A New Approach to VPR Tool's FPGA Placement

Ednaldo Mariano Vasconcelos de Lima, Dr. Antônio Carlos Cavalcanti and Dr. Lucídio dos Anjos Formiga Cabral

ABSTRACT- Versatile Place and Route Tool (VPR) is the state of art in FPGA (Field Programmable Gate Array) placement and route academic tool. Concerning the placement stage, its great success in terms of quality is based on the use of Simulated Annealing and simple routines that generate random positions for swapping logical and I/O blocks. For complex circuits, however, the search of the best result can consume too much computing time. This work is focused on the development of three main points: a constructive heuristic for the initial placement, in order to accelerate the iterative phase of the VPR Tool; an alternative implementation of the VPR logical blocks swapping routines, reducing the random factor; a new way to calculate the initial temperature for the annealing phase. In comparison to the original VPR, this new implementation produces considerable reduction of the total computational cost, verified by the execution time at least 2X speed-up, without significant loses in the placement quality.

Keywords: FPGA, Placement, Simulated Annealing

INTRODUCTION

Field Programmable Gate Arrays (FPGAs), since their commercial introduction in the mid-80's, have revolutionized the way digital hardware has been designed and build. FPGAs are integrated circuits consisting of large number of programmable functional blocks and programmable interconnect network that allow, among other things, rapid complex circuits (re) implementation and time-to-market reduction. Two important goals of FPGA design optimization are reducing area and increasing speed, both closely related to the physical place where the blocks are located inside the FPGAs. So, by placing blocks which are functionally connected closer, there will be a reduction of the wiring required. On the other hand, balancing the wiring density on the FPGA will maximize the circuit speed, but can compromise the area optimization. Maximization of all those metrics simultaneously implies an exponential growth of the solution space; therefore, it may require prohibitive computation times.

We can summarize the design stages of FPGA circuits as follows: Synthesis and packaging of logical blocks, Placement and Routing .

Manuscript received July 22, 2007; revised August 8,2007 This work was supported by Laboratory of Systems and Integrated Circuits (LASIC) at Federal University of Paraíba, Brazil.

Lima, E. M. V. and Cavalcanti, A. C. are with the Departament of Computer Science, Federal University of Paraíba, Brazil (edinlima@gmail.com and caval@lasic.ufpb.br).

Cabral, L. A. F. is with the Departament of Statistics, Federal University of Paraíba, Brazil (lucidio@de.ufpb.br).

(i) Synthesis consists of converting logical descriptions or schematic diagrams, into a list of interconnected basic logic gates. Then, those gates are packed into actual FPGA logical blocks, generating another list. That netlist can be optimized, aiming either to reduce the total amount of blocks or to increase the overall speed. (ii) The placement stage is responsible to determine which logic block within an FPGA should implement each of the logic blocks required by the circuit, on which this article is focused, consists of determining the best position for each logical block in a FPGA design. (iii) The Routing stage determines which programmable switches will be closed in order to physically interconnect all the logical block nets. Usually, FPGA routings are represented through guided graphs, where each node represents a logical block connection pins and each arc represents a wire.

This paper presents the improvements on compiling time of FPGA placement, obtained by new heuristics added to VPR tool. Some improvements on area and/or delay occurred, but were not significant.

1. THE PLACEMENT PROBLEM

The VLSI cell placement problem involves placing a set of cells on a VLSI layout, given a netlist that provides the connectivity between each cell, and a library containing layout information for each type of cell. This layout information includes the width and height of the cell, the location of each pin, the presence of equivalent (internally connected) pins, and the possible presence of feed through paths within the cell.

FPGA placement is a NP-complete combinatorial optimization problem to determine the best position for each logical block in a FPGA design. A placement algorithm, must not only minimize the total wire length by grouping the interconnected cells, but must also ensure the routability of the circuit, by confining them in minimized areas and, simultaneously, taking care of balancing theirs distribution in order to avoid the saturation of the FPGA channels with lots of connections. The placement quality affects both the area and speed of circuits. In case of FPGAs where wiring resources are strongly limited, placement becomes a key issue for routing success. Placement is one of the most time consuming tasks of integrated circuits physical design.

