Timing-Driven Placement for Heterogeneous Field Programmable Gate Array

Bo Hu
Velogix Inc. Santa Clara, CA 95054, USA
hu@velogix.com

Abstract - In this paper, a new timing-driven placement algorithm is proposed to handle complicated placement requirements inherent in FPGAs with heterogeneous resources (dedicated logic block, memory block). The new algorithm employs a multi-layer density system with each layer modeling a drastically different architectural resource. By introducing the multi-layer density system, a heterogeneous placement task is translated to a set of homogeneous ones, with each of them being handled at a different density layer. We also present a new iterative timing optimization scheme which is seamlessly integrated in the placement process. The tight interaction between the placement and timing optimization produces superior timing results for industrial designs.

I Introduction

As semiconductor process advances into deep sub-micron regime, the cost of manufacturing a complex Application-Specific Integrated-Circuit (ASIC) chip using the state-of-art technology is sky-rocketing. As a viable solution to reduce cost, shorten product development cycle and minimize production risk, Field Programmable Gate Array (FPGA) has been gaining acceptance in various applications than ever before. Traditional homogeneous FPGA is mainly based on programmable Look-Up Tables (LUTs). Its logic density and performance are usually inferior to ASIC implementation. However, as the leading-edge technology is more rapidly adopted in FPGA industry, and more ASIC-like dedicated functional blocks are integrated nowadays, the overall density and performance disadvantages are mitigated in modern high-end FPGAs [10][11]. The integration of such dedicated blocks marks the transition from homogeneous architecture to heterogeneous. Fig. 1 shows a simplified example of a heterogeneous architecture. It consists of two-dimension array of Basic Process Unit (BPU). Each BPU contains a two-dimension array of LUTs, a computing unit (CU) and a memory block. In this paper, heterogeneous FPGAs refer to those architectures with heterogeneous resources (like CUs, memory blocks) embedded sparsely in homogenous LUT distribution.

Heterogeneous architectures present new challenges for FPGA design tools, for example, placement tool. Given a netlist of design components, the task of a timing-driven placer is to assign the components into the proper locations on the target FPGA chip while optimizing design performance. A component might be as simple as a single LUT; it can also be a complex functional block.

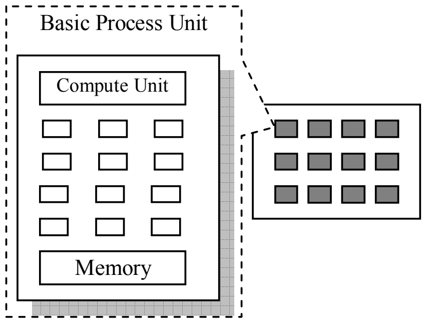

Figure 1: An abstract view of a simplified example heterogeneous FPGA

Conventional timing-driven FPGA placement algorithm was based on simulated annealing [6]. It might be possible to adapt simulated annealing to handle heterogeneous FPGA architectures. But the excessive CPU-complexity of simulated annealing makes it not an attractive solution. Recently, analytical placement has regained attention in design automation world due to its superior speed [8] and excellent placement quality [4][5]. Analytical placement formulation is adopted mostly to solve standard-cell or mixed-size placement problems. The inputs to an analytical placer are a graph representing the design netlist and a region specifying where the netlist should be placed. If the placer is timing-driven, it also reacts to timing analysis results to produce an optimized timing result. Each node in the graph is assigned a geometric shape. The output of the placer is an overlapping-free placement of all the nodes in the graph. Two nodes overlap if their geometric shapes intersect with each other. One way of handling the non-overlapping requirement is through density $D(x,y)$. $D(x,y)$ is defined for every location (x,y) within the placement region. It quantifies the amount of overlapping at (x,y). Suppose that each node contributes a unit density to any location where its shape covers. By making sure that the density at any location is equal or less than a unit density, the placer automatically produces an overlapping-free placement. The geometric shape assigned to a node is usually a rectangle. During the placement, rectangles are mapped onto the placement region to compute density distribution. The placement is done when the peak density is less than a threshold value and/or other criteria are met. Since $D(x,y)$ is a two-dimension function, we call it a single-layer density system. The only layer in this system refers to *xy* plane. It has been shown in numerous literature

[2][4][5][7][8] that this single-layer density system works well for standard-cell or mixed-size placement. But as the following example demonstrates, for FPGAs with heterogeneous resources, using single-layer density system makes it hard to choose an appropriate geometric shape for a placement node.

