International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:12, 2007

Faster FPGA Routing Solution using DNA Computing

Manpreet Singh, Parvinder Singh Sandhu, and Manjinder Singh Kahlon

Abstract**—**There are many classical algorithms for finding routing in FPGA. But Using DNA computing we can solve the routes efficiently and fast. The run time complexity of DNA algorithms is much less than other classical algorithms which are used for solving routing in FPGA. The research in DNA computing is in a primary level. High information density of DNA molecules and massive parallelism involved in the DNA reactions make DNA computing a powerful tool. It has been proved by many research accomplishments that any procedure that can be programmed in a silicon computer can be realized as a DNA computing procedure. In this paper we have proposed two tier approaches for the FPGA routing solution. First, geometric FPGA detailed routing task is solved by transforming it into a Boolean satisfiability equation with the property that any assignment of input variables that satisfies the equation specifies a valid routing. Satisfying assignment for particular route will result in a valid routing and absence of a satisfying assignment implies that the layout is un-routable. In second step, DNA search algorithm is applied on this Boolean equation for solving routing alternatives utilizing the properties of DNA computation. The simulated results are satisfactory and give the indication of applicability of DNA computing for solving the FPGA Routing problem.

Keywords**—** FPGA, Routing, DNA Computing.

I. Introduction

THE Routing affects the performance of FPGA-based systems in two major ways. First, a typical design must be partitioned and mapped onto several FPGAs. Because FPGA size is fixed, the ability to pack larger partitions onto a single FPGA can reduce the total number of partitions (and hence FPGAs) required to implement the design. The feasibility of implementing a piece of the design on a single FPGA is often limited by routing-resource availability.

Second, since FPGA resource utilization typically does not exceed 80%, considerable flexibility remains onboard the FPGA for optimizing the routing.

Paper has Submitted for review on Nov. 22, 2007.

Parvinder S. Sandhu is Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA (Phone: +91-98555-32004; (Email: parvinder.sandhu@gmail.com)

Manpreet Singh is Lecturer with Information Technology Department, Guru Nanak Dev Engineering College, Ludhiana (Punjab) - 141006 INDIA (Phone:+91-9888879002; Fax: +91161-2490339; (e-mail: mpreet78@gmail.com)

Manjinder Singh Kahlon is working with Computer Science & Engineering Department, Rayat Institute of I.T., Railmajra (Punjab,) INDIA

For example, we could reduce signal propagation delay through critical paths by using the most direct interconnections (i.e., shortest paths), where a secondary criterion is to minimize wirelength in order to reduce capacitance and conserve routing resources.

There are two models of routing networks: the segmented and non-segmented.

a) Non-segmented model: A non segmented model as a regular grid of five horizontal and five vertical metal lines passing between switch blocks S. The switch blocks are rectangular switch boxes they are used to connect the wiring segments in one channel segment to those in another. Depending on the topology of the S block, each wiring segment on one side of S may be switch able to either all or some fraction of wiring segment on each side of the S block. Fewer the wiring segments the wiring can be switched to, the harder the FPGA is to route.

In addition to the switch blocks, there are connection blocks that are used to connect the logic blocks pins to the routing channel depending on the topology, each L block pin may be switch able to either all or some fraction of wiring segments a pin can be switched to, the harder the FPGA is to route.

b) Segmented model: In segmented model, the tracks in the channels contain predefined wiring segments of same or different lengths. Other wiring segment passes through the channels vertically each input and output of a logic block is connected to a dedicated vertical segment, as a result there are no vertical constraints. There are additional global vertical lines, which provide connection between different channels. Connection between two horizontal segments is provided through an antifuse, where as a connection between horizontal and vertical segment is provided through cross fuse programming one of these fuses provides a low resistance bi-directional connection between two segment . When blown, anti-fuses connect the two segment two form a longer one. In order two program fuse a high voltage is applied cross.

There are additional global vertical lines, which provide connection between different channels. Connection between two horizontal segments is provided through an antifuse, where as a connection between horizontal and vertical segment is provided through cross fuse programming one of these fuses provides a low resistance bi-directional connection between two segment [1]. When blown, anti-fuses connect the

two segment two form a longer one. In order two program fuse a high voltage is applied cross.

There are many classical algorithms for finding routing in FPGA. But Using DNA computing we can solve the routes efficiently and fast. The run time complexity of DNA algorithms is much less than other classical algorithms which are used for solving routing in FPGA. In this paper we have proposed two tier approach for the FPGA routing solution.