For a long time, the most popular method for placement is known as *simulated annealing* and Versatile Place and Route (VPR) Tool [5] has been the state-of-art academic FPGA placement tool. Recently, there are a number of works that incorporate additional optimization to further improve the results of VPR. In [9], [10], logic replication is proposed as a post-processing step after placement by VPR to further improves circuit delay. Singh and Brown [11] proposed a post-placement retiming and showed how to modify VPR to make it "retiming aware". Chen and Cong [12] presented a

simultaneous placement and replication to minimize the longest path delay.

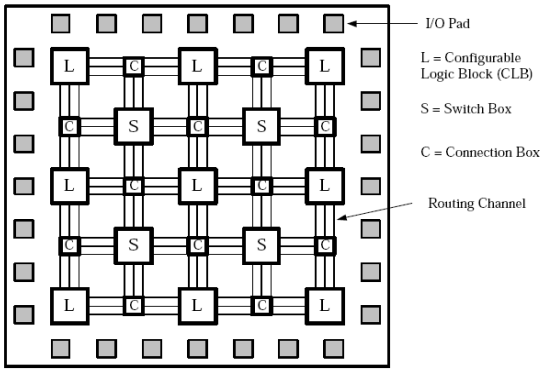

Figure 1: Island Cells Style FPGA

There are other initiatives using iterative heuristics like Thermodynamic Combinatorial Optimization [14], Tabu Search based placement heuristics[13] proposed by Emmert; and Maidee, who proposed a partition-based placement method[15].

To improve FPGA CAD tools performance, fast placement methods are critical. Currently, there are two main categories of placement algorithms: **construction algorithms** that, from a partial or incomplete placement, issue a complete placement; and **iterative algorithms**, that transform a complete placement into another one, improved, complete placement.

1.1. CONSTRUCTION ALGORITHMS

Growth of groups a**lgorithm** is a construction algorithm that makes use of a bottom-up method that selects components not yet placed and incorporates them it a partial positioning. The election of components is made on the basis of the already placed components, providing a local optimization. This algorithm is of easy implementation, has low computational complexity - $O(n^2)$, but they produce low quality results [2].

Partitioning-based placement algorithm [9] has a top-down approach, or the opposite of the precedent. It considers the circuit as a whole and then proceeds the partitioning of it, recursively, until each part is composed by a single component, when the placement is considered done. The major problem here is that the optimum partitioning search is NP-Complete; however, good heuristics have been developed to attack this problem, as is the case of mincut [3], of easy implementation and reduced computational cost.

1.2. ITERATIVE ALGORITHMS

The objective of iterative algorithms is to optimize a complete positioning already existing, normally gotten from construction algorithms. In this class we can mention:

Permutation of Pairs algorithm: each component is selected and swapped with another one. If that swap results in a placement improvement, then it is accepted as final, else it is discarded. The complexity of this algorithm type is $O(n^2)$[2].

Disconnected Sets algorithm, developed for Steinberg [2], selects the components, dividing them in sets that do not have common nets. Thus, the placement of each set is individually optimized, without tacking into account the other sets. This algorithm produces a number of permutations far lesser than in the permutation of pairs algorithm, since the sets possess a lesser number of components than the complete circuit. Experiments with this algorithm do not present good results, since in circuits with densely linked components – typical case in VLSI circuits - it is very complex to partition the circuit in disconnected sets.

Simulated Annealing algorithm [4][5][18]**:** differently of the two above, that only accept a given intermediate positioning in the case of having an improvement, limiting them to a local optimization, this metaheuristics tries to solve this problem through out the use of a search algorithm to find the configuration of lower energy of a confined set of molecules. Its name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

```
Procedure SA(f( ),N( ), α, SAmax, T0, s)
 s*← s; {better solution until now}
 IterT ← 0;{# iterations at temperature T}
  T ← T0;              {current temperature}
  while (T > 0) do
    while (IterT < SAmax) do
      IterT ← IterT + 1;
      Generate a neighbor s' ∈ N(s);
      Δ = f(s0) - f(s);
      if(Δ < 0) then  s ← s';
if (f(s') < f(s*)) then
   s* ← s';
else
   take x ∈ [0,1];
   if(x < e^(-Δ/T) ) then
      s ← s';
   end if;
end if;
     end while;
     T ← α×T;
    IterT ← 0;
   end while;
   s ← s*;
   Return s;
end SA;
```

Table 1 – Simulated annealing algorithm

2. PLACEMENT BY VPR

In order to assist the understanding of the implementation presented in this work, it will be described, in general lines, the execution of placement in the VPR tool.