Without loss of generality, let us suppose that the design to be placed consists of only CUs and LUTs. Because LUT distribution has the finest granularity, it can be conceptually viewed as being available everywhere. We use W/W_{LUT} and H/H_{LUT} to annotate the width and height of the rectangle shape for a LUT. W is the width of the placement region, and W_{LUT} is the number of LUT columns. So W/W_{LUT} is the average width of a LUT column. Similarly, H/H_{LUT} defines the average height of a LUT row. For CU, the selection of a geometric shape is not straightforward. Compared to LUT, CU is much more sparsely distributed across the chip. As a result, average column width W/W_{CU} and average row height H/H_{CU} are much larger than their counterparts for LUT. If we use W/W_{CU} and H/H_{CU} to generate the rectangle shape, as Fig. 2 shows, non-overlapping placement prevents LUT components from using the available LUT resources covered by the shape. On the other hand, if we assign CU a smaller geometric shape, as can be seen from Fig. 3, a group of CU components might be closely located in some local region where there are not enough CU resources available. To reduce the demand, some CU components need to be moved away from their optimized positions. Because these heterogeneous resources like CUs are sparsely distributed, finding the nearest resource might still incur significant placement disturbance. As a result, placement quality (timing, routability, etc.) is likely to be damaged.

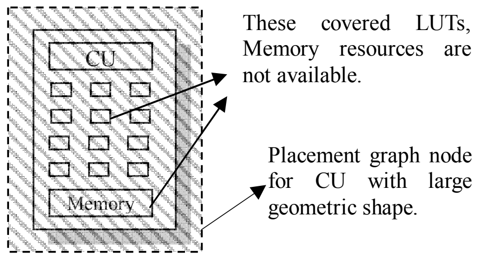
CU
Memory
These covered LUTs, Memory resources are not available.
Placement graph node for CU with large geometric shape.

Figure 2: large geometric shape becomes blockage

To handle the complicated placement requirements inherent in FPGAs with heterogeneous resources (CU, memory), a new timing-driven placement algorithm is proposed. The new algorithm employs a multi-layer density system with each layer modeling a drastically different architectural resource. By introducing the multi-layer density system, a heterogeneous placement task is translated to a set of homogeneous ones, with each of them being effectively handled at a different density layer. The situations shown in Fig. 2 and Fig. 3 are thus avoided. We also present a new iterative timing optimization scheme which is seamlessly integrated in the placement process. The tight interaction between the placement and timing optimization produces superior timing results for industrial designs.

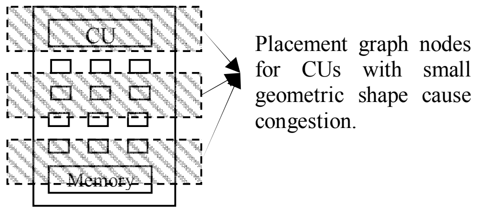
CU
Memory
Placement graph nodes for CUs with small geometric shape cause congestion.

Figure 3: small geometric shape causes congestion

The rest of the paper is organized as follows: section II formulates the timing driven placement problem for heterogeneous FPGAs. Section III presents the new multi-layer density system. The new algorithm based on the proposed density system is discussed in section IV. Section V provides experimental validation and section VI gives the conclusions.

II. Problem Formulation

Before we start the discussion on our major contributions, let us first formulate the timing-driven placement problem addressed in this paper:

(1) An architectural description for the target heterogeneous FPGA chip. A placement region with width W and height H is built based on the description. W_X and H_X are the number of columns and rows respectively for resource type X. So the total number of resources of type X is $W_X \times H_X$.

(2) A design $M(C, I)$. C and I denote the set of components and interconnects respectively. Each interconnect $i(D,R)$ in I connects a subset of C. D is the set of driving components and R is the set of receiving components. A connection in the design is defined as a driver and receiver component pair $p(u \in C, v \in C)$. Each connection is annotated with a slack s_p based on static timing analysis. s_{worst} is the worst slack among all connections. Slack is a metric to measure how well the actual timing meets design requirements. Larger s_{worst} usually suggests that the design can function correctly at a higher clock frequency. A component can be a terminal, a LUT, or a computational block. A terminal is an interface of the design to outside environment. In this work, all terminals have fixed locations. A computational block (CB) is a pre-designed functional block implemented using the

resources available on the chip. The example CB shown in Fig. 4 consists of 3 CUs, 3 memory blocks and 12 LUTs relatively placed within a 2x2 BPU region.

