CONSTRUCTIVE AND LOCAL SEARCH HEURISTIC TECHNIQUES FOR FPGA PLACEMENT

Xiaojun Bao and Shawki Areibi

School of Engineering
University of Guelph
CANADA N1G 2W1
E-mail: {xbao, sareibi}@uoguelph.ca

ABSTRACT

The logic capacity of FPGAs has increased so rapidly in the last decade (up to 40-million gates) that it will take a considerable amount of time for users to synthesize and compile these circuits. These prohibitively long compile times may adversely affect instant manufacturability of FPGAs and become intolerable to users seeking very high speed compile. This paper presents two novel placement heuristic algorithms that significantly reduce the amount of computation time required to achieve acceptable-quality placements. The first algorithm is a Cluster Seed Search technique (CSS) that is considered to be a constructive based method and can be implemented in trivial time compared with other placement algorithms. The second algorithm is an enhancement of local search and is implemented as a Simple Local Search (SLS) and Immediate Neighborhood Local Search (INLS). Both techniques achieve reasonably good solutions quickly.

1. INTRODUCTION

Field-Programmable Gate Arrays(FPGAs) have become a popular way to realize digital systems because of their dramatic reduction of turn-around time and start-up cost compared with traditional Application-Specific Integrated Circuits (ASICS). Today FPGAs, as one of the most popular VLSI design styles, could lose their time-to-market advantage and capacity, since users don't have patience to wait for long periods of time to compile their design. Therefore users are willing to accept the decrease in quality of final results with less FPGA compiling time.

The focus of this paper is on the placement phase in CAD compile process. We present two novel placement heuristic algorithms that significantly reduce the amount of computation time required to achieve acceptable-quality placements. The first algorithm is Cluster Seed Search (CSS), a constructive based method that can be implemented in trivial time. In the common VLSI cell placement, constructive placement algorithms are generally based on primitive connectivity [1]. But in FPGA placement, CCS uses the fanout number criteria to select the best block and create an improved initial and legal placement solution.

The second algorithm is an enhancement of local search and is implemented in two different ways. The first is implemented as a Simple Local Search (SLS) which uses the simplest iterative improvement strategy. SLS attempts to achieve the reduction in wire-length cost by swapping blocks in a window which limits the swapping region. Initially the window is large, and as the heuristic progresses the window shrinks in size. Limiting the scope of swap within the region of the original block position gives superior results compared to unrestricted moves [2]. Local search is also implemented as an Immediate Neighbor Local Search (INLS). This technique can achieve suboptimal solutions in a very short period of time by only swapping the adjacent blocks around the selected blocks.

The rest of this paper is organized as follows: Section 2 summarizes the previous work done so far on FPGA placement. Section 3 describes our new fast placement algorithms, CCS, SLS and INLS. Section 4 presents the experimental results. Conclusions and future work are presented in Section 5.

2. BACKGROUND

Most CAD tools for FPGA development pay close attention to the placement stage to achieve as efficient a circuit mapping as possible, which is critical to the routing phase that follows. Since FPGA placement is an NP-hard combinatorial optimization problem [1], the heuristic techniques in CAD tools have to take a reasonable amount of time to produce a good placement solution.

The optimization techniques for FPGA placement can be classified into two categories: min-cut(partitioning-based) algorithms [3, 4] and iterative improvement algorithms [5, 6]. Min-cut or partitioning-based placement algorithms can run in a comparatively short time compared with iterative improvement algorithms. They usually produce high quality results using a divide-and-conquer strategy to reduce the problem space by repeatedly partitioning the problem into subproblems. In such algorithms, a circuit is recursively divided, and the blocks with high connectivity are left in one of the partitions by minimizing the number of nets crossing. This partitioning process repeats until finally the partition comes to a small size. Highly connected blocks are therefore grouped in the same partition.

Iterative improvement approaches, such as Simulated Annealing placement algorithms [5], are used to obtain very high quality placement but suffer from long computation times. Usually, these algorithms start with a random initial solution and attempt to find an improvement by seeking small perturbation to the placement that results in good solution. In order to explore the larger solution space, iterative improvement algorithms require a larger number of iterations to reach an optimal solution at the expence of large CPU time [7].

3. METHODOLOGY

Most heuristic algorithms for FPGA placement are capable of achieving good results at the expence of long runtime. Nowadays the time issue is becoming more important in the FPGA design as the technology shrinks. Two novel heuristic algorithms have been developed to solve the FPGA placement problem in a short period of time.

