Reducing Reconfiguration Time of Reconfigurable Computing Systems in Integrated Temporal Partitioning and Physical Design Framework

Farhad Mehdipour[1], Morteza Saheb Zamani[1], Hamid Reza Ahmadifar[3], Mehdi Sedighi[1] and Kazuaki Murakami[2]

[1]Amirkabir University of Technology
IT and Computer Engineering Department
#424 Hafez Ave., Tehran, Iran
{mehdipur, szamani, msedighi}@ce.aut.ac.ir

[2]Kyushu University
Dep. of Informatics
Graduate School of Information Science and Electrical Engineering
Fukuoka, Japan
murakami@i.kyushu-u.ac.jp

[3]Guilan University
Engineering Faculty
Rasht, Iran

Abstract

In reconfigurable systems, reconfiguration latency is a very important factor impact the system performance. In this paper, a framework is proposed that integrates the temporal partitioning and physical design phases to perform a static compilation process for reconfigurable computing systems. A temporal partitioning algorithm is proposed which attempts to decrease the time of reconfiguration on a partially reconfigurable hardware. This algorithm attempts to find similar single or pair of operations between subsequent partitions. Considering similar pairs instead of single nodes brings about less complexity for routing process. By using this technique, smaller reconfiguration bit-stream is obtained, which directly decreases the reconfiguration overhead time at the run-time. A complementary algorithm attempts to increase the similarity of subsequent partitions by searching for similar pairs and using a technique called dummy node insertion. An incremental physical design process based on similar configurations produced in the partitioning stage improves the metrics over iterations.

1. Introduction

Recent advances in the programmable hardware and architecture have resulted in the introduction of reconfigurable computing systems (RCS) [1, 3]. These systems usually consist of a host processor connected to a reconfigurable hardware, like a field programmable gate array (FPGA) [2, 3]. Reconfigurable systems offer a compromise between the performance advantages of fixed functionality of ASIC and the flexibility of general-purpose processors [13]. Currently, it appears that general-purpose CAD tools for reconfigurable computing systems that support design and implementation of desired applications are not readily available. Absence of appropriate design methodology and appropriate compiler and long reconfiguration time of programmable devices are some of the main challenges in reconfigurable computing [6, 7].

To implement a large circuit on an FPGA, it has to be partitioned into multiple stages. Then, the configurations of the FPGA are swapped in and out to implement each stage one by one and perform the function of the original circuit. This type of partitioning is known as temporal partitioning [14]. Using partially reconfigurable devices, parts of the design can be replaced while other parts are still active. This is useful in systems that must implement many modules at different periods of time on a device [20].

The focus of this work is physical design as well as temporal partitioning. Two major phases of physical design are placement and routing. Physical design for reconfigurable computing systems is often done according to traditional placement and routing algorithms used for FPGAs [18]. In the placement phase, optimal positions of modules on the target device should be determined. Minimizing the connection length, area and the longest wire are some of the main objectives in this process [18]. Simulated annealing has long been one of the most

1-4244-0054-6/06/$20.00 ©2006 IEEE

successful placement algorithms. Routing is the process of identifying exactly, which routing segments and switches should be used to create connected paths from net sources to net destinations for all nets in a circuit. Global and detailed routing should be done after placement for creating the routes between modules [4].

In this paper, a new temporal partitioning algorithm for partitioning and scheduling is proposed which tries to increase similarity of subsequent configurations in such a way that the time of reconfiguration on a partially reconfigurable hardware decreases**.** In this version of temporal partitioning algorithm, the similarity of node pairs and interconnection between them are taken into account. The similarity of the interconnections similar between node pair can largely reduce the reconfiguration time because of the major effect of routing stage in compilation process. Integration of temporal partitioning as a post synthesis stage and physical design as a low level process is one of the main contributions of this work.

We explain temporal partitioning and physical design phases and related works in Section 2. New similarity-based temporal partitioning algorithms, which exploit the single node and node pair similarity, are proposed in Section 3. Section 3 explains also the details of dummy node insertion technique as a new method for increasing node pair similarity in subsequent configurations. Section 4 explains the integrated temporal partitioning and physical design approach. In Section 5, the details of our framework are explained and experimental results are presented. Finally, Section 6 concludes the paper.