The output is the legal assignment of all the LUTs and CBs on the chip such that s_{worst} is maximized.

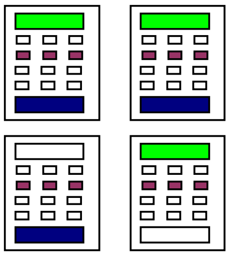
Figure 4: An example computational block (CB)

A placement graph *G(V, E)* is created based on the connectivity of the input design. *V* and *E* is the set of nodes and edges respectively. Each node *v* in *V* represents a component in *C*. The edge set *E* is built by constructing a clique over set $D \cup R$ for every interconnect *i(D,R)* in *I*. Each edge *e* is assigned an initial weight $w_e = 1/(|D|+|R|-1)$. Clearly, every connection has a corresponding edge in the graph while some edges might not have corresponding connections because set *E* is a superset of the set of all connections.

III. Multi-layer Density System

We have discussed in the introduction that a single-layer density system is not sufficient to handle complicated placement requirements for heterogeneous FPGAs. More specifically, each type of architectural resource (LUT, CU, memory block) has its own distribution on the chip. Usually, CUs and memory blocks are much more sparsely distributed than LUTs. A single-layer density system is unable to satisfy these drastically different distribution requirements simultaneously. However, as demonstrated in standard-cell ASIC designs, single-layer density system can handle homogeneous resources very well. It seems feasible to extend single-layer density system to multi-layer system with each layer modeling a unique placement requirement originating from a particular resource type. For example, for the architecture shown in Fig. 1, three layers are constructed. They are used to model the distribution of LUTs, CUs, and memory blocks during the placement respectively.

Because a component represented by a placement node can contain any type of resource, it contributes differently to different density layer. Before a node is mapped to a density layer, we need to first determine its geometric shape. For different layers, the shape is different. Specifically, we choose a rectangle with width W/W_X and height H/H_X for a single resource of type *X*. If a component demands multiple resources of *X*, multiple rectangles are combined according to the relative position defined by the component.

In the following, we use the example component in Fig. 4 to illustrate how rectangles are combined to form a complex shape.

Fig. 5 shows the complex shape generated for CU density layer. The given component includes three CUs organized in an upper triangular shape. It can be seen that the generated shape matches exactly the triangular organization. Similarly, Fig. 6 shows that the complex shape for memory blocks resembles the L-shape composition. For twelve LUTs in the component, two parallel strips are created as shown in Fig.7.

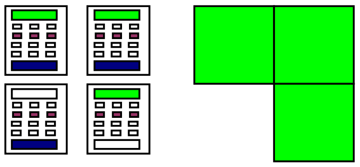
Figure 5: Complex shape for CU density layer

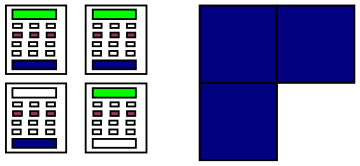
Figure 6: Complex shape for memory block density layer

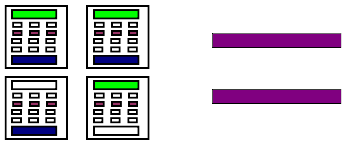
Figure 7: Complex shape for LUT density layer

With the new multi-layer density system and the proper shape generation procedure shown above, a heterogeneous placement task is translated to a set of homogeneous ones, with each of them being handled at a different density layer. In other words, the placer needs to make sure that density is properly distributed for all the layers in order to avoid both resource waste (Fig. 2) and resource competition (Fig.3). Because CUs and LUTs are mapped to different density layers, overlapping between CUs and LUTs is legal and does not cause resource competition.

As a remark on the multi-layer density system, we point out that it can be applied in other placement contexts such as thermal placement. If a design consists of several big power consumers, it’s desired that they are placed far away from each other to even out temperature distribution. This can be done by constructing a two-layer density system with one layer representing the power density.

IV. Timing-Driven Placement

The new multi-layer density system creates multiple placement tasks of different characteristics. In general, geometric shapes generated for sparsely distributed resources (CUs, memory blocks) are usually a lot bigger than those for LUTs. It is especially true when a computational block involves complicated logic/arithmetic computation. Distribution of these shapes is more like floorplanning than placement in traditional sense. What it means is that a traditional ASIC-standard-cell or a FPGA LUT placer may be used for LUT layer while a ASIC floorplanner or mixed-size placer is a good fit for CU and memory block layers. Since the placement at one layer is intimately affected by those at other layers, it is desirable to perform the placement at each layer simultaneously. As can be seen in the rest of this section, our new algorithm can be viewed as starting a separate placement or floorplanning engine for each individual layer at the same time and letting them interact with each other along the process.

