\ (C) \quad (4)$$

Where $AreaFlow_n(C)$, $EdgeFlow_n(C)$ and $SwitchFlow_n(C)$ correspond to $AreaFlow(C)$[8], $EdgeFlow(C)$[5] and $SwitchFlow(C)$[9] normalized according to (5), (6) and (7).

$$AreaFlow_n\ (C) = \frac{AreaFlow\ (C)}{\max_{\forall C' \in cuts(v)} AreaFlow\ (C')} \quad (5)$$

$$EdgeFlow_n\ (C) = \frac{EdgeFlow\ (C)}{\max_{\forall C' \in cuts(v)} EdgeFlow\ (C')} \quad (6)$$

$$SwitchFlow_n\ (C) = \frac{SwitchFlow\ (C)}{\max_{\forall C' \in cuts(v)} SwitchFlow\ (C')} \quad (7)$$

The area flow is an extension of the area concept. This heuristic is applied to give a global view of how useful for the mapping is one cut choice. For a node *n*, whose representative cut is *C*, the definition of the area flow cost function is shown in (8):

$$AreaFlow\ (n) = \frac{Area\ (n) + \sum_i AreaFlow\ (Input_i\ (C))}{NFanouts\ (n)} \quad (8)$$

In (8), $Area(n)$ is the area cost of the LUT used to map the node n. This cost is zero if the node is a PI or one for the other ones. $Input_i(C)$ represents each input of *C* and $NFanouts(n)$ is the number of output edges of node *n* in the current mapping. The use of $NFanouts(n)$ in the denominator is fundamental to reduce duplication, since it takes into account the sharing and favors the choice of nodes with more fanouts to make part of the mapping.

Edge flow is a heuristic similar to area flow and it predicts the total number of pin-to-pin connections in the transitive fanin of a node. Minimizing the number of connections between the LUTs improves the routability, since reduces the number of wires during placement and routing.

The edge flow is defined in (9), where $Edge(n)$ is the number of inputs edges of the LUT used to map the representative cut of the node *n*.

$$EdgeFlow\ (n) = \frac{Edge(n) + \sum_i EdgeFlow\ (Input_i(C))}{NFanouts\ (n)} \quad (9)$$

The switching flow estimates the switching associated with the logic required to map the node *n* that has a cut *C* as representative cut. The switching flow is defined in (10). Where $Switch(n)$ is the estimated activity switching to the node *n*.

$$SwitchFlow\ (n) = \frac{Switch\ (n) + \sum_i SwitchFlow\ (Input_i\ (C))}{NFanouts\ (n)} \quad (10)$$

This second pre-selection is performed by determining the weight limit for each cut valid. Cuts of each node, whose weights are highest that the threshold L, will be invalidated. Let be *M* the average weights of all cuts of a node *i* and *m* the smallest weight among all cuts of *i*. Then L is:

$$L = \frac{(M - m)}{F} + m \quad (11)$$

The values of α_M, β_M e γ_M in (4) allow adjust the influence of area, amount of edges and power switching in the final mapping and *F* is a constant whose value has effect in the number of cuts validated for a node. The cut pre-selection process of a node can be seen in Fig. 11. In our example, the node has nine valid cuts,

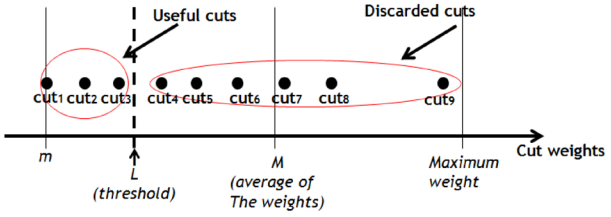

Figure 11. Cut pre-selection process

these cuts are ordered according to their weights. The threshold L is between the minimum weight (m) and the average (M). To this example, after pre-selection, the number of valid cuts was reduced to three.

The pre-selection process is dynamic and is executed every generation along with update of the flow measurements. The equations 8, 9 and 10 use the fanouts number in their denominator. However, the exact fanouts number can only be determined after the mapping, given that so me fanouts of the original network can be encapsulated within LUTs.