In the old versions at VPR, the cost function, showed at (1), is focused only in wire length and penalizes placements which require more routing in areas of the FPGA that have narrower channels. All the results in this paper are obtained with FPGAs in which all channels have the same capacity, hence C_{av} is a constant and the linear congestion cost function reduces to a bounding box cost function resulting in wire cost how showed at equation (2). The last version of VPR (4.30) implements a new cost function considering wirelength-based driven

placement and timing-driven based placement, like proposed by Maquardt [16], according to showed at (3).

$$\cos t = \sum_{i=1}^{N_{nets}} q(i)\left[\frac{bb_x(i)}{C_{av,x}(i)^{\beta}} + \frac{bb_y(i)}{C_{av,y}(i)^{\beta}}\right] \quad (1)$$

$$wiring\_\cos t = \sum_{i=1}^{N_{nets}} q(i)\left[bb_x(i) + bb_y(i)\right] \quad (2)$$

In timing-driven placement VPR use a normalized cost function that depends on the change in timing cost and wiring cost. It uses a trade-off variable called λ to determine how much weight to give each component.

$$\text{Timing\_Cost}(i,j) = \text{Delay}_{(i,j)} \cdot (\text{Critically}_{(i,j)})^{\theta} \quad (3)$$

θ = critically expoent

$$\Delta\text{Cost} = \lambda\,(\Delta\text{timing\_cost} / \text{prev\_timing\_cost}) + (1-\lambda)(\Delta\text{wiring\_cost} / \text{prev\_wiring\_cost}) \quad (4)$$

The λ factor adjusts the weight of the two components. If λ = 1 then we have an algorithm that focuses only on timing, but ignores wire-length minimization. If λ = 0, then we have an algorithm that focuses only on minimizing wire-length. The default (adopted) value on VPR is λ=0.5.

As described in [4] and [5], the algorithm starts with a random initial positioning. The initial temperature is defined through the execution of N_{blocks} moves (N_{blocks} = # of CLBs + I/Os). The standard deviation of the cost of this N_{blocks} moves, multiplied for twenty, defines the initial temperature that will allow any movement to be accepted at anneal start. After that, inner_num x $(N_{blocks})^{1.33}$ moves are evaluated at each temperature. The default value of inner_num is 10. This default number can be overridden on the command line; however, to allow different CPU time or placement quality tradeoffs. At iteration, a block (CLB or IO) and a position of block are randomly chosen; that position will be internal if the block is a CLB. If it is an IO, the position will be peripheral. The number of moves evaluated by simulated annealing at each temperature is quite large. The evaluation of a move may result in three cases: (1) a block is moved to a new (empty) position; (2) two blocks are swapped; (3) the move is rejected.

When the temperature is so high that almost any move is accepted, we are essentially moving randomly from one placement to another and little improvement in cost is obtained. Conversely, if very few moves are being accepted (due to the temperature being low and the current placement being of fairly high quality), there is also little improvement in cost. Then, a new temperature update schedule, which increases the amount of time, spent at temperatures where a significant fraction of, but not all, moves are being accepted.

Fraction of moves accepted	*Alfa*
$R_{accept} > 0{,}96$	0,5
$0{,}8 < R_{accept} < 0{,}96$	0,9
$0{,}15 < R_{accept} < 0{,}8$	0,95
$R_{accept} < 0{,}15$	0,8

Table 2 – temperature update schedule

A new temperature is computed as $T_{new} = \alpha * T_{old}$, where the value of α depends on the fraction of attempted moves that were accepted (R_{accept}) at Told, as shown at Table 2.

Another important factor in this process is the variable D_{limit} that defines the range of search for the destination positions. This value is initially defined for the whole chip and is updated whenever the temperature is calculated by:

$$D_{limit\_new} = D_{limit\_old}\,(1 - 0{,}44 + R_{accept\_old}) \quad (5)$$

and then clamped to the range: $1 \leq D_{limit} \leq M$ (maximum dimension of chip)

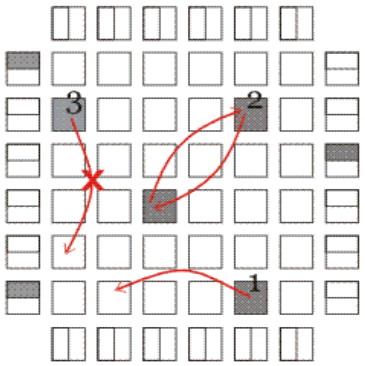

Figure 2: VPR moves

This way, at the beginning of anneal, the area where the blocks are swapped concerns the whole chip extension; as the temperature decreases, at intermediate stages, this area goes shrinking, becoming 1 at the final stage, when the temperature is the lowest.