II. Proposed Solution

We have tried to solve geometric FPGA detailed routing task by transforming it into a Boolean satisfiability equation with the property that any assignment of input variables that satisfies the equation specifies a valid routing [2]. Satisfying assignment for particular route will result in a valid routing and absence of a satisfying assignment implies that the layout is unroutable. In second step DNA search algorithm is applied on this Boolean equation for solving routing alternatives utilizing the properties of DNA computation. The approach relies on DNA Satisfiability Detailed Router (DSDR) that uses systematic search with quantum search algorithms capable of handling very large SAT instances. To make the picture clearer let us take a brief look at what is Boolean satisfiability (SAT).

The Boolean satisfiability problem (SAT) is a decision problem considered in the complexity theory. An instance of the problem is defined by a Boolean expression written using only AND, OR, NOT, variables, and parentheses. The question is: given the expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true. Detailed FPGA routing problem can be solved by transforming the routing problem as large but atomic Boolean equation. By representing the routing problem as Boolean function one can also prove that particular routing alternative does not exist or the netlist is unroutable. Modern SAT-solvers are enriched with clause-learning and backtracking techniques to help prune the solution space. Boolean Satisfiability (SAT) is the problem of finding a solution (if one exists) to the equation f=1, where f is a Boolean formula to be satisfied. The formula (f) can be represented in Conjunctive Normal Form (CNF), or with Binary Decision Diagrams (BDDs) [3]. There are two classes of high-performance algorithm for solving instances of SAT in practice: modern variants of the David-Putnam-Loveland algorithm, such as GRASP, Zchaff, and stochastic local search algorithms, such as WalkSAT.

Particularly in hardware design and verification applications, satisfiability and other logical properties of a given propositional formula are often decided based on a representation of the formula as a binary decision diagram (BDD). Classically many search style solutions have been proposed for SAT, the most well known being variations of the Davis-Putnam procedure. The best-known version is based on a backtracking search algorithm that, at each node in the search tree, elects an assignment and prunes subsequent search by iteratively applying the unit clause and the pure literal rules. The other algorithms that are used in the SAT based problems are backtracking search [3], resolution based checker, integer linear programming based routing, BDD, recursive learning. We have taken DNA search algorithms to solve this problem Using these concepts we can develop DNA search algorithms that can find the required routing solutions more quickly and effectively than is possible on a classical computer.

In the proposed method FPGA detailed routing is formulated in Boolean Satisfiability problem (SAT). The basic idea was that we construct a set of Boolean functions representing routing constraints over the entire FPGA, and invoke a quantum Boolean SAT solver on the generated function to find any satisfying assignments. Finally the found SAT solution determines precisely a full FPGA detailed routing solution. Our new, DNA satisfiability based detailed FPGA router is based on the DNA properties to solve the Boolean satisfiability.

For each two-pin connection, a global router produces a set of individually feasible global route alternatives [4]. Due to detailed route conflicts, not all the global alternatives can survive. DSDR then considers current detailed routing solutions as well as the multiple global route alternatives by DNA strands. Each two-pin connection generates a Boolean routability function R (X) that captures all the possible routing constraints over the existing routing solution simultaneously. Finally, a DSDR Boolean SAT solver is invoked on the routability function to determine if there exists any legal detailed routing solution.

Our Boolean routability function R (X), where X is a suitable Boolean vector of binary variables that encode the track number for each two-pin connection, can be expressed as the conjunction given below:

$$R(X)=L(X)\wedge E(X) \tag{1}$$

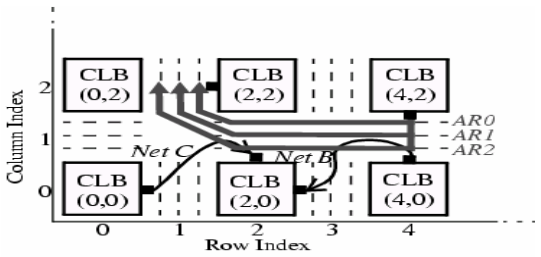
Liveness constraint function L (X) guarantees that at least one global route alternative per two-pin connection should be chosen as a final legal routing solution. *Exclusivity constraint function E (X)* ensures that electrically distinct nets with overlapping vertical or horizontal spans in the same channel are always assigned to different tracks.