In order to estimate the fanout number of the nodes in the network, some models have been tested, and the model that showed best results was proposed in [8]. Thus, at the end of each generation, an individual of the population is selected as the reference solution and the fanout number of each node is estimated based on this solution and using the selected estimation model. After each generation, the estimated fanouts number gets closer to the actual value. According to the proposal, the estimated fanout number of a node v is defined as in (12):

$$NFanouts_{est}(v) = \frac{NFanouts'_{est}(v) + \alpha \cdot NFanouts\ (v)}{1+\alpha} \qquad (12)$$

Where $NFanouts'_{est}(v)$ is the estimation of fanouts number of the previous iteration, $NFanouts(v)$ is the fanouts number of the current iteration, and α is a constant that assumes values among 1.5 and 2.5.

H. Decoding

Decoding is the passage of genotype space (search space) to the phenotype space (space solutions). In this process, the individual represented by a vector, with its elements encoded in integer number, is translated to a mapped network. The first step of decoding is associating to each node a cut chosen as representative cut and the second one is traversing the network in reverse topological order defining the nodes that are part of the mapping.

I. Adjust of Mutation

The mutation has an important role in MogaMap2. From this operator, the best cuts are selected for each node and, the diversity of the population is ensured. The first generation of this algorithm (MogaMap) has used a fixed mutation probability. This choice can limit the potential of the mapping because genetic algorithms are characterized by exploration in the first generations and exploitation in the final generations. A high mutation probability during all the process may hinder the exploitation required for local searches in promising regions, and at the same time, a low mutation probability during the whole process can prevent proper exploration of the search space.

In order to find the best mutation probabilities for each stage of the algorithm, experiments were performed which measured the variation of area (LUTs count) and number of edges within intervals of 10 evolutionary cycles. The experiments used the MCNC benchmark and evaluated these variations for some values of mutation probabilities. For each circuit, the algorithm was run 10 times and the average result was obtained on executions. These results can be seen in Fig. 12 and Fig. 13. From the Fig. 12, we can see that for the first generations, the mutation probability of 30% is the more appropriate, since it presents greater reduction in the number of LUTs. Between 10 and 20 generations, the mutation probability of 10% presents greater reduction. We can also observe that for the range between 20 and 30 generations, the mutation probabilities of 30% and 20% start to present stagnancy and, the mutation probability of 10% is better. After 30 generations, the use of probabilities 30% and 20% is inadequate because

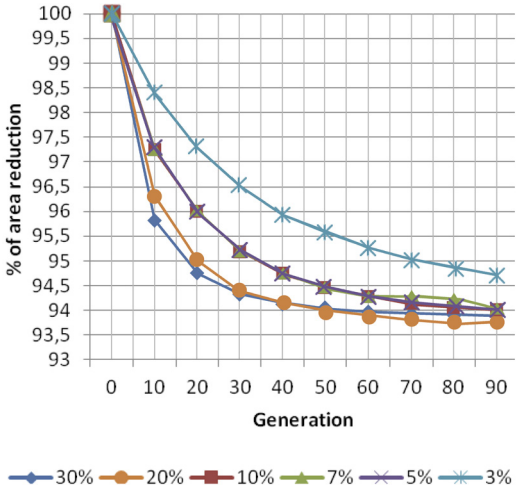

Figure 12. Percentage of area reduction per generation

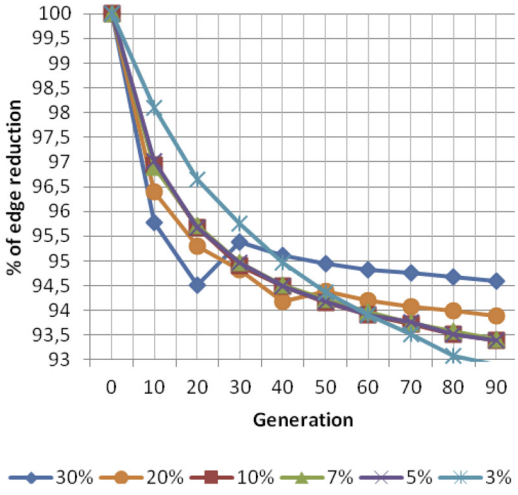

Figure 13. Percentage of edge reduction per generation

it has practically no reduction in number of LUTs. This is because the search space has been explored and lower mutation probabilities should be used for more refined searches around promising regions. At this phase the probabilities of 10%, 7% and 5% had larger reductions that 20% and 30%, but the probability that stands out, with greater reductions in the number of LUTs after 30 generations, is 3%.