3.1. Cluster Seed Search

Cluster Seed Search (CCS) is a constructive heuristic method which is used to build an initial/legal placement. In the common VLSI cell placement paradigm, constructive placement algorithms are generally based on primitive connectivity rules [1]. Since the number of input and output pins of each logic block is fixed, the connectivity rules for common VLSI design do not work properly in FPGA design. CSS uses the fanout number criteria to select the best block and creates an improved initial and legal placement solution. A logic block with high fanout indicates that the block belongs to a net which has more terminals. Moving these logic blocks with high fanout together attempts to shrink the bounding box of the net containing more terminals with higher probability.

```
1.  Seed = RandomSelectSeed();
    /*randomly pick up a block as a seed at the start*/
2.  SetLocationOfSeed();
    /*place random seed at the first position of FPGA*/
3.  While(Initial Solution Not Complete)     /* start of loop*/
4.  {       CreateListOfFanoutNumber(Seed);
              /*create the list of fanout number of Blocks connected to the seed*/
5.          Seed = SelectBestBlock();
            /*select the block with the highest fanout number as the next seed*/
6.          SetLocationOfSeed();
            /*place current seed at the location close to the previously placed seed*/
7.  }   /*end of loop*/
8.      /* get the improved initial solution*/
```

Fig. 1. Pseudo-Code for CSS

The Pseudo-code for CSS is shown in Figure 1. Typically, a seed block (a logic block) is selected randomly and placed in the FPGA layout. Next, a block is chosen from the rest of unplaced blocks which are connected to the previously placed seed block according to their fanout. This block is placed at a vacant location closest to the previous chosen seed, such that the wire-length is minimized. The current placed block becomes the next candidate seed. The process is repeated until an improved initial and legal solution is constructed.

3.2. Simple Local Search

As one of the most basic iterative heuristic methods, which typically produce suboptimal solutions (far away from optimal) in short period of time, local search algorithms can find approximate solutions to large scale combinatorial optimization problems [8]. The fundamental principle underlying a local search algorithm is that it always moves from the current solution to the next better solution of the neighborhood in a greedy way.

```
1.  SetExitCriteia();    /*set iteration number*/
2.  S = InitialPlacment(); /*start of loop*/
3.  While(ExitCriteria() == false)
4.  {    Block1 = RandomSelectBlock();
5.       Block2 = RandomSelectBlock();
           /*randomly pick up two blocks in the whole neighborhood region*/
6.       C = Cost(Block1) − Cost(Block2);
           /*calculate the change of cost if swapping these two blocks*/
6        if( $\Delta C < 0$ )     /*only accept teh improving swaps*/
             S = SwapPosition(Block1, Block2);
7. }    /*end of the loop*/
8.     /*get the final placment solution S*/
```

Fig. 2. Pseudo-Code for simple local search

In this paper, Simple Local Search (SLS) uses the simplest iterative improvement strategy that swaps the blocks in a window which limits the swapping region. Initially the window is large, and as the algorithm progresses the window shrinks in size. By starting from an initial legal solution, a considerable amount of time is required to search the whole neighborhood which is prohibitively large for NP-hard problem, while attempting to achieve improvements. The pseudo-code for this strategy is shown by Figure 2.

3.3. Immediate Neighbor Local Search

A new local search method is developed to achieve a good placement solution in a short time. Limiting the scope of swaps within the region of the original block position has shown to give superior results compared to unrestricted moves when a globally good placement is already achieved [2]. Therefore, this method checks the vicinity of target block and swaps the nearby blocks around the target block as shown by Figure 3. The next seed block is also chosen from the immediate neighbors of previously selected block.

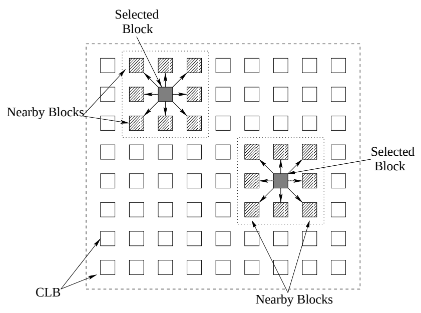

Fig. 3. Searching Region of INLS

The Pseudo-code for INLS is shown in Figure 4. The basic principle underlying this heuristic typically involves the interchange of any pair of blocks that would eventually lead to an improvement in wire-length cost. In INLS algorithm,

```
1. S = InitPlacement();
2. While(ExitCriterion() == False)          /*start of loop*/
3. {    for(i = 0; i < NumberOfTotalBlocks;  i++)      /*start of inner loop*/
4.      {    createListOfNearby(SeedBlock(i)); /*create the list of blocks around the selected block*/
5.           for(j = 0; j < NumberOfNearbyBlocks; j++) /*search the adjacent neighbors*/
6.           {      Candidate = SelectNearbyBlock();
7.                  $\Delta C$ = Cost(Candidate) − Cost(SeedBlock(i));  /*calculate the change in the cost*/
8.                  if( $\Delta C$ < 0)        /*only accept the improving swaps*/
9.                  {
10.                         S = SwapPosition(SeedBlock(i), Candidate);
11.                         Break;
12.                 }      /*avoid doing greedily search, once find the improvement, break*/
13.          }
14.          if( no swap happen);    /*avoid the early convergence*/
15.          {      Candidate = randomSelectBlock();
16.                 $\Delta C$ = Cost(Candidate) − Cost(SeedBlock(i));
17.                 if( $\Delta C$ < 0)
18.                        S = SwapPosition(SeedBlock(i), Candidate);
19.          }
20.     }  /*end of inner loop*/
21. }  /*end of loop*/
        /*get the improved initial solution*/
```