A global router is invoked to assigns a set of *n* global route alternatives for each two-pin connection. The method of generating global routes per two-pin connection is an independent procedure from the detailed routing formulation. Liveness and exclusivity constraint are generated to yield the routing constraint Boolean function *R (X)* in conjunctive normal form. The routing alternatives of a netlist are modeled in terms of Boolean variables that represent all of the detailed routes admissible by the given global routing solution. Within the global routing region specified for net *A*, for example, there are only three possible detailed routes indicated by the three Boolean variables *AR0, AR1, AR2*. A similar set of routes and corresponding route variables is created for nets *B*

and *C*. A particular route is considered as the routing solution if its corresponding Boolean variable is assigned the logic value 1, and is excluded otherwise. The liveness constraint for a given two-pin connection has a simple form, namely an OR over the connection's F_c route variables (see Fig.1-b). For a netlist with *n* two-pin connections, liveness constraints yield a set of *n* CNF clauses, each containing F_c positive literals. This type of routing constraint is evaluated as follows, e.g. Excl (Resource (4,1,0))= $\left(\overline{AR0}\right)\vee\left(\overline{BR0}\right)$ indicates that the routing resource, track segment 0 of *C*-block (4,1), can only be used by either detailed route of net *A* or detailed route 0 of net *B*, but not both. In general, if different detailed routes from different nets are competing for the same routing resource, a set of exclusivity $k(k-1)/2$ constraints are created to insure that at most one of those detailed routes are assigned to that resource. The routability of a netlist for a given placement and global routing configuration is expressed by a single Boolean function which is the conjunction of all liveness and exclusivity constraints:

$$R(X)=\left[Live(n)\wedge Excl(n)\right] \quad for\ all\ n\in\ all\ Nets \qquad (2)$$
$$where \quad r\in all\ \text{Re}\ sources$$

Where X is a vector of Boolean variables that represent the possible detailed routes for each of the nets.



NET A route Boolean variable (AR0, AR1, AR2)
NET B route Boolean variable (BR0, BR1, BR2)
NET C route Boolean variable (CR0, CR1, CR2)
(a)

Liveness (A)= $(AR0\vee AR1\vee AR2)$
Liveness (B) = $(BR0\vee BR1\vee BR2)$
Liveness (C) = $(CR0\vee CR1\vee CR2)$
(b) Liveness constraints

Exclusively (Resource (4,1,0))= $\overline{AR0}\vee\overline{BR0}$
Exclusively (Resource (4,1,1)) = $\overline{AR1}\vee\overline{BR1}$
Exclusively (Resource (4,1,2)) = $\overline{AR2}\vee\overline{BR2}$
Exclusively (Resource (2,1,0)) = $\overline{AR2}\vee\overline{CR0}$
Exclusively (Resource (2,1,1)) = $\overline{AR2}\vee\overline{CR1}$
Exclusively (Resource (2,1,2)) = $\overline{AR2}\vee\overline{CR2}$
(c) Exclusivity constraints

Fig. 1 Global routing configuration for NETS A, B and C and three possible detailed routes for NET A.

We propose and analyze a simple new randomized algorithm called ResolveSat for finding satisfying assignments of Boolean formulas in conjunctive normal form. The algorithm consist of two stages: a preprocessing stage in which resolution is applied to enlarge the set of clauses of the formula, followed by search stage that uses a simple randomized greedy procedure to look for satisfying assignment. We show that for each k, the running time of ResolveSat on a k-CNF formula is significantly better than $2^{n,}$ even in the worst case. In particular, we show that the algorithm finds a satisfying assignment of a general satisfiabilty 3-CNF in time O $(2^{.448n})$ with high probability. First we need a few definitions. For our purpose A CNF Boolean formula F(x_1, x2,….., xn) is viewed as both a Boolean function and a set of clauses. We say that F is a k-CNF if all the clauses have size at most k. for a clause C, We write vars (C) for the set of variables appearing in C. If v ε vars(C), the orientation of v is positive if the literal v is in C and is negative if $\bar{w}$ is negative. The following simple subroutine takes as input an arbitrary assignment and tries to modify it to a satisfying assignment of formula f by considering the variables one by one in the order given by permutation π

Procedure **Modify** (CNF formula G(x_1, x_2,….,x_n),
Permutation π of {1, 2…n}, assignment y)
G_0=G
For i=1 to n