It should be noted that one of our major contributions is the introduction of multi-layer density system which makes it feasible to apply existing placement, floorplanning, or mixed-size placement algorithms [5][6] at individual layers to solve heterogeneous FPGA placement problem. The specific algorithms applied at individual layers are not our focus. In this work, we choose to use expansion-based placer [3][4] as the underlying placement engine and enhance it with the new timing optimization scheme.

In the following, we first briefly review the expansion technique. We then discuss how density is computed at different layers, and finally present the new timing-driven placement algorithm with multiple density layers.

A. Expansion Basics

Expansion refers to the process during which geometric shapes are gradually distributed over a specified region. Expansion in [3] is based on fixed-points addition technique. Basically, in analytical placement formulation, nodes tend to cluster to each other due to intrinsic attracting forces induced by connections/edges. The magnitude of an intrinsic force is determined by the weight and the length of the connection. A connection with larger weight and longer length induces stronger intrinsic force. Fixed-points are used to apply additional attracting forces on the nodes and work against intrinsic ones in order to pull the nodes away from high density area. As a result, the peak density usually decreases as expansion proceeds. The placer based on expansion consists of a sequence of expansion iterations. It stops when density distribution satisfies preset criteria.

B. Density

We impose a two-dimension bin structure on each layer. Density at bin *b* is defined as follows:

$$d(b) = (\sum_{n} A(b,n) / A(b)$$

Where *A(b,n)* is the intersection area between *b* and node *n*; *A(b)* is the area of bin b. *A(b,n)* is summed up over all nodes intersecting bin *b*. To compute the density efficiently, the bin size varies at different density layers. LUT layer has the finest granularity while the ones for memory blocks and CUs are larger depending on the architecture.

C. Timing-Driven Expansion

Our new timing-driven expansion algorithm, TD-ML, is given in Fig. 8.

```
TD-ML
{   buildNodeShapes();
    initialPlacement();
    expansion iterates until some stopping criteria {
        timingOptimization();
        For each density layer l{
            computeDensity(l);
            computeExpansionFixed-points(l);
            performExpansion(l);
        }
    }
    legalization();
}
```

Figure 8: The new algorithm

buildNodeShapes() builds a set of shapes, one per each layer, for each placement node as Fig. 5-7 illustrate. Next, an initial placement is generated before expansion starts. Within the expansion loop, the first step is timing optimization. In general, to maximize s_{worst}, critical connections should be as short as possible. In this work, it is done by adjusting weights on critical connections. The basic idea is to increase weights on long critical connections such that they become shorter in next expansion iteration. First, timing analyzer is called to calculate slack s_p for all connections and the worst slack s_{worst} based on the present placement. We use the following weighting strategy:

$$w_p[j] = w_p[j-1] \times (1.0 + f[j])$$

$w_p[j]$ and $w_p[j-1]$ is the weight for connection *p* at *j*th and *j*th -1 expansion respectively. *f[j]* is the adjustment factor at *j*th expansion and determined as follows:

$$f[j] = \begin{cases} 0 & s_p > s_{worst} + \varepsilon \,||\, l_p \le l_{p\min} \\ f_0[j]\left(1 - \dfrac{s_p - s_{worst}}{\varepsilon}\right) \times \left(\dfrac{l_p - l_{p\min}}{l_{p\max}}\right) & otherwise \end{cases}$$

$f_0[j]$ is the preset maximum adjustment factor at *j*th iteration. As increasing weights on connections adversely affects the expansion process by making it difficult to move nodes due to larger intrinsic attracting forces induced by increased weights, $f_0[j]$ is decreased as the placement proceeds. In this work, $f_0[0]$ is set to be 1 and gradually approaches to zero. ε is a preset value used to decide whether a connection is critical. A connection is critical if s_p is smaller than $s_{worst} + \varepsilon$. l_p is the current length of connection *p*. $l_{p\min}$ and $l_{p\max}$ is the minimum and maximum length of *p* respectively. $l_{p\min}$ is determined by enumerating all possible placements for the driver and the receiver component of the connection at all density layers. Simply, $l_{p\min}$ is the optimum length for connection *p*. The conditional equation above suggests that a connection gets a non-zero adjustment factor if and only if it is critical and its length is larger than $l_{p\min}$. Clearly, there is no meaning to further reduce the length beyond $l_{p\min}$ because any length less than $l_{p\min}$ is infeasible in the architecture. $\left(1-\frac{s_p - s_{worst}}{\varepsilon}\right)$ part represents the criticality of a connection. It evaluates to be 1 if a connection is the most critical one. $\left(\frac{l_p - l_{p\min}}{l_{p\max}}\right)$ is used to penalize longer critical connections than shorter ones. If a connection is already close to its minimum length, only a small change in weight might be sufficient.