Fig. 4. Pseudo-Code for INLS

we limit this neighborhood to a very small region – adjacent to the target block. The algorithm initiates the search from any location within the FPGA layout. The first seed block is chosen randomly and the next seed block is selected from the neighbors of previously selected block. If swapping nearby blocks lead to a deterioration of the solution quality, a random block is chosen to further reduce the wire-length. This approach aids in expanding the search space and prevents early convergence. The INLS algorithm uses the greedy strategy to scan and evaluate the current placement and accepts the best solution until no further improvement is obtained. This greedy method can guarantee the best possible move in the solution space and eventually stops at a local minima.

Circuit name	FPGA matrix	Number of CLBs	Number of I/O Pads	Number of Nets	Average Fanout
e64	17x17	274	130	290	3.94
tseng	33x33	1047	174	1099	4.28
ex5p	33x33	1064	71	1072	4.73
alu4	40x40	1522	22	1536	4.52
seq	42x42	1750	76	1791	4.46
frisc	60x60	3556	136	3576	4.48
spla	61x61	3690	62	3706	4.73
ex1010	68x68	4598	20	4608	4.49
s38584.1	81x81	6447	342	6485	4.18
clma	92x92	8383	144	8445	4.61

Table 1. MCNC Benchmark circuit suite used as test cases

4. EXPERIMENTAL RESULTS

In this paper, our algorithms target an island-based FPGA model with each CLB including a 4-input lookup table and a D flip-flop. Ten MCNC [9] benchmark circuits, shown in Table 1, are divided into three categories: small, medium and large. The algorithms were implemented in C++/C and executed on a Sun Sparc10 dual alpha CPU workstation with Solaris UNIX 8.0 operating system. We carried out CSS

Circuit name	Average random inital cost	Average final cost	Average CPU runtime(s)	Total Improvement%
e64	7542	6992	0.01	6%
tseng	41286	34808	0.05	16%
ex5p	42301	37381	0.04	12%
alu4	61504	53175	0.08	14%
seq	79903	69390	0.11	13%
M.avg	46489	40349	0.05	12%
frisc	229152	177393	0.44	23%
spla	236251	181525	0.47	23%
ex1010	332664	264977	0.74	21%
s38584.1	559870	478670	1.33	15%
clma	796591	631368	2.20	21%
L.avg	430905	346786	1.03	20%
Avg	238697	193567	0.54	16%

Table 2. Performance of Cluster Seed Search

with random initial solutions at first. As shown in Table 2, CCS achieves on average 16% improvement over a randomly placed circuit in a short period of time. However, CSS obviously results in poor quality of placement (which will not be accepted as the final solution). In general, a good starting point can help in reducing the convergence time of local search techniques. To evaluate the performance of SLS and INLS, we implemented both methods and compared their performance. Both methods begin with either random initial solutions or solutions constructed by CCS. The results in Table 3 and 4 are obtained by running SLS and INLS 50 times over the ten MCNC benchmark circuits. The results in Table 3 are based on random initial solutions. Table 4 are similar except that initial solutions are based on CCS constructive method. Tables 3 and 4 confirm that both SLS and INLS are fast heuristics. INLS achieves better improvement with less CPU time over medium circuits than SLS. But over the large circuits, INLS takes less CPU time to obtain the same improvement as simple local search. We also conducted some experimental evaluation of a hybrid technique that combines the three methods together. In this experimental setup, initial solutions are obtained via CCS. This is followed by SLS to explore (diversify the search) the solution space. Finally, INLS was used to fine tune the search (intensify the search). Table 5 compares CCS-SLS-INLS with solutions obtained by SLS-R and INLS-R. The table clearly indicates that this hybrid approach achieves on average 15% improvement over the other individual heuristic approaches.