If G_{i-1} contains the unit clause x $\pi(i)$
Then $u_{x(i)}$ =1

Else if G_{i-1} contains the unit clause $\bar{x}$ $\pi(i)$

Then u $\pi(i)$ =0

Else u $\pi(i)$ =y $\pi(i)$

G_i= $G_{i-1}\left|\ x_{\pi(i)=u_{\pi(i)}}\ \right|$

[End of for loop]
Return u;
[End of Procedure Modify]

The algorithm Search is obtained by running Modify (G,π,y) on many pairs (π,y) where π is a random permutation and y is a random assignment.

Procedure **Search** (CNF-formula F, integer I)
Repeat I times
π = uniformly random permutation of 1…n
y= uniformly random vector $\in (0,1)^n$
u= Modify (F, π , y);
If u satisfies F

Then output (u); exit;
End of loop
Output ('unsatifiable')

End of Procedure Search

The algorithm Search was analyzed and summarize the results in theorem 1. The algorithm we investigate here is obtained by combining Search with a preprocessing step consisting of bounded resolution.
Resolve (CNF Formula F, integer s)
F_s=F.
While F_s has an S-bounded resolve pair C_1, C_2
With R (C_1, C_2).
Return (F_s)
We analyze the following simple combination of Resolve and Search
ResolveSat (CNF-formula F, integer s, positive integer I)
F_s=Resolve (F, s)
Search (F_s, I)

The algorithm Search was analyzed it is easily seen that Search(F,I) runs in time I/F/poly(n).it is also clear that Search(F,I) always answer unsatisfiable if F is unsatisfiable and tha problem of interest is to upper bound the error probability in the case that F is satisfiable. For a formula F and assignment z write $\tau(F,z)$ to be the probability over random π and y that Modify (F, π, y) returns the assignment z. Define $\tau(F)$ to be the sum of $\tau(F,z)$ over z that satisfy f i.e. τ (F) is the probability that Modify (F, π ,y) finds some satisfying assignment.

Theorem 1: For any satisfiable k-CNF formula F on n variables $\tau(F) \geq 2^{-(1-1/k)n}$. Thus the algorithm Search with $I=2^{(1-1/k)n}$ has the error probability O (e^{-n}) and runs in time $2^{(1-1/k)n}$ poly(n)

III. Form Of DNA Molecules And Operations

Our algorithm requires 2n + 3 well-behaved sequences of DNA:

1. A header sequence, h.
2. A separator sequence, s.
3. A primer sequence, p.
4. n "true" sequences, each denoted x_i^T representing the assignment " x_i=true".
5. n "false" sequences, each denoted x_i^F , representing the assignment "xi =false".

The algorithm requires synthesis of 2n assignment sequences which are used to append variable assignments to solution strands. There is a true and false sequence for each variable x_i,

Fig. 2 Structure of b_2^F assignment sequence for "x2 = false".

In the Fig. 2 Each box (s, p, $\bar{s}$ and so on) represents a DNA subsequence. $\bar{s}$ is the sticky end that anneals to the s sticky end on the solution strand during an APPEND.

The allowable operations are now:

1. APPEND (t, {s_1, s2 … s_k}): append to the end of each strand in tube t one of the subsequences s_1, s2… s_k. at random. This operation is a generalization of the standard append introduced by [5].We use this to append one variable assignment at a time.
2. u ← combine(t_1,t_2..... t_k): combine the contents of tubes t_1 through t_k into a single tube u. Tubes t_1,..... t_n are left empty (unless, of course, u = t_i for some 1 <= i <= k).
3. DETECT (t): select one strand at random from tube t, if any, and sequence it.
4. u ← extract(t, s): extract from tube t all strands containing the subsequence s

   and place them in tube u.
5. {u_1, u2… U_k) ← POUR (t): pour out, or aliquot, the contents of t into k

   equal portions in test tubes u_1 through u_k. Tube t is left empty.
6. TO-DOUBLE-STRANDED (t): make each of the single-stranded molecules

   in tube t double-stranded except for a sticky end as shown in Fig. 3.
7. TO-SINGLE-STRANDED (t): denature each double-stranded molecule in tube t and remove one strand, leaving the other as a single-stranded molecule in t.