2. Related Works

Temporal partitioning can be stated as partitioning a data flow graph into a number of partitions such that each partition can fit in the device and also, dependencies among the graph nodes are not violated. For a partially-reconfigurable hardware, parts of the hardware can be programmed without disturbing the rest of the design since common parts of two successive configurations can remain unchanged.

Karthikeya et al. [10] proposed algorithms for temporal partitioning and scheduling of large designs on area constrained reconfigurable hardware, but do not consider the reconfiguration time overhead. Bobda [6] proposed two methods to solve temporal partitioning problem. The first one is an enhancement of the well-known list vector space. The second method uses a spectral placement to position the modules in a three-dimensional space. In [21], Spillane and Owen have focused on finding a sequence of conditions for activating an appropriate component at a particular time and optimization successive configurations to achieve the desired trade-offs among reconfiguration time, operation speed and area. SPARCS [9] is an integrated partitioning and synthesis framework which has a temporal partitioning tool to temporally divide and schedule the tasks on a reconfigurable system. SPARCS does not perform physical design with respect to the partitions generated by the temporal partitioner. Luk et al. [12] proposed a methodology to take advantage of common operators in successive partitions. It attempts to reduce configuration time and thus the application execution time. This model does not consider timing aspects and does not perform any partitioning but Tanougust et al. [22] attempt to find the minimum area while meeting timing constraints.

In our previous work [15], a similarity based partitioning algorithm was proposed which addresses the long reconfiguration time of the programmable hardware. The proposed algorithm finds modules with identical functionality in a data flow graph and then attempts to increase the similarity of adjacent configurations. This results in a smaller placement time for similar partitions in FPGA at the compilation process and shorter reconfiguration time. Our proposed algorithm is performed at the design time and therefore, there is enough time to explore for the optimality of design criteria.

We develop a static compiler and use traditional iterative placement and routing algorithms for producing high quality placed and routed configurations. We integrate temporal partitioning and physical design and develop a framework, which performs partitioning taking design performance into account. None of the above approaches proposed physical design algorithms to be performed after temporal partitioning. In this work pair similarity based algorithms as a complementary algorithm for temporal partitioning is proposed which attempt to increase the similarity of partitions by using a new technique called dummy node insertion.

3. Similarity-Based Temporal Partitioning Algorithms

In our previous work [15], A new factor, namely similarity value, has been defined, which determines the level of similarity (in terms of the functionality of their nodes) between two succeeding partitions. Our main goal is to reduce the reconfiguration time and overall run-time of applications and the area needed for their implementation. We assume that the target programmable device is partially programmable. In this section, first, we represent a temporal partitioning algorithm, which works based on the similarity factor and considers the single node similarity [15]. Then, the similarity of node pairs is taken into account to consider the interconnection between nodes and reducing the routing process complexity and reconfiguration time.

The proposed algorithm in [15] takes a data flow graph (DFG), the nodes of which represent pre-designed firm modules in a library. In order to ensure that all computations will be performed correctly when the circuit is decomposed into stages, certain temporal constraints must be satisfied [14]. For example, a node can be executed if all of its predecessors have already been executed. In an algorithm, in the first stage, level assignment is performed according to as soon as possible (ASAP) algorithm [6, 16]. ASAP schedules a data flow graph in an attempt to minimize latency by topologically sorting of the nodes of the graph. In the partitioning stage of DFG, the level number of modules, their sizes and the size of target hardware are the most important factors which should be considered. After generating initial partitions, a complementary iterative algorithm tries to increase the similarity between two successive partitions.

3.1. Node Pair Similarity Algorithm

By using the algorithm presented in [15], only non-similar parts of configurations have to be placed and routed. This brings about the reduction in time needed to generate configurations and also in run-time reconfiguration latency.

Routing is a time consuming process at the compilation phase and also in routing resources reconfiguration phase. Generally, about 70-90% of configuration bit-stream relates to routing resources and interconnections. In addition, 80% of path delay is related to interconnections [24]. In this section, a new temporal partitioning algorithm is proposed, which considers the node pairs and the interconnections between them as a basic component to apply the similarity-based algorithm but first, the definitions of some terms and symbols are presented.

Definition 1: For a node *i* in a DFG, *F(i)* represents a module in the module library with the same functionality as the node *i.*

Definition 2: For two nodes *I* and *j* in a DFG, and a partition P_k of the DFG, *NP(i,j,k)* is a *Node Pair* where $i \in P_k$ and $j \in P_k$ and *i* and *j* represent two nodes in the DFG with an edge from *i* to *j*.