Circuit name	Ave.random initial cost	**SLS-R**		**INLS-R**	
		Ave. cost	Ave.CPU runtime(s)	Ave cost	Ave.CPU runtime(s)
e64	7542	4006	0.06	4004	0
tseng	41286	16478	0.32	15803	0.06
ex5p	42301	21670	0.33	21352	0.14
alu4	61504	28797	0.46	28635	0.18
seq	79903	39080	0.62	39096	0.25
M.avg	46489	26506	0.43	26221	0.12
frisc	229152	102676	1.67	102901	0.59
spla	236251	111485	1.74	110372	1.18
ex1010	332664	138229	2.38	138479	3.0
s38584.1	559870	205301	3.97	204574	2.62
clma	796591	332142	6.82	330038	3.37
L.avg	430905	177967	3.31	177272	2.15
Avg	238697	99986	1.88	99575	1.13
Avg.Impro	-	58%	-	59%	40%

Table 3. Performance of SLS and INLS

Circuit name	Initial Solution Created by CCS	**SLS-C**		**INLS-C**	
		Ave. cost	Ave.CPU time(s)	Ave. cost	Ave.CPU time(s)
e64	6992	3904	0.1	3905	0.02
tseng	34808	16036	0.41	15729	0.18
ex5p	37381	21002	0.42	20445	0.22
alu4	53175	27956	0.56	27517	0.27
seq	69390	38062	0.81	37174	0.36
M.avg	40349	25766	0.55	25216	0.26
frisc	177393	94479	2.09	94450	1.04
spla	181525	101300	2.13	101296	1.65
ex1010	264977	128431	2.82	128115	3.72
s38584.1	478670	194515	5.27	194362	3.94
clma	631368	319227	9.01	318885	6.14
L.avg	346786	167594	4.26	167422	3.31
Avg	193657	94493	2.34	94180	1.76
Avg.Impro	-	51%	-	52%	25%

Table 4. Comparison of SLS and INLS (CCS initial)

5. CONCLUSIONS

Time issues have become a critical concern in FPGA design. Users may be willing to trade placement quality for a reduction in runtimes. In this paper, two new algorithms for the FPGA placement problem have been presented and investigated. CSS constructs a good starting point for other iterative approaches in a very short time. INLS greatly mitigates the runtime in FPGA placement process, while yielding acceptable quality of placements. Future work will concentrate on implementing GRASP like heuristics and Tabu Search to further enhance the solution quality.

Circuit name	**SLS-R**		**INLS-R**		**CSS-SLS-INLS**	
	Ave. cost	Ave.CPU runtime(s)	Ave cost	Ave.CPU runtime(s)	Ave cost	Ave.CPU runtime(s)
e64	4006	0.06	4004	0	3647	0.11
tseng	16478	0.32	15803	0.06	14059	0.7
ex5p	21670	0.33	21352	0.14	20076	0.75
alu4	28797	0.46	28635	0.18	25927	1.07
seq	39080	0.62	39096	0.25	35997	1.5
M.avg	26506	0.43	26221	0.12	24014	1.01
frisc	102676	1.67	102901	0.59	92098	4.12
spla	111485	1.74	110372	1.18	100592	5.2
ex1010	138229	2.38	138479	3.0	110097	8.01
s38584.1	205301	3.97	204574	2.62	173668	13.9
clma	332142	6.82	330038	3.37	272831	20.21
L.avg	177967	3.31	177272	2.15	149857	10.28
Avg	99986	1.88	99575	1.13	84898	5.56

Table 5. Performance of SLS, INLS and Hybrid

6. REFERENCES

[1] K. Shahookar and P. Mazumder, "Vlsi cell placement techniques," in *ACM Computing Surveys (CSUR), Volume 23 Issue 2*, June 1991.

[2] J. Lam, J. Delosme, and C. Sechen, "Performance of a new annealing schedule," in *IEEE transactions on Computer-Aided*, March 1988.

[3] Huiqun Liu, Kai Zhu, and D. F. Wong, "Circuit partitioning with complex resource constraints in fpgas," in *Proceedings of the 1998 ACM sixth international symposium on Field programmable gate arrays*, March 1998.

[4] Pongstorn Maidee, Cristinel Ababei, and Kia Bazargan, "Compilation techniques for reconfigurable devices: Fast timing-driven partitioning-based placement for island style fpgas," in *Proceedings of the 40th conference on Design automation, pp 598 - 603*, June 2003.

[5] Vaugh Betz and Jonathan Rose, "**VPR**: A new packing, placement and routing tool for fpga research," in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, 1997.

[6] Yaska Sankar and Jonathan Rose, "Trading quality for compile time: ultra-fast placement for fpgas," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, February 1999.

[7] Chandra Mulpuri and Scott Hauck, "Runtime and quality tradeoffs in fpga placement and routing," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays,* February 2001.

[8] E.Arts and J. K. Lenstra, *Local Search in Combinatorial Optimization*, Princeton University Press, 2003.

[9] S. Yang, "Logic synthesis and optimization benchmarks," in *Tech. Report, Microelectronic Center of North Carolina*, 1991.