With respect to the number of edges among LUTs (Fig. 13), we can conclude that for the first generations, as well as in LUT counts, we have also had more significant reductions to the mutation rate of 30%. From 10 generations, we noticed hegemony in the results that correspond to the use of lower mutation probability of 3%. Also, we realize that in some intervals there is an increase in the number of edges. The explanation for this increase comes from the fact that we are working with a multi-objective problem, where in some situations, opt for a solution that reduces a goal can lead to the increase of the other one. This occurs in the range between 20 and 30 generations when the mutation probability of 30% is used and between 40 and 50 generations, when the mutation probability of 20% is used.

The last conclusion we can draw from our experiment is that, regardless of the probability of mutation, over the generations, the goal of number of LUTs stabilizes much faster than the goal of number of edges. The explanation for this fact is that the number of possibilities of edges is much larger than the number of possibilities of LUTs, in other words, we can find solutions with the same LUTs count but several possibilities of edges count among these LUTs.

In MogaMap2, we set the mutation probability to 30% in the first 10 generations, 10% between 10 and 20 generations and 3% for other generations.

J. Core of MogaMap2

As in [4] we present the core of the MogaMap2 in Fig. 14. The first task is determining the switching probability for each node (line 2), the Lag-one Model [10] is used to determine it. After that, we use the cut-enumeration method to compute the K-feasible cuts of each node (line 3).

According to section III-G, we need to select the cuts that guarantee the optimum depth mappings, therefore, we determine the minimum level of each cut and the limit depth of each node (lines 4 to 9). The optimum depth of the mapping is the highest level among all the primary outputs (lines 10 to 14) and after define it, we can determine the limit depth of the nodes and we can invalidate cuts that do not meet depth constraint (line 15).

The next step is the second pre-selection (lines 16 to 23) as defined in section III-G. In this case, we determine the area flow, edge flow and switching flow

```
1:  Begin
2:  Determine_switching_probability(N);
3:  Cut_enumeration(N);
4:  for each node n in DFS order do
5:      for each cut of n do
6:          Determine_minimum_level();
7:      end for;
8:      depth_min(node) = minimum_level(cuts);
9:  end for;
10:     Optimum_depth = 0;
11:     for each PO node do
12:         if (depth_min(node) > Optimum_depth)
13:             Optimum_depth = depth_min(node);
14:     end for;
15:     Determine limit_depth(N, Optimum_depth);
16:     for each node n in DFS order do
17:     for each cut of n do
18:             Determine_flows(cut);
19:             Determine_weight(cut);
20:         end for;
21:         L = Determines_Weight_limit();
22:         Determine_valid_cuts(cuts, limit_depth(n), L);
23:     end for;
24:     P0 = Initialize_population();
25:     num_iter = 0;
26:     Fast_nondominated_sort(P);
27:     t = 0;
28:     while(num_iter < NUM_GENERATIONS) do
29:         |Qt |= 0;                    //offspring pop
30:         Pmutation = DetermineProbMutation(num_iter);
31:         while (|Qt| < POP_SIZE) do
32:             Select_parents_by_tournamet(Pt);
33:             Crossover (Pcrossover);
34:             Mutation (Pmutation);
35:             Qt = Qt U children;
36:         end while;
37:         Rt = Pt U Qt;                  //parents U offspring
38:         F = Fast_nondominated_sort(Rt);
39:         i = 1;
40          while (|Pt+1| < POP_SIZE) do
41:             Crowding_distance_assignment(Fi);
42:             Pt+1 = Pt+1 U Fi
43:             i = i + 1;
44:         end while;
45:         Sort(Pt+1, ≥n);          //order by non- dominance
46:         Pt+1 = Pt+1[0:POP_SIZE];
47:         Ind_Ref = Select_reference_individual( F1);
48:         Actualize_output_edges(N, Ind_Ref);
49:         Actualize_valid_cuts(N, Ind_Ref);
50:         t = t+ 1;
51:         num_iter= num_iter + 1;
52:     end while;
53:     Ind_Ref = Select_reference_individual( ( F1);
54:     Set_Mapping(Ind_Ref);
55: end;
```