Definition 3: For two node pairs $NP(i_1, j_1, k)$ and $NP(i_2, j_2, k+1)$ in a DFG, and two consecutive partitions P_k and P_{k+1} of the DFG, $SNP(i_1, j_1, i_2, j_2, k)$ is a *Similar Node Pair* where $i_1, j_1 \in P_k$ and $i_2, j_2 \in P_{k+1}$ and $F(i_1) = F(i_2)$ and $F(j_1) = F(j_2)$.

Definition 4: *Similar Node Pair Set (SNPS)* of partition P_k is defined as the set of all *Similar Node Pairs* in P_k and P_{k+1}:

$$SNPS(P_k) = \{SNP(i_1, j_1, i_2, j_2, k) \left| \begin{array}{l} \forall i_1, j_1 \in P_k \\ and \quad \forall i_2, j_2 \in P_{k+1} \end{array} \right. \}$$

Definition 5: *Partition Pair Similarity Value (PPSV)* for P_k is the number of similar pairs between two subsequent partitions k and k+1: $PPSV(P_k) = |SNPS(P_k)|$

Definition 6: *Graph Pair Similarity Value* is the number of total similar pairs in the DFG.

$$GPSV = \sum_{k=1}^{N-1} PPSV(P_k)$$, *N* is the number of partitions

Definition 7: *Dummy Node* is a node, which performs no operation such as addition to 0 or multiplication by 1 and does not affect the running sequence of DFG.

In the proposed pair similarity based algorithm, first, input DFG is partitioned using previous temporal partitioning algorithm, which considers the reconfiguration overhead time and area constraints. Then node pairs list in each partition is created and similar pairs in subsequent configurations is found.

In Figure 1(a), a DFG is shown. In Figure 1(b), this DFG is partitioned and node pairs for each partition are shown. Figure 1(c), shows the single node similarity in subsequent partitions and Figure 1(d), shows the similar node pairs extracted from the DFG.

3.2. Dummy Node Insertion Algorithm

In the configurations generated by the compilation process, some wasted area is usually produced. This area may be large enough to insert extra small modules. In this case, small modules can be selectively added to each partition to increase node pair similarity. A new node should be dummy node which performs a null operation. Dummy node insertion algorithm is as follows:

1. Perform temporal partitioning according to temporal partitioning algorithm in [15].
2. Create node pair list for each partition.
3. For *k= 0* to *n-1* (*n* is the number of partitions)
   - 3.a. Select a node pair *NP(i, j, k)* from the list of node pairs in P_k .
   - 3.b. Select node $t \in P_{k+1}$ in such a way that *t* is similar to node $i \in P_k$ (i.e. *F(i)=F(t)*). If there is not any node in P_{k+1} similar to *i,* repeat steps 3.a and 3.b
   - 3.c. Add a dummy node *l* to P_{k+1} similar to node $j \in P_k$.
   - 3.d. Compute total configuration area plus the memory required for intermediate data between P_k and P_{k+1} in partition P_{k+1} .
   - 3.e. If the space required by partition P_{k+1} is smaller than the target hardware area, then commit the node insertion operation and perform necessary changes in dependencies between nodes of partition P_{k+1} . In this case, *NP(t,l,k)* is added to $SNPS(P_k)$ and so, $PPSV(P_k)$