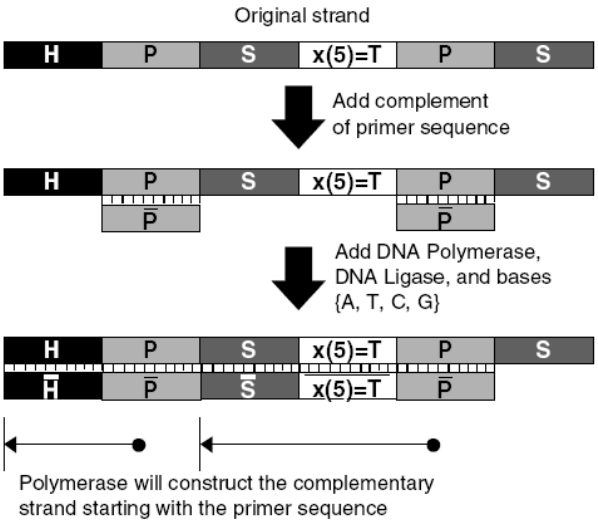


Fig. 3 Implementation of operation TO-DOUBLE-STRANDED.

The reason for the increased number of operations used in our model is that in our algorithm, the assembly of potential solution strands is very involved whereas this step in most other algorithms is relatively easy. However, all the biotechniques required to implement these operations are standard procedures used in other extract-based DNA algorithms. Finally, we note that our measure of time complexity will be the number of extract steps in the computation, and the space complexity will be the number of strands in the system.

IV. Results

The running time of ResolveSat (F, s, I) can be bounded as follows. Resolve(F,s) adds at most $O(n^s)$ clauses to F and can be implemented easily in time n^{2s} /F/poly(n). Search (F_s, I) runs in time $I(|F| + n^s)$poly(n). Hence the overall running time of resolveSat(F,s,I) is crudely bounded form above by $I(|F| + n^s)$poly(n). The simulated results are satisfactory and gives the indication of applicability of DNA computing for solving the FPGA Routing problem.

V. Conclusion

This paper has proposed a faster approach for finding the FPGA Routing solution using DNA Computing. Because the DNA Computing, due to its high degree of parallelism, can overcome the difficulties that may cause the problem intractable on silicon computers, however using DNA computing principles for solving simple problems may not be suggestible. To make the DNA computing applicable in practice further research in both fields- Computer science and biology – is necessary. Computer science needs to develop more elaborate DNA algorithms, while better enzymes and protocols are needed to from biology to manipulate DNA molecules more selectively with minimal errors.

References

[1] Eliezer L. Lozinskii, Impurity: Another phase transition of SAT, Journal on Satisfiability, Boolean Modeling and Computation, vol. 1, 2006, pp. 123-14

[2] Gi-Joon Nam, K. A. Sakallah, R. A. Rutenbar, A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 21, Issue 6, June 2002 pp. 674 – 684.

[3] E. Bach, A. Condon, E. Glaser and C. Tanguay, DNA models and algorithms for NP-complete problems, Proceedings of the 11th Annual IEEE Conference on Computational Complexity, March 1996, pp. 290.

[4] N. Jonoska and S. A. Karl, A molecular compututions of the Road Coloring problem, Proceedings of the 2nd Annual Meeting of DNA based computers, 1996.

[5] Leonard M. Adleman, "Computing with DNA", Scientific American, August 1998.

Manpreet Singh received the B.Tech. Electronics & Electrical Communication from Guru Nanak Dev Engineering College, Ludhiana and M.Tech. in Computer Science & Engineering from P. A. U., Ludhiana. He is presently working with Department of CSE & IT, Guru Nanak Dev Engineering College, Ludhiana. His current research interests are Bioinformatics, Distributed Computing and Data Mining. He has published around 20 research papers in various National and International conferences.

Parvinder Singh Sandhu is working as Professor in the Department of Computer Science and Engineering with Rayat & Bahra Institute of Engineeirng & Bio-technology, Sahauran, Mohali and previously he was with Guru Nanak Dev Engineering College, Ludhiana (Punjab). He is Master of Engineering in Software Engineering (Thapar University, Patiala), M.B.A. and Bachelor in Computer Engineering from National Institute of Technology (NIT), Kurukshetra. He has published 18 research papers in referred International journals and 17 papers in renowned international conferences. His current research interests are Software Reusability, Bio-informatics, Software Maintenance and Machine Learning.