Finally, the anneal is finished when

$$T < 0.005 * \text{Cost} / N_{nets}. \quad (6)$$

The movement of a logic block will always affect at least one net. When the temperature is less than a small fraction of the average cost of a net, it is unlikely that any move that results in a cost increase will be accepted, so we terminate the anneal.

3. IMPROVING PLACEMENT ON VPR

In this section it will be described the actions proposed to improve the placement stage present on VPR tool, achieve an execution time speedup at least 2X in comparation with VPR and without to lose quality in results.

As verified in the analysis and description of the original VPR placement algorithm, it is based on simulated annealing heuristic and use a random routine to choice the blocks to swap when trying improves the placement cost. Neither the choice of the source block to swap nor the choice of the position to locate it requires any complex routine; the only required routine is to generate a random coordinates. This routine is showed below.

Despite the good results, the necessary number of movements for the attainment of good quality results is extremely high and exponentially grows with the amount of logical blocks to be placed. The most important part of the VPR placement routine is the function shown at Table 3, called TRY_SWAP() and is responsible for the choice of blocks and positions to try to swap.

```
Procedure TRY_SWAP
  b_from, b_to {source and destination block }
  x_to, y_to        {coordinates at destination position}
  T        {current temperature}
  choice b_from  { draw b_from }
  choice x_to e y_to { draw a valid cell  }
  if position(x_to,y_to) empty then
     move b_from to position( x_to,y_to)
  else
     b_to = block(x_to,y_to)
     swap (b_from , b_to)
  endif
   calculate cost variation (ΔC)
   Take x ∈ [0,1]; { draw x value}
   if (x < e^(-ΔC/T) ) then
    return move accept;
  else
     move is rejected (undo move);
    return move rejected;
  endif
end TRY_SWAP
```

Table 3 – Try_swap function

For each temperature value, and depending of inner_num value, at each iteration the routine try_swap () is called, a block of origin and a position of destination is randomly chosen until either the amount of movements has reached its limit or the net temperature has reached its minimum limit.

The proposal of this work is modifying tree points in VPR tool. (i) First, replace the random initial placement by a constructive heuristic for initial placement; (ii) a new function to calculate the initial temperature to anneal, (iii) modify the TRY_SWAP routine to minimize the random on choice at destination position to place a selected block.

How to purposed by Banerjee [8], in a k x k array L, instead of randomly placing the entire netlist of CLBs and IOBs, are placed only the primary outputs randomly along the periphery of L. For the given circuit specified as a netlist, let us define a directed graph D = <V,E>, where V = { v | v is either a CLB or an IOB} and E = { $<v_i,v_j>$ | $v_i \in$ fanin(v_j) and $v_j \in$ fanout(v_i)}.

For each of the primary outputs present in the netlist a cone is defined. A cone f_i of O_i is the set consisting of O_i and all its predecessors [17]. In other words, f_i = cone(O_i) = {u | ∃ a simple directed path from u to O_i D}. The apex of the cone is the primary output O_i itself and let its level l be 0. We transverse the fan-in blocks of every block at previous level in breadth-first manner, till we find no new CLBs or primary inputs for the cone f_i . Breadth-first transversal of the cone results in a tree structure with O_i as root.

Thus when placed all CLBs and input blocks on O_i root neighbor. This gives the initial placement configuration for the technology-mapped netlist specified, as input to an iterative procedure for further improvement in the placement configuration. In comparation with VPR, the initial wiring-cost was improved in average fifty percent. Hence, to further improve the placement, an ultra-low temperature simulated annealing is executed on this initial placement to obtain the final placement configuration.