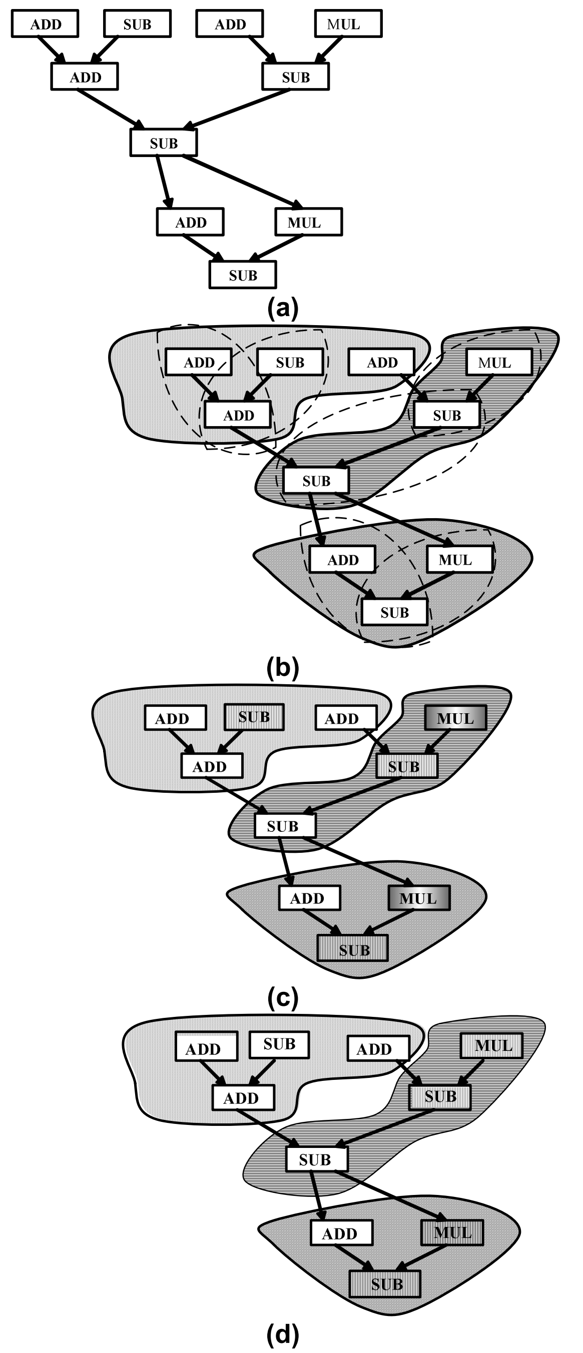


Figure1.(a) A task graph, (b) a partitioned task graph and node pairs, (c) single similar nodes and (d) similar node pairs.

and *GPSV* increases by 1.

3.f. If the space required by partition P_{k+1} is larger than the target hardware area, then remove the dummy node l from P_{k+1}. In this case, *PPSV(P$_k$)* and *GPSV* remain unchanged.

3.g. Repeat the steps 3.b to 3.f for all node pairs in partition P_k.

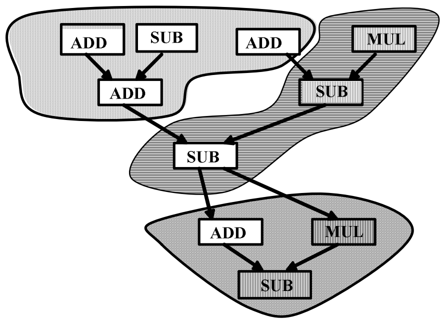
In Figure 2(a), a partitioned DFG is shown. Only one similar pair (MUL, SUB) exists in the second and third partitions, so *PPSV(P$_2$)= 1* and *GPSV= 1*. Insertion of a dummy node ADD to the second partition adds a new node pair (SUB, ADD) to it (Figure 2(b)). This pair is similar to the pair (SUB, ADD) in the first partition. In addition, insertion of a dummy node SUB in the third partition adds another new pair (SUB, SUB) to the second and the third partitions. Therefore, the number of similar pairs in the DFG increases by 2 ($GPSV_{new}$= *3*).

3.3. Choosing Between Single and Pair Similarity

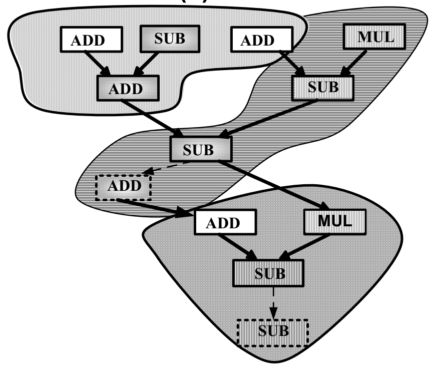
The single and the pair similarity algorithms were presented for temporal partitioning. In the former case, the number of similar nodes between subsequent configurations is considered. This similarity decreases the placement time and does not affect the routing process. On the other hand, in the latter case, the interconnection between modules is also taken into account. Therefore, similar nodes in two adjacent partitions remain unchanged during placement. In addition, routing resources between the two nodes remain unchanged during routing. Thus, by using pair similarity instead of single node similarity, complexity and the time of both placement and routing processes are reduced. In addition, during run-time reconfiguration, reconfiguration latency decreases accordingly.