Figure 14. PseudoCode of MogaMap

for each valid cut and determine the weight according to (4). The cuts of each node, whose weights are highest that the threshold L (11), will be invalidated.

At line 24, a initial population P_0 is created randomly. This population is evaluated and sorted according to non-domination (line 26). After that, the evolutionary cycle is started and runs for a number of generations (line 28). For each generation we set the mutation probability according to discussed in section III-I (line 30). In the generation t, the parent population P_t is used for selection, crossover and mutation to create the offspring Q_t (lines 31 to 36).

Since all of the children were created, the parent and the children population are combined (line 37) creating the set R_t. The population R_t is ordered according to non-domination (line 38). The new parent population P_{t+1} is formed by adding solutions from the first front till the size exceeds POP_SIZE (line 40 until 44). Thereafter, the solutions of the last accept front are sorted according to $\geq_n$ and the first POP_SIZE elements are picked (line 46).

From the first front, the reference individual is selected. This individual should be chosen according to the user preference. In our case we select the one that produces the smaller area mapping (line 47).

The mapping is completed when the preference solution is selected from the first front (line 53) and the mapped circuit is produced (line 54).

V. EXPERIMENTAL RESULTS

A. Mapping Results

In our experiments we use the MCNC benchmarks in order to determine the MogaMap2 performance. We compare three mappers: the wiremap (considered a baseline), MogaMap and MogaMap2.

During our experiments, we configured the MogaMap2 parameters as follows: K = 6, α_M = 0.5, β_M = 0.45, γ_M = 0.05, F = 40, $P_{crossover}$ = 90%, $p_{mutation}$ = 30% in the first 10 generations, 10% between 10 and 20 generations and 3% for other generations, POP_SIZE = 20 and the NUM_GENERATIONS = 50. The FPGA architecture used is composed by: CLBs with 10 BLEs. Each CLB has 33 input ports, 20 output ports and one clock signal. A BLE contains a 6 inputs LUT, a flip-flop and some combination logic. The experiments have considered the 45nm transistor technology.

The results are presented in Table II. The columns #LUTs and #edges show the number of LUTs and total number of pin-to-pin connections, respectively. The column t(s) shows runtime in seconds. The row Ratio1 shows the ratio of all results obtained from MogaMap and MogaMap2 to the results obtained from WireMap. Similarly, the Ratio2 shows the ratio of all results obtained from MogaMap2 to results obtained from MogaMap.

For setting parameters applied, the results lead to conclude that, in average, MogaMap is not able to reduce the number of LUTs significantly in comparison to baseline (only 1%). But it is able to reduce the number of edges in 7%. For the same setting, the number of LUTs has been reduced in almost 3% and the number of edges in almost 11% comparing MogaMap2 versus WireMap.

The experimental results also show that, according to ratio2, our algorithm reduces in average the number of LUTs in 2% and the number of edges in 4% in comparison to the first generation of the algorithm, the MogaMap.