Then, is necessary a new manner to calculate the initial temperature to anneal. Because, on the old form, the execution of N (N = # CLBs + #IOBs) moves on high temperature, the good initial placement is so much perturbed and deteriorated.

```
Procedure  Modified_TRY_SWAP
   define b_from; {source block }
   define, b_to;          {destination block }
   define x_to, y_to{destination coord position}
   define T          {current temperature}
   define FACTOR {depends on the temperature }
   choice b_from { random value }
   choice x_to and y_to { random cell }
  do
     if position(x_to, y_to) is empty then
          move b_from to position(x_to,y_to)
     else
          b_to = block at position(x_to,y_to)
          swap( b_from, b_to)
     endif
     calculate  cost variable (Δ)
     Take x ∈ [0,1];
     if (x < e^(-Δ/T) ) then
       return accept
      else
         undo move
         if (b_from == CLB and  T <  10
                and total moves < FACTOR)
                   b_to = search neighbor to b_to
          endif
      endif
      enddo ( move not accept
                and total moves < FATOR
               and  b_from=CLB)
      return rejected
end Modified_TRY_SWAP
```

Table 4 – Modified TRY_SWAP() function

A new form to calculate the temperature is trying moving all blocks, one by one, with zero temperature. The initial temperature then is defined by t_init = 0.035φ to wirelength-driven placement, and by t_init = 54.05 φ/ num_blocks to timing-driven placement (φ = standard deviation of costs over N swaps and num_blocks = # CLBs + #IOBs, the values 0.035 and 54.05 was defined to force the same conditions at VPR).

On the TRY_SWAP routine, a variable FACTOR limits a number of attempts to swap using a some source block, whose for each temperature range there are specific values defined empirically.

It is defined that from a given temperature, at each iteration, after the origin block and the destination position are chosen, if a movement is not accepted, a new movement is attempted, keeping the origin block and choosing a new position of destination. Then, another movement can be tested without the need of choosing a new origin block. If the initial movement is not accepted and the origin block is a CLB, then an empty position at the destination block immediate neighborhood is searched. In case of success, an attempt to swap the origin block to this position is made; conversely, any of the neighbor blocks position can be used to perform the same attempt. This procedure is repeated until a movement is accepted or the limit of attempts defined through the variable FACTOR is reached.

Temperature (t)	FACTOR value
t > 50.0	2
2.0 ≤ **t** < 50.0	1
0.005 ≤ **t** < 2.0	4
0.0025 ≤ **t** < 0.005	2
t ≤ 0.0025	1

Table 5 : FACTOR range

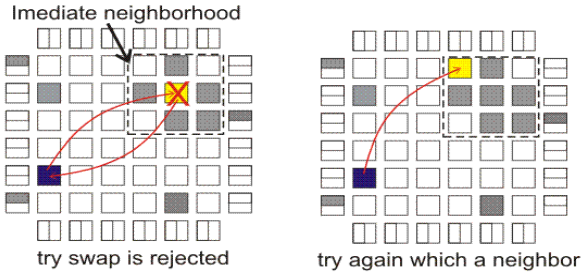

Figure 3 : search in neighborhood

Finally, a new default number of moves evaluated at each temperature is defined by $mov\_lim=\eta.10.(N_{blocks})^{1.33}$. In case of wirelength-driven placement, $\eta = 1/3$ and, in case of timing-driven placement, $\eta = ¾$, reducing the number of moves per temperature and speed up placement without prejudice the final result. The loss in quality of result is recompensed by gain provided by heuristics showed above.

4. EXPERIMENTAL RESULTS

The implementation and corresponding tests discussed in this article had been carried out in a microcomputer HP Pavilion zv6000, with AMD Athlon 64 Processor, 756MB of memory RAM and operational system Linux Fedora 5 - 64. The results is obtained by implementing 20 MCNC benchmark circuits [19], presented at Table 6, in the FPGA architecture technology-mapped to cells enclosing one four inputs LUT and one flip-flop, in according to [5].

Circuit	# CLBs	# I/O	# nets	FPGA Size
clma	8383	144	8445	92x92
s38417	6406	135	6435	81x81
s38584	6447	342	6486	81x81
ex1010	4598	20	4608	68x68
pdc	4575	56	4591	68x68
spla	3690	62	3706	61x61
elliptic	3604	245	3736	61x61
frisc	3556	136	3576	60x60
bigkey	1707	426	1937	54x54
apex2	1878	41	1916	44x44
dsip	1370	426	1600	54x54
seq	1750	76	1791	42x42
s298	1931	10	1935	44x44
diffeq	1497	103	1562	39x39
alu4	1522	22	1536	40x40
misex3	1397	28	1411	38x38
apex4	1262	28	1271	36x36
ex5p	1064	71	1072	33x33
tseng	1047	174	1100	33x33
e64-4lut	274	130	339	17x17

Table 6 – MCNC benchmark circuits

The comparison between VPR (version 4.30) and this approach denominated Modified-VPR (M-VPR) presented in table 7, show the relative improvement of M-VPR with respect to VPR on wirelength-driven placement, and in table 8, is showed the measures on timing-driven placement.