In the single similarity case, the part of the bit-stream which is related to the placement of similar modules is not reloaded into the hardware and the reconfiguration time of logic blocks decreases. On the other hand, in the pair similarity case, both the placement and routing part of the bit-streams related to the similar node pair and their interconnections are not need to be loaded into the hardware. Therefore, the time needed to reconfiguration logic blocks and routing resources decrease.

Assume that *S* is the single node similarity value and *P* is the node pair similarity value in DFG. There are *S* modules, whose positions remain unchanged in the subsequent configurations. The value of *S* affects the placement process time in compilation and reconfiguration phases. On the other hand, there are *2*x*P* similar nodes and *P* similar interconnections in the configurations. Therefore, the placement of *2*x*P* modules plus the routing of *P* connection wires are not performed and also reconfiguration time of *2*x*P* logic blocks and some parts of routing resources are not needed to perform in run-time reconfiguration phase.



(a)



(b)

Figure 2. (a) A partitioned task graph that have one similar pair and (b) dummy node insertion algorithm adds another similar pair to the second partition

We assume that γ is interconnection efficiency, which describes the interconnection routing weight relative to the placement of logic blocks. For example, if we assume that γ is set to 2, it represents that interconnection routing between two modules has the same effect as the placement of two modules. We choose one of the single or pair similarity methods for generating partitions. If the overall similarity value of single similarity method (S) is greater than $2 \times P + \gamma \times P$, then we choose the single similarity method; otherwise we choose the pair similarity method.

4. Integrated Temporal Partitioning and Physical Design

The proposed approach focuses on physical design stage as well as temporal partitioning. We attempt to use benefits of our greedy temporal partitioning approach in physical design process to achieve better performance in a reconfigurable system. VPR is one of the popular tools for placement and routing of FPGA designs [4, 5]. We used this tool as an appropriate option for developing our framework. VPR uses traditional FPGA placement and routing algorithms but we have modified it for the placement and routing of each configuration generated by our temporal partitioner. VPR uses simulated annealing for placement. Our temporal partitioning algorithm generates a number of configurations in the first stage. Then, we add input and output registers as data memories for transferring data between successive configurations. The memory usage of each configuration depends on the number of output signals used in the succeeding configuration [15].

The number of memory cells, which must be added to the netlist of each partition, is equal to the memory usage of that partition. A netlist is then generated for each configuration and the modified VPR is applied to it. VPR can generate the final configurations on a general island-style FPGA [23, 4]. This programmable device has an island-style architecture, which contains a square array of logic blocks called configurable logic blocks (CLBs) embedded in a uniform mesh of routing resources. The FPGA CLBs contain one or more Look-up Tables (LUTs), that can be programmed to perform any logic function of a small number of inputs (typically 4-5), a small number of simple logic gates and one or more flip-flops [24].

We modified some parts of the placement algorithm used in VPR. In our tool, after generating the first configuration, the placement of subsequent partitions is performed incrementally. According to our design flow, common blocks in two subsequent configurations are fixed and remain unchanged during the placement phase of the second configuration. Since an incremental placement algorithm is performed, swapping and moving fixed blocks should be avoided. In this way, the run time of placement reduces, accordingly.

5. Experimental Results

In our developed framework, the nodes in the input data flow graphs are firm-modules. A library consisting of the required firm modules was developed. Figure 3 illustrates the CAD flow we used for generating firm-modules. First, each module was described in VHDL and was then synthesized by Leonardo Spectrum synthesis tool to obtain a structural description of the module based on logic gates. The SIS synthesis package [17] was used to perform technology-independent logic optimization of each module circuit. Next, each circuit was technology-mapped into 4-LUTs and flip flops by FlowMap [8]. The output of FlowMap is a netlist of LUTs and flip flops in .blif format. T-VPack [4, 5] then packed this netlist of 4-LUTs and flip flops into more coarse-grained logic blocks, and generated a netlist in .net format. VPR [5] then placed and routed the module.

In this way, the DFG nodes were generated as firm-modules and added to our library. The architecture of the target programmable device was chosen to be a general island-style FPGA. We assume that it is a partially-reconfigurable device. To our knowledge, there is not any common use or standard benchmarks for static data flow graphs. We chose six static data flow graphs and applied our tool to them. The first five of them were selected from [6] and [11]. The sixth one was a data flow graph for FEAL cryptography algorithm [19].