Table II. Comparison of WireMap, MogaMap and MogaMap2

Benchmarks	WireMap			MogaMap			MogaMap2		
	#LUTs	#edges	t,s	#LUTs	#edges	t,s	#LUTs	#edges	t,s
ex5p	715	3667	2.59	711	3484	5.75	715	3427	6.70
apex4	896	4976	1.50	857	4147	6.51	863	4047	7.17
misex3	886	4523	2.30	860	4169	6.36	842	3996	6.88
alu4	934	4743	2.18	874	4265	5.80	833	3953	5.81
seq	1071	5657	2.13	1036	5053	6.80	1017	4845	7.53
apex2	1200	6202	2.42	1133	5455	6.34	1112	5210	9.84
des	1065	5024	3.28	1044	4811	8.13	994	4534	8.79
spla	2317	12764	6.44	2203	10817	21.24	2214	10543	18.55
ex1010	3260	16720	7.49	3105	15231	27.55	3109	14636	24.26
pdc	2713	15504	7.37	2706	13395	29.97	2715	13099	25.71
bigkey	575	2952	3.24	575	2952	5.28	575	2952	5.42
clma	4315	22757	6.88	4268	20862	50.15	4279	20421	48.11
diffeq	762	3575	2.36	749	3323	6.74	707	3146	.24
dsip	687	3400	2.68	689	3180	9.42	689	3180	5.65
tseng	748	3204	2.29	711	2902	5.29	715	2815	5.13
elliptic	1930	9157	4.59	2020	8898	11.73	1911	8500	13.00
frisc	1984	10098	6.34	1988	9803	20.53	1988	9688	17.18
S298	805	4244	2.56	883	4250	5.77	819	3805	7.99
S38417	3151	13254	7.78	3261	13286	15.20	3190	12999	21.89
S38584.1	2779	12269	5.21	2914	12620	26.06	2786	11817	34.15
GeoMean	**1348**	**6746**	**4**	**1333**	**6270**	**11**	**1308**	**6037**	**11**
Ratio1	**1**	**1**	**1**	**0.989**	**0.929**	**3.006**	**0.971**	**0.895**	**3.150**
Ratio2				**1**	**1**	**1**	**0.981**	**0.963**	**1.048**

Table III. Comparison between MogaMap2 and WireMap.

Benchmarks	MogaMap2/WireMap				
	#CLBs	W	LEN	f	EC
ex5p	1.02	0.88	0.97	0.99	1.10
apex4	0.95	0.89	0.92	1.04	0.84
misex3	0.94	0.86	1.03	0.98	0.90
alu4	0.98	1.11	1.08	1.07	0.93
seq	0.97	1.00	1.00	1.40	0.72
apex2	0.95	0.96	0.97	1.06	0.88
des	0.69	1.00	0.85	1.21	0.92
spla	0.93	0.94	0.84	1.04	0.86
ex1010	0.96	1.07	1.03	1.06	0.96
pdc	0.90	0.84	0.76	1.17	0.65
bigkey	1.00	1.00	1.00	1.00	1.01
clma	1.01	0.94	0.98	1.25	0.73
diffeq	1.04	1.00	1.14	1.00	1.15
dsip	1.01	0.92	1.03	1.07	0.77
tseng	1.02	1.15	0.88	1.37	0.90
elliptic	1.04	1.00	1.11	1.03	1.01
frisc	1.02	1.00	1.02	1.04	0.98
S298	0.96	1.16	1.29	0.91	1.23
S38417	1.02	1.07	1.06	0.92	1.05
S38584.1	1.03	0.95	1.01	1.00	0.98
GeoMean	0.97	0.98	0.99	1.07	0.92

B. Pos-Route Results

To evaluate the contribution of MogaMap2 to the quality of circuits post placement and routing, we run the CAD flow steps in the VTR framework [11] that integrates the tools: ODIN II that performs elaboration and synthesis on Verilog netlist; ABC, where we integrated the MogaMap2 and VPR 6.0 that performs clustering, placement and routing. The power estimative was performed by VersaPower [12].

In Table III we present the post routing results of number of CLBs (#CLBs), channel width (W), total wirelength (LEN), estimated frequency operation (f) and estimating energy consumption (EC). These results refer to ratio of the values obtained using the MogaMap2 and WireMap.

These experimental results lead to some observations: MogaMap2 achieved a 3% reduction in CLBs while improved the frequency operation in 7%, we cannot notice a significant reduction in the average of channel width and total wirelength, only 2% and 1% respectively, but the energy consumption estimated is reduced in 8%.