As shown at Tables and figures 7 and 8, the processing time reduction average is 53% is wire-driven case, and 56% in time-driven case, without compromising significantly the quality of the final results, and even improving it in some cases. In all tables are considered the wire length cost (Bounding-box cost).

	VPR			M-VPR		
	BB-Cost		Time (s)	BB-Cost		Time (s)
	Init	Final		Init	Final	
clma	7979	138	**1035**	3676	1398	**707**
s3841	5791	672	**874**	1966	637	**433**
s3858	5597	658	**656**	1807	641	**438**
ex101	3326	655	**361**	2094	654	**245**
pdc	3249	898	**357**	1581	873	**185**
spla	2370	594	**303**	1148	611	**126**
ellipti	2291	457	**239**	986	455	**158**
frisc	2284	516	**242**	1081	523	**163**
bigkey	1093	186	**75**	335	186	**46**
apex2	902	270	**85**	529	272	**53**
dsip	889	170	**56**	322	176	**34**
seq	796	248	**92**	419	255	**28**
s298	750	204	**77**	319	209	**26**
diffeq	670	146	**60**	328	145	**39**
alu4	607	190	**57**	316	190	**28**
misex	578	190	**54**	308	192	**26**
apex4	510	179	**46**	322	180	**28**
ex5p	424	162	**38**	276	164	**19**
tseng	410	92	**43**	185	93	**20**
e64-	73	29	**7**	54	29	**3**
average	2029	395	**238**	903	394	**140**

Table 7 - benchmark results in wirelength-driven placement

	VPR			M-VPR		
	BB-Cost		Time (s)	BB-Cost		Time (s)
	Init	Final		Init	Final	
clma	7964	150	**2397**	3730	1555	**1207**
s3841	5790	702	**1360**	1983	710	**760**
s3858	5626	673	**1350**	1823	677	**561**
ex101	3347	683	**841**	2129	689	**394**
pdc	3254	944	**876**	1617	945	**463**
spla	2350	628	**597**	1179	637	**360**
ellipti	2283	511	**565**	992	511	**281**
frisc	2257	587	**546**	1109	607	**315**
bigkey	1108	214	**163**	336	216	**92**
apex2	912	279	**177**	533	282	**95**
dsip	906	202	**125**	324	203	**72**
seq	797	263	**157**	424	261	**68**
s298	769	228	**174**	328	227	**84**
diffeq	658	161	**118**	336	160	**66**
alu4	614	200	**112**	317	200	**47**
misex	575	197	**101**	309	199	**43**
apex4	503	196	**89**	329	198	**42**
ex5p	423	180	**71**	279	179	**38**
tseng	418	104	**67**	190	104	**41**
e64-	73	30	**11**	54	30	**6**
average	2031	425	**495**	916	430	**252**

Table 8 - benchmark results in timing-driven placement

In next figures, are present the comparative at execution time to VPR and M-VPR .

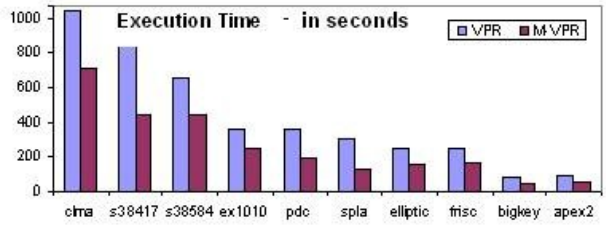

Figure 4: - benchmark results wirelength-driven placement

Figure 5: - benchmark results wirelength-driven placement

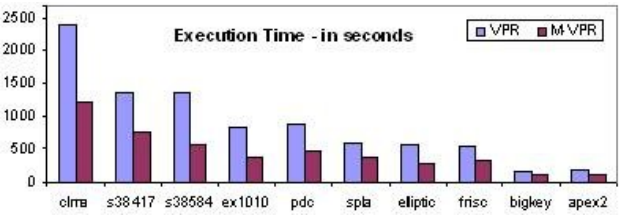

Figure 6: benchmark results in timing-driven placement

Figure 7: benchmark results in timing-driven placement

CONCLUSION AND FUTURE PROPOSAL

In this article, only some simple changes in the heuristics of VPR placement routines were considered. In this approach is not necessary to make any changes on the input netlist. A constructive initial placement apparently is very promising, as demonstrated by the results. Those changes can provide considerable gains in the processing time.