Input DFGs were implemented in a reconfigurable hardware. Now, we show how our similarity based temporal partitioning and incremental physical design can affect the overall performance of the application in a reconfigurable system. Table 1 shows the results of incremental and non-incremental implementations regarding different criteria, such as placement cost (in terms of wire length [3]), critical path delay and the number of routing channels used.

The quality of configurations produced by the incremental approach is comparable with those obtained using the non-incremental approach but the incremental approaches produce the results in a shorter time. In other words, increasing the similarity of configurations and using the incremental approach resulted in less placement cost and usually higher speed for the configurations produced. Table 1 also shows that for some of the DFGs, the number of required routing channels decreases because of the better quality of placement achieved in the incremental approach.

We assume that the reconfiguration time can be approximated by a linear function of the total area of functional units being downloaded, which is realistic in practice. The full configuration time is constant for a particular FPGA. For the DFG we used for our experiments, run time of each configuration is much less than a microsecond, whereas the full reconfiguration of a programmable device is typically done in several microseconds.

Therefore, reducing the reconfiguration time decreases the overall run time of the application accordingly. To evaluate our incremental approach, we attempted a non-incremental placement and routing of the designs after the similarity based temporal partitioning.

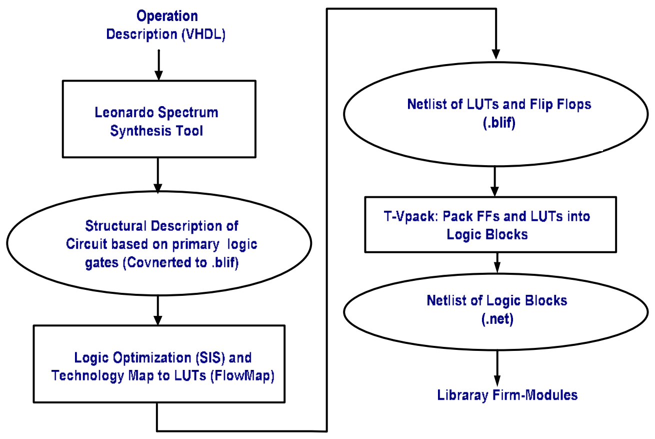

Operation Description (VHDL)
Leonardo Spectrum Synthesis Tool
Structural Description of Circuit based on primary logic gates (Covnerted to .blif)
Logic Optimization (SIS) and Technology Map to LUTs (FlowMap)
Netlist of LUTs and Flip Flops (.blif)
T-Vpack: Pack FFs and LUTs into Logic Blocks
Netlist of Logic Blocks (.net)
Libraray Firm-Modules

Figure 3. Generating library cells in .net format

Table 1. Results of incremental and non-incremental compilation

Data flow Graph	Non-Incremental			Incremental		
	Placement Cost	Critical Path Delay (nsec)	Channel Width Used for Routing	Placement Cost	Critical Path Delay (nsec)	Channel Width Used for Routing
DFG1	2.64	29.34	3	2.59	36.48	3
DFG2	16.33	64.67	5	14.5	64.67	5
DFG 3	1.05	8.44	3	0.97	7.84	2
DFG 4	16.50	24.33	4	16.20	27.34	2
DFG 5	42.58	72.29	5	41.35	70.01	5
DFG 6 (FEAL)	4.54	22.25	3	4.43	13.44	3

In other words, for a DFG, the placement and routing of each configuration are performed from the scratch in an attempt to obtain better results. For all the DFGs attempted, the reconfiguration time is reduced due to some common and similar parts in subsequent configurations.

Preliminary experiments were performed according to the single similarity algorithms. Figure 4, shows the design flow used to generate configurations based on single and pair algorithms. Input DFG is partitioned and fed to both of two single and pair similarity algorithms. According to approach mentioned in Section 3.4, one of the two outputs is selected and then the partitions generated are used to generate the netlist of each partition. In final step, placement and routing of each netlist is performed and final configurations are generated.

Two single and pair similarity algorithms are compared in Table 2. Experiments show that the dummy node insertion algorithm can increase the number of similar pairs in the generated configurations. Furthermore, the time needed to run the placement stage is reduced accordingly because the CLB's in similar operations have fixed positions. According to [24], we assume that the routing resources bit-stream is at least 70% of the total bit-stream size. Table 3 shows the effect of single similarity algorithm on the placement time, as well as the effect of pair similarity on the placement and routing time at the compilation phase. In addition, reconfiguration speedup is shown in Table 3 for both of the similarity-based algorithms.