Compared to net weighting strategies in [2][7], our approach introduces (1) an iteration-dependent maximum adjustment factor $f_0[j]$, which ensures well distribution of nodes at the end of expansions; (2) a slack-dependent component $\left(1-\frac{s_p - s_{worst}}{\varepsilon}\right)$ which avoids hard net constraints [7]; (3) a length-dependent component which prevents short connections from being assigned unnecessary weights while penalizing long connections.

After the weight for every critical connection is updated, expansion is iterated over all density layers. Note that the expansions are subject to new connection weights and thus work against the updated intrinsic attracting forces. At each layer, we first compute the current density distribution. Then for any placement nodes contributing density at this layer, we compute expansion fixed-points aiming at reducing highest density through expansion. Expansion fixed-points are properly normalized such that expansions at different density layers are executed at the same pace. The expansion is performed when fixed-points are introduced into the expansion solver and the locations of the nodes are updated when the expansion is done. It is worthy mentioning that even a node does not contribute density at current layer and thus is not assigned an expansion fixed point at this expansion, its location might still be changed due to the fact that all the nodes are interconnected and the movement of a single node might affect all the rest of movable nodes. For instance, the expansion of shapes at CU layer might cause relocations of the shapes at LUT layer.

The expansion described above is iterated until the peak density among all layers is less than a threshold or the number of iterations exceeds a preset maximum number.

Following the end of expansions, a legalization procedure is called to fit the components on the input architecture based on the expansion output. As a result, each component is assigned to the closest empty legal location.

V. Experiments

We implement the proposed new placement algorithm in a commercial FPGA design tool. A set of industrial Digital Signal Processing (DSP) designs are used to validate its effectiveness. Timing constraints are set for each design in order to achieve maximum frequency. All the experiments have been done on a 2.4Ghz Pentium 4 processor running linux Redhat9.0. The target FPGA device is the latest one from [9].

The new algorithm is first tested on a digital filter design including a 2-D FFT module. This module contains more than 100 CUs and 100 Memory blocks. We compare the new algorithm to the one with one-layer density system. Fig. 9a shows the placement using one density layer, and Fig. 9b shows the corresponding distribution of CUs. Clearly, although the density distribution is very well (Fig. 9a), CUs are not distributed evenly. A lot of overlapping occurs in Fig. 9b. It means that many CUs compete for the same resource during the legalization. Some CUs have to be moved away from their original positions in Fig. 9b. As a result, legalization incurs a significant discrepancy between the placement produced by the expansion and the final legalized result, and adversely affects timing. For this example, the design runs at 308 MHz after legalization.

In Fig. 10a and Fig. 10b, we show expansion results at LUT and CU layer for the same design using multi-layer density system. Compare Fig. 9a and Fig. 10a, the expansions are equally good. But new density system also did extremely well on the floorplanning of CU blocks as shown in Fig. 10b. The overlapping between CU shapes is marginal. After legalization, the design runs at 426 MHz, a significant 38% speedup over single-layer density system.

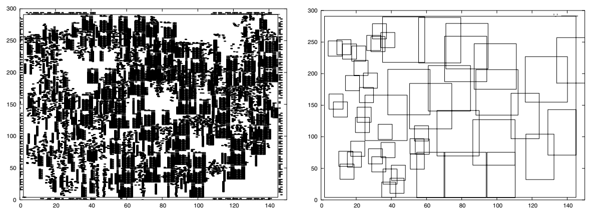
Figure 9: (a) single-density layer placement, (b) CU-layer floorplan

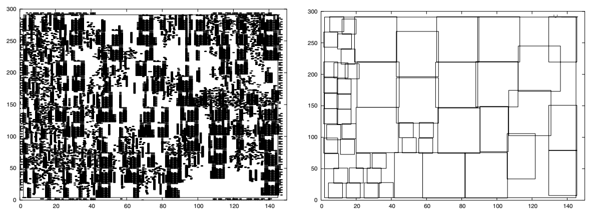
Figure 10: (a) LUT-layer placement (multi-layer), (b) CU-layer floorplan (multi-layer)