Since the initial objective has been reached we think this approach can be add to use clusters to generate a ultra-fast placement. We also intend to implement, in future works, changes on swap routines combining the strategies presented here with Tabu search [13] and change the simulated annealing by Thermodynamic Combinatorial Optimization [14].

REFERENCES

[l] Sait, S. M.,Youssef, H. "VLSI Physical Design Automation – Theory and Practice". IEEE Press, New York, 1995.

[2] Preas, B., Lorenzetti, M. "Physical Design Automation of VLSI Systems". Benjamin/Cummings Publishing Company, 1988.

[3] Kernighan, B. W. & Lin, S., "An Efficient heuristic produce for partitioning graphs". Bell System Technical journal. Vol 49, n. 2, pp. 291-307, February 1970.

[4] Betz V., Rose J., Marquardt A., "Architecture and CAD for Deep-Submicron FPGAs". Kluwer Academic Publishers, 1999.

[5] Betz V., Rose J., Marquardt A., "VPR: A new Packing, Placement and Routing Tool for FPGA research". In International Workshop on Field Programmable Logic and Applications, 1997.

[6] Júnior, L. F. L. "Pré-Posicionador de células em circuitos VLSI Standard-cells". Master's degree dissertation presented at UFPB, Campina Grande – PB - Brazil, 1994.

[7] Haldar M., Nayak A., Choudhary A., Banerjee P. "Parallel Algorithms For FPGA Placement", Northwestern University, 2000.

[8] Banerjee, P., "Accelerators for fpga placement," in The 4th Annual Inter Research Institute Student Seminar in Computer Science", April 2005.

[9] G. Beraudo and J. Lillis, "Timing optimization of FPGA Placements by Logic Replication", in proc. of Design Automation Conf., pp. 196-201, 2003.

[10] M. Hrkic, J. Lillis and G. Beraudo, "An approach to Placement-Coupled Logic Replications", in proc. of Design Automation Conf., pp. 711-716, 2004.

[11] D.P. Singh and S.D. Brown, "Integrated Retiming and Placement for Field Programmable Gate Arrays", in proc. of International Symp. On Field-Programmable Gate Array, pp. 67-76, 2002.

[12] G. Chen and J. Cong, "Simultaneous Timing-driven Placement and Duplication", in proc. of International Symp. On Field-Programmable Gate Array, pp. 51-59, 2005.

[13] J.M. Emmert and D.K. Bhatia, "Tabu Search: Ultra-Fast Placement for FPGAs", in 9th Intl. Workshop on Field Programmable Logic, pp. 81-90, 1999.

[14] J.D. Vicente, J. Lanchares, R. Hermida, "Annealing Placement by Thermodynamic Combinatorial Optimization", ACM Trans. On Design Automation of Electronic systems, vol 9, no. 3, pp. 54-60, 2004.

[15] P. Maidee, C. Ababei and K. Bazargan, "Fast timing-driven Partitioning-based Placement for island style FPGAs", Design Automation Conference, June 2003.

[16] A. Maquardt, V.Betz and J. Rose, "Timing-driven Placement for FPGAs", In ACM/SIGDA Int. Symp. on FPGAs, pages 203-213, 2000.

[17] A.Mathur and C.L.Liu, "Compression-Relaxation: A New Approach to Timing-Driven Placement for Regular Architectures", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 16, Issue 6, pp. 597-608, June 1997.

[18] Zhuo Y., Li H., Zhou Q., Cai Y., Hong X., "New timing and routability driven placement algorithms for FPGA synthesis" In: ACM Great Lakes Symposium on VLSI 2007, pp 570-575, 2007.

[19] http://www.eecg.toronto.edu/~vaughn/vpr/download.html