6. Conclusion

In this paper, a framework was proposed for the static compilation of reconfigurable computing systems. This framework integrates the temporal partitioning and physical design stages. Similarity-based algorithms proposed as complementary algorithms to temporal partitioning which attempt to increase the similarity of adjacent partitions and so, decrease the physical design process time and the latency of run-time reconfiguration overhead. Pair similarity-based algorithm considers two nodes and the interconnection between them to decrease the routing time in the compilation stage and run-time reconfiguration phase. Using a dummy node insertion technique brings about more similarity between adjacent partitions.

Our experiments show that the presented algorithm increases the similarity of partitions, and consequently, the hardware reconfiguration time decreases accordingly. An iterative incremental approach, developed for the placement stage, generated good quality configurations in a shorter time comparing with the non-incremental algorithm. In addition, this approach reduced the total application run-time.

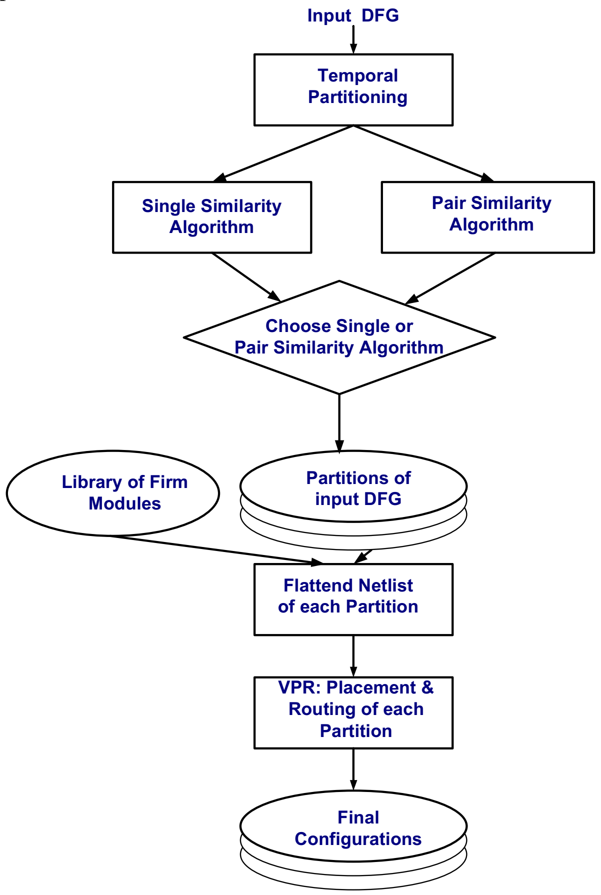

Input DFG
Temporal Partitioning
Single Similarity Algorithm
Pair Similarity Algorithm
Choose Single or Pair Similarity Algorithm
Library of Firm Modules
Partitions of input DFG
Flattend Netlist of each Partition
VPR: Placement & Routing of each Partition
Final Configurations

Figure 4. Stages of final configurations generation by using the single and pair similarity algorithms

References

[1] Barr M, A reconfigurable computing primer, Miller Freeman Inc., 1998.

[2] Bazargan K and Kastner R and Sarrafzadeh M, Fast template placement for reconfigurable computing systems, IEEE Design and Test of Computers, January-March 2000, pp. 68-83.

[3] Bazargan K and Orgenci M and Sarrafzadeh M, Integrating Scheduling and Physical Design into a Coherent Compilation Cycle for Reconfigurable Computing Architectures, Design Automation Conference (DAC), 2001, pp. 635-640.

[4] Betz V and Rose J and Marquardt, Architecture and CAD for deep-submicron FPGAs, Kluwer Academic Publishers, 1999.

[5] Betz V, VPR and T-VPack1 user's manual (Version 4.30), http://www.eecg.toronto.edu/~vaughn, 2000.

[6] Bobda C, Synthesis of dataflow graphs for reconfigurable systems using temporal partitioning and temporal placement, Ph.D thesis, Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, 2003.

[7] Compton K and Hauck S, Reconfigurable computing: A survey of systems and software, ACM Computing Surveys, 34 (2) (2002) 171-210.