We also test the new algorithm on other industrial designs. In addition, a non timing-driven flow is constructed to evaluate the effectiveness of our new timing optimization scheme. In Table I, six industrial designs are listed. The number of LUTs in these designs ranges from a few thousands to more than 20k. Each design contains a different DSP algorithm. Compared to non-timing-driven expansion algorithm NTD-SL (by skipping timing optimization in Fig. 8), timing optimization TD-SL shows on the average 9% improvement. If multi-layer density system is used, an average 24% better timing is achieved. Two observations can be made from the results. First, TD-ML is most effective for CU-intensive designs (the first three designs). As expected, for the designs with low CU utilization (the last three designs), little difference between TD-ML and TD-SL is observed. Overall, over 25% performance improvement was obtained. Interestingly, TD-SL sometimes is unable to improve timing (Filter1 and Filter2) because the timing optimization effort is overwhelmed by significant legalization discrepancy due to unbalanced CU/memory distribution as shown in Fig. 9b. Second, for the designs with low CU/memory usage (the last three designs), an average 20+% better performance was achieved by timing optimization.

Table II lists the total wire length and runtime results for three flows. CPU times are given in seconds. Overall, timing-driven flows (TD-SL and TD-ML) requires more CPU time than non-timing-driven one because of static timing analysis. Between TD-SL and TD-ML, the runtime difference is negligible. It can be also observed that TD-ML not only achieves much better timing, it also results in less total wire length, on the average 6% less than NTD-SL. This is because multi-layer density system ensures balanced distribution for all architectural resources (LUT, CU, memory block), and consequently causes much less placement discrepancy before and after legalization.

TABLE I
Performance comparison

Designs	Frequency(MHz)		
	NTD-SL	TD-SL	TD-ML
Filter1	317	308	426
Filter2	281	308	339
Filter3	339	317	426
Mult	278	376	380
Video	205	244	244
Encoder	167	179	179
Ave	1.0	1.09	1.24

VI. Conclusions

We presented a timing-driven placement algorithm based on a new multi-layer density system. The new algorithm is proved to be very effective to handle complex placement requirements inherent in heterogeneous FPGAs and produce superior timing results for industrial designs.

References

[1] L. Cheng, M. D.F. Wong, “Floorplan Design for Multi-Million Gate FPGAs”, in Proc. *Intl. Conf. on CAD*, pp. 292-299, 2004.
[2] H. Eisenmann, F. M. Johannes, “Generic Global Placement and Floorplanning”, in Proc. ACM/IEEE DAC, 1998.
[3] B. Hu, and M. Marek-Sadowska, "FAR: Fixed-points and Relaxation based Placement", Proc. Intl. Symp. on Physical Design, San Diego, 2002
[4] B. Hu, Y. Zeng and M. Marek-Sadowska, "mFAR: Fixed-points Addition based Placement Algorithm", ISPD05. April, 2005
[5] A. B. Kahng, S. Reda, Q. Wang, “APlace: A General Analytic Placement Framework”, in Proc. *Intl. Symp. on Physical Design,* pp. 233-236, 2005.
[6] A. Marquardt, V. Betz and J. Rose, “Timing-Driven Placement for FPGAs,” in Proc. *Intl. Symp. FPGAs*, pp. 203-213, 2000.
[7] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhary, B. Halpin, “Timing Driven Force Directed Placement with Physical Net Constraints”, in Proc. Intl. Symp. on Physical Design, pp. 60-66, 2003.
[8] N. Viswanathan and C. C.-N Chu, “FastPlace: Efficient Analytical Placement using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model,” in Proc. *Intl. Symp. on Physical Design*, 2004.
[9] Velogix, Inc. http://www.velogix.com
[10] http://www.xilinx.com
[11] http://www.altera.com

TABLE II
Total wire length and CPU time comparison

Designs	NTD-SL		TD-SL		TD-ML	
	WL	CPU	WL	CPU	WL	CPU
Filter1	5.23	70	5.12	210	5.60	216
Filter2	1.72	16	1.81	116	1.39	119
Filter3	1.16	10	1.26	95	1.01	102
Mult	6.48	25	5.90	242	5.73	247
Video	2.98	12	3.3	33	2.91	34
Encoder	6.14	9	6.24	13	6.24	14
Ave	1.0	1.0	1.01	5.6	0.94	5.8