VI. CONCLUSIONS

This article presented MogaMap2, a second generation of Multi-objective Mapping Algorithm based on genetic algorithm. MogaMap2 differs from the state of the art in that it is a hybrid algorithm that uses practical heuristics to conduct the search to promising regions. As a result, comparing with WireMap, when targeting 6-LUTs and using a specific setting, a reduction of 10% in the average number of wires (or pin-to-pin connections) in the design is observed, with a reduction of 3% in LUT count. In comparison with MogaMap, a reduction of 2% and 4% in the LUT count and in the number of wires, respectively was found.

In post place-and-route analysis, the reduction of LUT counts was translated into reduction of CLB counts. Intuitively, after placement and routing, the reduced number of wires leads to reduction of capacitances and, consequently, increased the estimated frequency operation in 7%. Besides that, the estimated energy consumption was reduced in 8%.

We show that the deterministic control of the mutation probability made the MogaMap2 more efficient than MogaMap. We believe that the control of others parameters could become the algorithm still better. In future works we intend include the development of strategies in order to make the other parameters of the algorithm adaptive.

ACKNOWLEDGEMENTS

The authors would like to thank to CNPq (grants Universal/472317/2013-0), FACEPE, and UFPE for the financial support of this research.

REFERENCES

[1] D. Chen, J. Cong, C. Dong, L. He, F. Li, and C. C. Peng, “Technology mapping and clustering for FPGA architectures with dual supply voltages,” Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 29, no. 11, 2010, pp. 1709–1722.

[2] Yu-Yi Liang, Tien-Yu Kuo, Shao-Huan Wang, and Wai-Kei Mak, “ALMmap: Technology Mapping for FPGAs With Adaptive Logic Modules,” IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 31, no. 7, 2012, pp. 1134–1139.

[3] J. Luu, J. H. Anderson, and J. S. Rose, “Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect,” in Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays - FPGA ’11, 2011, p. 227-236.

[4] Souza, V.L.; Silva-Filho, A.G., “MogaMap and DynPack: Multi-Objective mapping and packing algorithms for optimization of area, performance and power consumption in FPGAs,” in Proceedings of 27th Symposium on Integrated Circuits and Systems Design (SBCCI), 2014, pp.1-6.

[5] S. Jang, B. Chan, K. Chung, and A. Mishchenko. “Wiremap: FPGA technology mapping for improved routability and enhanced LUT merging,” ACM Trans. on Reconfig. Tech. and Systems, vol. 2, 2009, pp. 1–24.

[6] I. I. Bucur, C. Stefanescu, A. Surpateanu, and N. Cupcea, “Power consideration in mapping LUT based FPGA circuits,” in Proceedings of IEEE 5th International Conference on Intelligent Computer Communication and Processing ICCP 2009, 2009, pp. 369–372

[7] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, “A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II,” Evolucionary Computation, IEEE Transictions on, vol 6, issue 2, 2002, pp. 182-197.

[8] V. Manohararajah, Stephen D. Brown, and Z. Vranesic. “Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping”. in Proceedings of the International Workshop on Logic and Synthesis, 2004, pp. 14-21.

[9] S. Jang, K. Chung, A. Mishchenko, and R. Brayton, “A power optimization toolbox for logic synthesis and mapping,” Proceedings of International Workshop on Logic and Synthesis, 2009, pp. 1-8.

[10] R. Marculescu, D. Marculescu, and M. Pedram, “Switching activity analysis considering spatiotemporal correlations,” in Proceedings of the IEEE/ACM international conference on Computer-aided design (ICCAD ‘94), 1994, pp. 294–299.

[11] J. Rose, J. Luu, C.W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, “The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing,” in Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA ‘12), 2012, pp. 77-86..

[12] J. Goeders and S. Wilton, “VersaPower: Power estimation for diverse FPGA architectures,” in Proceedings of the International Conference on Field-Programmable Technology (FPT), 2012 , pp. 229–234.