[8] Cong J and Ding Y, Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, IEEE Transactions on CAD (1994)1-12.

[9] Govindarajan S and Ouaiss I and Kaul M and Srinivasan V and Vemuri R, An effective design approach for dynamically reconfigurable architectures, IEEE Symposium on FPGAs for Custom Computing Machines, 1998, pp.312-320.

[10] Karthikeya M and Gajjala P and Bhatia D, Temporal partitioning and scheduling data flow graphs for reconfigurable computers, IEEE Transactions on Computers, 48 (6) (1999) 579-590.

[11] Kastner R and Kaplan A and Sarrafzadeh M, Synthesis techniques and optimizations for reconfigurable systems, Kluwer-Academic Publishers, 2004.

[12] Luk W and Shirazi N and Cheung P Y K, Modeling and optimizing run-time reconfiguration systems, in:K.L. Pocek, J. Arnold (Eds.), Proceedings of IEEE Symposium on FPGA's Custom Computing Machines, IEEE Computer Society Press, 1996, pp.167-176.

[13] Maestre R and Kurdahi F J and Bagherzadeh N and Singh H and Hermida R and Fernandez, M, Kernel scheduling in reconfigurable computing, Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 1999, pp.90-96.

[14] Mak W.K and Young E.F.Y, Temporal logic replication for dynamically reconfigurable FPGA partitioning, IEEE Transactions of Computer-Aided Design of Integrated Circuits and Systems, 22(7):952-959, 2003.

[15] Mehdipour F and Saheb Zamani M and Sedighi M, A New Iterative Design Flow for Static Compilation of Reconfigurable Computing Systems, Proceedings of the 10th International Symposium on Integrated Circuits, Devices and Systems, Singapore, 2004.

[16] Micheli G.D, Synthesis and optimization of digital circuits, McGraw-Hill, 1994.

[17] Sentovich E M, SIS: A system for sequential circuit analysis, Tech. Report No.UCB/ERLM92/41, University of California, Berkeley, 1992.

[18] Sherwani N, Algorithms for VLSI physical design automation, Kluwer-Academic Publishers, Third Edition, 1999.

[19] Shimizu A and Miyaguchi S, Fast data encipherment algorithm FEAL,Transaction of IECE of Japan, J70-D (7) (1987)1413-1423.

[20] Shirazi N and Luk W and Cheung P Y K, Automating production of run-time reconfigurable designs, IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1998, pp. 147-156.

[21] Spillane J and Owen H, Temporal partitioning for partially reconfigurable field programmable gate arrays, IPPS/SPDP Workshops, 1998, pp. 37-42.

[22] Tanougast C and Berviller Y and Brunet P and Weber S and Rabah H , Temporal partitioning methodology optimizing FPGA resources for dynamically reconfigurable embedded real-time system, Microprocessors and Microsystems, 27 (2003)115-130.

[23] Tessier R, Fast place and route approaches for FPGAs, PhD thesis, Massachussetts Institute of Technology, 1999.

[24]www.cs.caltech.edu/research/ic/transit/rcgp/chapter1.6.1.htm

Table 2. Comparison of single and pair similarity algorithms

Data flow Graph	Single Similarity Alg.		Pair Similarity Alg.	
	No. of similar single nodes (before running similarity-based algorithm)	No. of similar single nodes (after running similarity-based algorithm)	No. of similar pairs (before dummy node insertion algorithm)	No. of similar pairs (after dummy node insertion algorithm)
DFG1	0	2	1	2
DFG2	3	3	0	0
DFG 3	0	0	0	0
DFG 4	4	4	0	2
DFG 5	5	5	0	2
DFG 6 (FEAL)	9	11	5	7

Table 3. Single and pair similarity algorithms effects on compile time and reconfiguration speedup

Data flow Graph	Single Similarity Alg.		Pair Similarity Alg.		
	Placement time improvement	Reconfiguration speedup	Placement time improvement	Routing time improvement	Reconfiguration speedup
DFG1	10%	1.03	13.7%	14%	1.16
DFG2	5.8%	1.02	0.0%	0.0%	0
DFG 3	0.0%	0.0	0.0%	0.0%	0
DFG 4	50%	1.17	48%	13.5%	1.31
DFG 5	64%	1.24	18%	7.5%	1.12
DFG 6 (FEAL)	37.5%	1.12	58%	16.6%	1.32